

# 5280

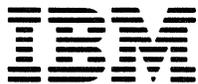


SC21-7790-1  
S5280-21

## **IBM 5280 Distributed Data System**

**Assembler Language Reference Manual**

Program Number 5708-AS1



SC21-7790-1  
S5280-21

# **IBM 5280 Distributed Data System**

**Assembler Language Reference Manual**

Program Number 5708-AS1

**Second Edition (April 1981)**

This is a major revision of, and obsoletes, SC21-7790-0 and incorporates TNL SN20-9582.

Because the changes and additions are extensive, this publication should be reviewed in its entirety.

Changes are periodically made to the information herein; these changes will be reported in technical newsletters or in new editions of this publication.

Use this publication only for the purposes stated in the *Preface*.

This material may contain reference to, or information about, IBM products (machines and programs), programming, or services that are not announced in your country. Such references or information must not be construed to mean that IBM intends to announce such IBM products, programming, or services in your country.

Publications are not stocked at the address below. Requests for copies of IBM publications and for technical information about the system should be made to your IBM representative or to the branch office serving your locality.

This publication could contain technical inaccuracies or typographical errors. Use the Reader's Comment Form at the back of this publication to make comments about this publication. If the form has been removed, address your comments to IBM Corporation, Information Design and Development, Department 997, 11400 Burnet Road, Austin, Texas 78758. IBM may use and distribute any of the information you supply in any way it believes appropriate without incurring any obligation whatever. You may, of course, continue to use the information you supply.

This reference manual is intended for programmers who want to write programs for the IBM 5280 using the assembler language. The programmer is expected to either have previous experience using an assembler language or be familiar with the 3741 Application Control Language (ACL).

Using this publication, the programmer should be able to:

- Understand the general organization of main storage.
- Understand the purpose of each control statement and the proper order for using each control statement in an assembler program.
- Understand the purpose of each instruction and the proper order for using each instruction in an assembler program.
- Write a source program.
- Load the assembler program product into the IBM 5280 system, respond to the assembler prompts, assemble the source program, and write the object program to a diskette.
- Understand the assembly listing and cross reference listing.
- Debug the assembler source program to get an error-free listing.

Chapter 1 contains a general overview of how (1) a source program is written, (2) an object program is executed, and (3) main storage is organized. It also explains the coding conventions used in the assembler language and in this publication.

Chapter 2 discusses such programming concepts as tables, subroutines, formats, external status, and self-check computations. It describes data management for input and output operations.

Chapter 3 describes each control statement.

Chapter 4 describes each instruction.

Chapter 5 explains how to load the assembler program product and how to assemble a source program. It describes an assembly listing and cross-reference listing.

Chapter 6 describes control areas and functions. The functions include optional common functions.

Chapter 7 explains how to use the ACL to assembler language conversion program to convert ACL programs.

Appendix A lists the instruction mnemonics in alphabetic order and gives the op code and format for each mnemonic.

Appendix B describes SCS control codes.

Appendix C describes the computations generated by the .SELFCHK control statement parameters.

Appendix D consists of codes and charts, including EBCDIC charts and scan codes.

Appendix E lists all error codes for the assembler program and conversion program.

#### Related Publications

- *IBM 5280 General Information*, GA21-9350
- *IBM 5280 System Concepts*, GA21-9352
- *IBM 5280 Functions Reference Manual*, GA21-9353
- *IBM 5280 Message Manual*, GA21-9354
- *IBM 3270 Information Display System Component Description*, GA27-2749



|  |           |   |            |
|--|-----------|---|------------|
| <b>CHAPTER 1. INTRODUCTION</b> . . . . .         | <b>1</b>  | Pointer I/O . . . . .                                 | 37         |
| Overview of the Assembler Language . . . . .     | 1         | Keyed Data Sets . . . . .                             | 38         |
| The Control Statements . . . . .                 | 1         | Shared Data Sets . . . . .                            | 40         |
| The Instructions . . . . .                       | 2         | SCS Conversion Data Sets . . . . .                    | 40         |
| The Source Program Format . . . . .              | 2         | Extending Data Sets . . . . .                         | 40         |
| The Assembler Program . . . . .                  | 3         | Self-Check . . . . .                                  | 41         |
| Loading the Object Code . . . . .                | 3         | Choosing Your Algorithm . . . . .                     | 44         |
| Overview of Program Execution . . . . .          | 3         | Using the GSCK Instruction . . . . .                  | 47         |
| Overlapped I/O . . . . .                         | 4         | Using the IF . . . . .CHK Instruction . . . . .       | 47         |
| External Status . . . . .                        | 4         | <b>CHAPTER 3. ASSEMBLER LANGUAGE CONTROL</b>          |            |
| Data Input . . . . .                             | 4         | <b>STATEMENTS</b> . . . . .                           | <b>49</b>  |
| Data Manipulation . . . . .                      | 4         | Format . . . . .                                      | 51         |
| Data Output . . . . .                            | 5         | Blanks . . . . .                                      | 51         |
| Overview of Main Storage . . . . .               | 5         | Comments . . . . .                                    | 51         |
| Logical Device Identifiers . . . . .             | 6         | Initialize the Partition Control Area . . . . .       | 52         |
| Common Functions . . . . .                       | 6         | .START Control Statement . . . . .                    | 52         |
| Partitions . . . . .                             | 6         | .KBCRT Control Statement . . . . .                    | 53         |
| Modes of Operation . . . . .                     | 6         | .EDITC Control Statement . . . . .                    | 56         |
| Partition Control Area . . . . .                 | 8         | Declare and Label Data Areas . . . . .                | 57         |
| Indicators and Registers . . . . .               | 8         | .DC Control Statement . . . . .                       | 57         |
| Storage . . . . .                                | 11        | .DCLBR Control Statement . . . . .                    | 65         |
| Addressing Methods . . . . .                     | 11        | .DCLDR Control Statement . . . . .                    | 65         |
| Partition Work Area . . . . .                    | 12        | .DCLIND Control Statement . . . . .                   | 65         |
| Main Storage Boundary Alignment . . . . .        | 13        | .EQUATE Control Statement . . . . .                   | 66         |
| Blanks, Constants, and Coding Symbols . . . . .  | 14        | Set Up and Initialize Device Control Blocks . . . . . | 69         |
| Symbols Used in This Manual . . . . .            | 15        | .COMM Control Statement . . . . .                     | 69         |
| <b>CHAPTER 2. PROGRAMMING CONCEPTS</b> . . . . . | <b>17</b> | .DATASET Control Statement . . . . .                  | 72         |
| Tables . . . . .                                 | 17        | Set Up and Label Tables . . . . .                     | 79         |
| System Tables . . . . .                          | 17        | .TABLE Control Statement . . . . .                    | 79         |
| Data Tables . . . . .                            | 18        | .LABTAB Control Statement . . . . .                   | 81         |
| Label Tables . . . . .                           | 18        | .SYSTAB Control Statement . . . . .                   | 82         |
| Data Types . . . . .                             | 19        | Set Up Edit Formats . . . . .                         | 84         |
| Subroutines . . . . .                            | 19        | .FMTST Control Statement . . . . .                    | 85         |
| The Partition Subroutine Stack . . . . .         | 20        | .FMTFLD Control Statement . . . . .                   | 85         |
| Subroutine Returns . . . . .                     | 20        | .FMTEND Control Statement . . . . .                   | 92         |
| The Status Line . . . . .                        | 22        | Set Up Screen Control Formats . . . . .               | 92         |
| Nondisplay of the Status Line . . . . .          | 23        | .SFMTST Control Statement . . . . .                   | 97         |
| External Status and Error Conditions . . . . .   | 24        | .SFMTCTL Control Statement . . . . .                  | 98         |
| Keyboard Data Entry . . . . .                    | 25        | .SFMPMT Control Statement . . . . .                   | 100        |
| Keystroke Buffering . . . . .                    | 26        | .SFMCNS Control Statement . . . . .                   | 102        |
| Modes of Entry . . . . .                         | 26        | .SFMTFLD Control Statement . . . . .                  | 103        |
| Automatic Functions . . . . .                    | 28        | .SFMTEND Control Statement . . . . .                  | 106        |
| Auto Enter . . . . .                             | 29        | Field Type Keywords . . . . .                         | 107        |
| Auto Duplicate/Skip . . . . .                    | 29        | Field Definition Keywords . . . . .                   | 109        |
| Alternate Record Advance . . . . .               | 29        | Control the Assembly Listing . . . . .                | 113        |
| Screen Formats . . . . .                         | 29        | .TITLE Control Statement . . . . .                    | 113        |
| Prompts . . . . .                                | 30        | .EJECT Control Statement . . . . .                    | 114        |
| Constant Insert Data . . . . .                   | 31        | .SPACE Control Statement . . . . .                    | 114        |
| Field Definitions . . . . .                      | 31        | .PRINTON Control Statement . . . . .                  | 115        |
| Field Control . . . . .                          | 32        | .PRINTOFF Control Statement . . . . .                 | 115        |
| Returning (RG) Exits . . . . .                   | 34        | Miscellaneous Control . . . . .                       | 115        |
| Edit Formats . . . . .                           | 34        | .INCLUDE Control Statement . . . . .                  | 116        |
| Data Directed Formatting . . . . .               | 35        | .SELFCHK Control Statement . . . . .                  | 116        |
| Field Modification Indicators . . . . .          | 35        | .XTRN Control Statement . . . . .                     | 120        |
| Diskette Data Management . . . . .               | 36        | .END Control Statement . . . . .                      | 121        |
| Label Update . . . . .                           | 36        | <b>CHAPTER 4. IBM 5280 ASSEMBLER LANGUAGE</b>         |            |
| Physical and Logical Buffers . . . . .           | 36        | <b>INSTRUCTIONS</b> . . . . .                         | <b>123</b> |
| Automatic Logical Buffering . . . . .            | 36        | Instructions Format . . . . .                         | 123        |

|  |     |
|--|-----|
| Blanks   | 123 |
| Symbolic Labels                                | 123 |
| The Instruction Fields                         | 124 |
| Comments                                       | 124 |
| Storage Specifications                         | 124 |
| Labeled Addressing                             | 125 |
| Base Displacement Addressing                   | 125 |
| Constant Specifications                        | 125 |
| Register and Indicator Specifications          | 126 |
| Operation Types                                | 126 |
| Assembly Time Arithmetic                       | 132 |
| Arithmetic Expressions                         | 132 |
| The ADDR Function                              | 133 |
| The LENG Function                              | 134 |
| Changing a Declared Length                     | 134 |
| Changing a Data Type                           | 134 |
| Arithmetic/Logical Instructions                | 135 |
| Binary Register Arithmetic/Logical             | 135 |
| Binary Register Shift/Rotate                   | 139 |
| Decimal Register Arithmetic                    | 141 |
| Decimal Register Shift                         | 145 |
| Decimal Register Zone Modification             | 148 |
| Branch and Skip Instructions                   | 149 |
| Unconditional Branch                           | 149 |
| Subroutine Call and Return                     | 151 |
| Full Conditional Branch on Test                | 154 |
| Short Conditional Branch on Relational Compare | 158 |
| Skip on Constant Compare                       | 162 |
| Skip on Bit Mask                               | 164 |
| Skip on AND/Exclusive-OR Mask                  | 165 |
| Loop Control                                   | 166 |
| Communications Instructions                    | 168 |
| Diskette Instructions                          | 174 |
| Control Operations                             | 174 |
| Search Operations                              | 188 |
| Printer Instructions                           | 192 |
| Error Recovery Procedures                      | 197 |
| Keyboard and Display Instructions              | 199 |
| Key Entry Instructions                         | 199 |
| Keyboard Operations                            | 204 |
| Data Movement Instructions                     | 229 |
| Load Binary Register                           | 229 |
| Load Decimal Register                          | 231 |
| Store at a Labeled Address                     | 233 |
| Store at Base Displacement Address             | 235 |
| Exchange Data                                  | 236 |
| Convert Register Contents                      | 237 |
| Move Bytes Between Decimal Registers           | 240 |
| Move Bytes in Storage                          | 241 |
| Move Bytes Between Storage and Screen          | 243 |
| Move Formatted Data                            | 246 |
| Partition Load and Exit Instructions           | 248 |
| Load a Partition                               | 249 |
| Exit a Partition                               | 250 |
| The Load Parameters                            | 250 |
| Partial Overlay                                | 252 |
| Error Recovery                                 | 252 |
| Table Instructions                             | 253 |
| Table Read Operations                          | 254 |
| Table Write Operations                         | 255 |
| Table Search Operations                        | 258 |
| Global Tables                                  | 261 |
| Miscellaneous Instructions                     | 262 |
| Compare Logical Character Strings              | 262 |
| Generate a Self-Check Digit                    | 263 |

|  |            |
|--|------------|
| Modification for Indirect Instruction Execution        | 263        |
| Duplicate a Byte in Storage                            | 265        |
| Search Resource Allocation Table                       | 265        |
| Set Bits with Mask                                     | 266        |
| Set Indicators   | 266        |
| Exclusive-OR Write, Skip on AND Mask                   | 267        |
| System Lock and Unlock                                 | 268        |
| Translation  | 268        |
| <b>CHAPTER 5. HOW TO ASSEMBLE YOUR PROGRAM</b>         | <b>271</b> |
| The IBM 5280 Assembler                                 | 271        |
| Loading the Assembler into a Partition                 | 272        |
| The Assembler Prompts                                  | 272        |
| The Assembler Listing                                  | 279        |
| A Printed Assembly Listing                             | 279        |
| The Cross-Reference Listing                            | 280        |
| Error Messages   | 281        |
| <b>CHAPTER 6. CONTROL AREAS</b>                        | <b>283</b> |
| System Indicators within a Partition                   | 283        |
| System Registers within a Partition                    | 284        |
| Program Check Errors                                   | 285        |
| Keystroke Counters                                     | 285        |
| Data Entry Keystroke Counter                           | 285        |
| Verify Correction Keystroke Counter                    | 286        |
| Common Function Routines                               | 286        |
| Common Function Error Routines                         | 288        |
| Keyboard/Display External Status                       | 302        |
| Restricted External Status Indicator                   | 304        |
| External Status Subroutines                            | 304        |
| External Status Conditions                             | 305        |
| <b>CHAPTER 7. THE ACL TO ASSEMBLER LANGUAGE</b>        |            |
| <b>CONVERSION PROGRAM</b>                              | <b>317</b> |
| Operation  | 317        |
| Notes About the Converted Program                      | 320        |
| The Format of the Converted Program                    | 320        |
| Labels and Sequence Numbers                            | 320        |
| The Format of the Display Screen                       | 321        |
| Buffers  | 323        |
| The .FORMAT Control Statement                          | 325        |
| Indexed Branch Instructions                            | 325        |
| The OPEN Instructions                                  | 325        |
| The ENTR Instructions                                  | 325        |
| The EXEC Instruction                                   | 326        |
| Keyed Data Sets  | 326        |
| The ACL Deleted Record Subroutine                      | 326        |
| Physical Buffers                                       | 326        |
| Print Instructions                                     | 327        |
| Leading Blanks   | 327        |
| Function Keys  | 327        |
| The Minus Key  | 327        |
| ACL Toggle Switches                                    | 329        |
| Conversion Chart                                       | 329        |
| Control Statements                                     | 330        |
| Instructions   | 331        |
| Indicator Conversion                                   | 334        |
| <b>APPENDIX A. MNEMONIC TO OPERATION CODE</b>          |            |
| <b>CONVERSION CHART AND INSTRUCTIONS FORMAT</b>        | <b>337</b> |
| <b>APPENDIX B. SCS CONTROL CHARACTERS</b>              | <b>343</b> |
| Function Types Available for Use with the Format (Fmt) |            |
| Printer Control Character                              | 345        |
| Valid Values for the SHF and SVF Set Types             | 347        |

|  |            |
|--|------------|
| <b>APPENDIX C. SELF-CHECK COMPUTATIONS</b>       | <b>351</b> |
| <b>APPENDIX D. KEYBOARD CODES AND EBCDIC</b>     |            |
| <b>CHARTS</b>                                    | <b>369</b> |
| EBCDIC Charts for Printable Characters           | 369        |
| Keyboard Functions: EBCDIC Codes and Bit Numbers | 371        |
| IBM 5280 Mode Keyboard Functions                 | 371        |
| IBM 3270 Mode Keyboard Functions                 | 375        |
| <b>APPENDIX E. ERROR MESSAGES</b>                | <b>377</b> |
| Assembler Errors and Messages                    | 377        |
| Conversion Program Errors                        | 379        |
| <b>APPENDIX F. SAMPLE PROGRAM</b>                | <b>381</b> |
| <b>GLOSSARY</b>                                  | <b>387</b> |
| <b>INDEX</b>                                     | <b>395</b> |



The IBM 5280 is a diskette-based data entry system with partitioned main storage. It consists of keyboard/display data stations with optional diskette drives, a communications line, and printers. The IBM 5280 operates with multiple tasks, each running in a main storage partition. It can be used in data entry, remote batch, remote inquiry, or preprocessing environments. Input source records can be edited, verified, and placed into main storage registers, tables, or other data areas. In main storage, the records can be manipulated with arithmetic and logical operations. The records can then be reformatted and written to a data set. (A data set is a group of records stored on a diskette.) The data sets on the diskettes can then be used as input to a data processing system.

The data stations and I/O (input/output) devices are described in the General Information manual. You should be familiar with these units before you begin programming in the IBM 5280 assembler language. You must also be familiar with the organization of main storage, which is described in this chapter. Preceding the overview of main storage, this chapter gives overviews of the assembler language and of program execution. These overviews briefly describe the format of the source statements, the generation of the object code, how the IBM 5280 executes the object code instructions, and the major functions the object code can perform.

### OVERVIEW OF THE ASSEMBLER LANGUAGE

The IBM 5280 assembler language consists of control statements and instructions. The control statements define the main storage control and data areas. The instructions specify the operations and operands. No job control language is necessary for the IBM 5280.

#### The Control Statements

In your source program, a control statement is always preceded by a period (.). Control statement parameters are written with uppercase letters. The control statements are described in Chapter 3, where they are organized by function:

- Initialize control areas and I/O control blocks (IOBs)
- Declare and label data areas
- Organize tables
- Set up screen formats
- Set up edit formats
- Control the assembly listing

## The Instructions

In your source program, the instructions specify the operations and the operands. Operations are specified by arithmetic symbols or by uppercase mnemonics. Operands are specified as immediate data or as the contents of a data area. Data areas are referred to by a label or by a base displacement address. The instructions are described in Chapter 4, where they are organized by the types of operation they perform. The operations include:

- Arithmetic/logical
- Branch and subroutine
- Communications input and output
- Input and output to diskette or printer
- Input and output to keyboard/display
- Data movement
- Partition load and exit
- Table read, write, and search
- Miscellaneous

## The Source Program Format

Source statements are written with a length of 72 positions per line. Parameters are separated with spaces. You may space freely between parameters, but spaces are not allowed between a parameter and a parameter value. A control statement may be written on one or more lines. An instruction, however, must be complete within the first 72 positions of a line. Comments may be written on a control statement or instruction line, or an entire line may be written as a comment line.

Certain control statements must be written in a prescribed order. This order is explained in Figure 3-1, *Control Statement Summary* in Chapter 3.

The control statements and instructions of a source program must be written to a diskette data set before the source program can be processed by the assembler program. Enter each line of the source program as an 80-position record. The assembler program ignores the data in positions 73-80.

## The Assembler Program

The IBM 5280 assembler program reads the source program from the diskette and uses it to generate the object code. It detects syntax errors in the source control statements and instructions. It converts each label and base displacement address to an address relative to the beginning of the partition. It converts each series of screen format control statements to a string of object code, which is referred to as a screen format control string. From each source instruction, it generates a 4-byte object code instruction; the first byte always contains the operation code that determines the operation, and the other 3 bytes contain the operands. An operand may be immediate data, a format number, a table index, or the address of data to be operated upon. When the assembler program has converted the source program to object code, it then writes the object code to a diskette data set. It also generates an assembly listing that can include:

- Source code and object code
- Syntax error messages
- Storage allocation messages
- Alphabetic cross-reference of symbols used in the source program

The assembly listing can be written to a printer or to a diskette data set. Chapter 5 describes how to load and execute the assembler program.

## Loading the Object Code

The object code data set that is written by the assembler program must be loaded into a main storage partition for execution. The object code for a program can be loaded into any partition that is of sufficient size. An operator may load the object code by responding to a load prompt. Or a program being executed in a partition can have instructions to load another object program into another partition or into the same partition. See *Partition Load and Exit Instructions* in Chapter 4 for more information about loading the object code.

## OVERVIEW OF PROGRAM EXECUTION

When the object data set is loaded into main storage partition, control information and address pointers are stored in a partition control area. This control information is used by the IBM 5280 and the I/O devices during program execution. The control information is followed (1) by the data areas specified in the source program control statements and (2) by the 4-byte object code instructions.

The IBM 5280 executes the object code instructions sequentially until a specified time limit is expired or until an I/O instruction is encountered. When the time limit expires, the IBM 5280 suspends processing in that partition. The IBM 5280 then enters the next partition that has been loaded with an object data set and begins executing instructions in that partition.

If an I/O instruction is encountered, the IBM 5280 determines which I/O device is to process the operation. It places control information into the partition control area and issues the I/O instruction to the device. The I/O device processes the I/O operation, using the control information in the partition control area and in the IOB that describes that I/O operation.

### **Overlapped I/O**

Certain instructions may specify overlapped I/O. (The instruction descriptions in Chapter 4 indicate when overlapped I/O may be specified.) When the IBM 5280 encounters an I/O instruction that requests overlapped I/O, it issues the instruction to the appropriate I/O device. The IBM 5280 then either: (1) remains in the current partition and executes the instruction following the I/O instruction, or (2) if the time limit has expired for the current partition, exits the current partition and executes instructions in the next partition that contains an object data set. The I/O device processes the I/O operation concurrently with the sequential instruction execution.

If overlapped I/O is not specified, the IBM 5280 issues the I/O instruction to the I/O device and exits the partition. The instruction following the I/O instruction is not executed until the I/O instruction is completed by the device.

### **External Status**

While an I/O device is processing an I/O operation, it may encounter an external status condition that requires operator intervention or processing by the IBM 5280 controller. A four-digit condition code is placed into the IOB; it may also be displayed on the status line. These condition codes are described in Chapter 2 under *External Status and Error Conditions*.

### **Data Input**

For input via the keyboard/display, the screen format (which you specify with control statements) determines the prompts that are displayed on the screen and the display attributes for the screen, such as blink or underscore. The screen format can specify which characters are valid for each individual field of the input record. Valid fields of the input record are stored in an I/O buffer.

For input from a diskette data set, a program instruction can direct the IBM 5280 to read a data set record. The records in a data set can be accessed sequentially, directly by relative record number, or directly by key. The input record is stored in the I/O buffer.

### **Data Manipulation**

Your instructions direct the IBM 5280 to move the record from the I/O buffer. You can move a complete record or individual fields of a record to registers for arithmetic/logical operations. You can place the data into a table and can search the table entries for logical comparisons. You can keep running totals or perform self-check validation. You can test the contents of a register or a storage byte. You can perform simple or complex data movement and data comparison operations.

## Data Output

Your program instructions and formats also control record output. Records can be moved from main storage data areas to an I/O buffer. An edit format can reformat the record and insert punctuation. The records can then be written to a display, a diskette data set, a printer, or the communications line.

## OVERVIEW OF MAIN STORAGE

Main storage is organized into areas for system control, tables, common functions, partitions, and a system work buffer, as illustrated in Figure 1-1.

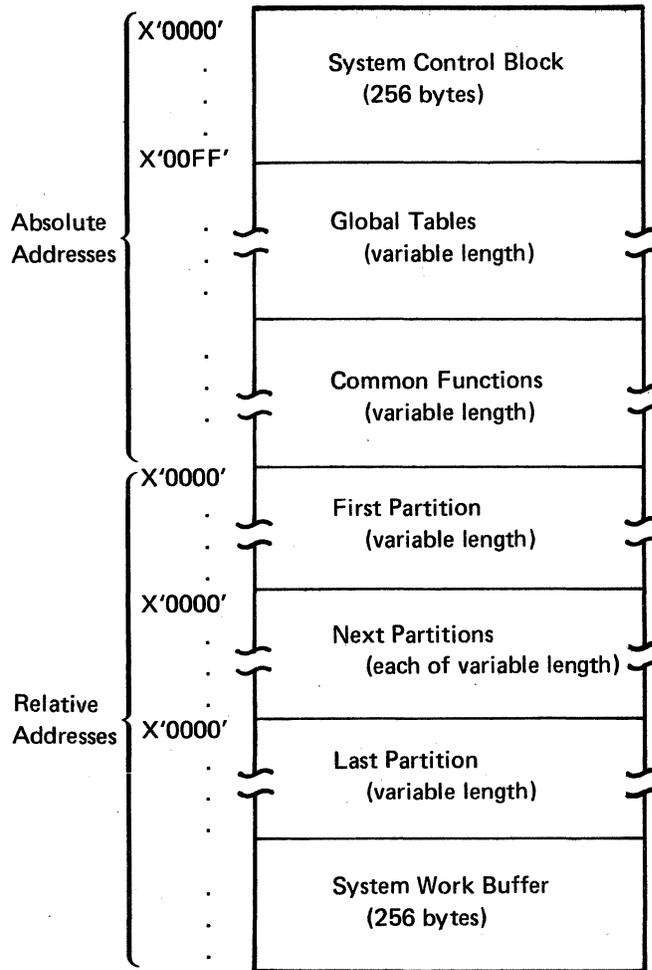


Figure 1-1. The Organization of Main Storage

The system control block is located in the first 256 bytes of main storage.

The fields of the system control block have fixed locations. However, all partitions, and all storage locations within a partition, are accessed by pointers. The pointers, which are set up and maintained by the IBM 5280, are located in the fixed locations of the system control block. These pointers make it possible for each of your assembler source programs to address locations as they relate to the partition, rather than as they relate to main storage as a whole. These relative addresses remain valid for any partition into which your program is loaded.

## Logical Device Identifiers

Logical device identifiers are two-character IDs that allow you to symbolically address a resource independently of machine or partition configuration. The logical device IDs are stored in a resource allocation table, which is created and loaded into the global tables area by the system configuration portion of the SCP (system Control Program). The resource allocation table specifies the logical devices that can be accessed by each partition. Each resource allocation table entry contains both the logical device ID and the physical address of that device. Whenever a program instruction requires a device address, you can specify the two-character ID. The IBM 5280 searches the resource allocation table for the physical address of the device with the matching ID. The IBM 5280 uses the device at that physical address to access the data set that is available to that device.

The logical device IDs are used only in program instructions. Do not enter a logical device ID via the keyboard in response to a prompt that requests a physical address.

## Common Functions

The common functions area contains IBM-supplied global subroutines. They can be accessed by a subroutine call from any partition. The labels and functions of these subroutines are listed in Chapter 6 under *Common Function Routines*.

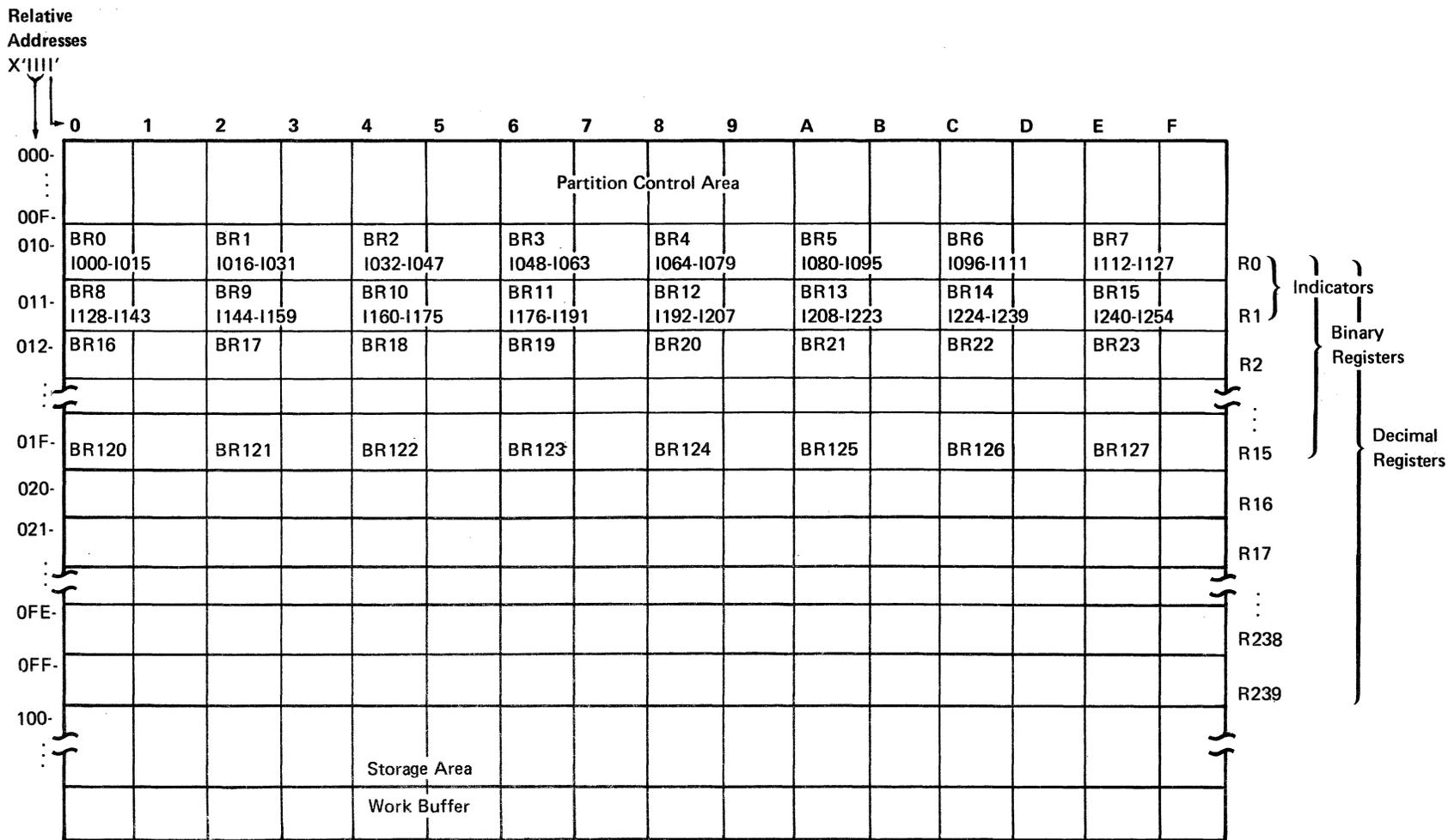
## PARTITIONS

There may be up to eight partitions numbered sequentially from zero. There must be at least one partition for each keyboard. A partition is of variable length, but it cannot cross a 64 K byte boundary. The number, size, and location of the partitions is defined at system configuration time. The first 256 bytes of each partition contains control information in fixed locations from the beginning of the partition. The next 3840 bytes may be used as needed for indicators, decimal registers, binary registers, or storage areas. This area is followed by a variable-length storage area. The last 256 bytes of each partition is used for a work area. Each byte of a partition is addressable relative to the first byte of the partition. Figure 1-2 shows the different areas of a main storage partition.

## Modes of Operation

Each partition operates in either IBM 5280 mode or IBM 3270 mode (emulation). See the *IBM 5280 Distributed Data System, IBM 5280-3270 Emulation Reference Manual, SC34-0384* for specific information concerning this mode of operation.

Figure 1-2. The Organization of a Main Storage Partition. The first three digits of the hexadecimal relative addresses are at the left, and the fourth digit is at the top.



## Partition Control Area

The partition control area contains control information that describes the program that is loaded into the partition and defines the I/O devices used in the program. The IBM 5280 loads this information into fixed locations within the control area, using information from the common area and from the source program control statements. During program execution, the IBM 5280 uses this control information each time it enters the partition to determine the partition status, the I/O status of the program, and the address of the next executable instruction.

## Indicators and Registers

Immediately following the partition control area is an area that may be used for indicators, binary registers, and decimal registers. These bytes may be used in any desired combination of indicators and registers as described in the following paragraphs: if some of these bytes are not used for their binary register/decimal register capabilities, the unused bytes may be used as storage. Figure 1-2 shows the bytes that may be used for indicators and registers.

### *Indicators*

The first 32 bytes of this area contain 255 one-bit indicators. In your source program, the indicators can be represented by the letter I and the indicator number. They are numbered sequentially from 10 to 1254. The first 100 indicators (10-199) may be labeled, set, tested, and reset as your source program dictates. These indicators are referred to as program indicators. The remaining indicators (1100-1254) are set and maintained by the IBM 5280, and are referred to as system indicators. System indicators have specific meanings assigned to them, as described in Chapter 6 under *System Indicators Within a Partition*.

You can label program indicators with a .DCLIND control statement. When the assembler processes the .DCLIND control statement, it assigns each specified label to an available program indicator.

You can label system indicators with an .EQUATE control statement. The .EQUATE control statement allows you to specify the number of the indicator you want assigned to each label. You could use the .EQUATE statement to label program indicators also; however, you usually don't care which program indicator is assigned to each label.

Two instructions are available to test indicators. The IFI instruction can test the indicator and perform a conditional branch. The IFIR instruction tests the indicator and performs a conditional branch, but it also resets the indicator to 0. Your program can use these instructions to test program or system indicators.

You can use the instruction SON to set an indicator (1), or the instruction SOFF to reset an indicator (0). See *Set Indicators* under *Miscellaneous Instructions* in Chapter 4 for a description of these instructions.

As Figure 1-2 illustrates, the bytes that are used for the indicators are also used for the first 16 binary registers or for the first two decimal registers. The last bit of the sixteenth binary register, or the second decimal register, is not used as an indicator.

## Binary Registers

The first 256 bytes of this area may be used for up to 128 two-byte binary registers. Binary registers can be represented by the letters 'BR' or 'B' followed by the register number. The registers are numbered sequentially from BR0 to BR127. BR0-BR15 are used as indicators (as described in the preceding paragraphs), and BR16-BR31 are used as system registers. The system registers are used and maintained by the IBM 5280 during program execution and hold information as described in Chapter 6 under *System Registers Within a Partition*. You should not assign these registers to any other purpose. The system registers should always be reserved (see the RGLT parameter of the .START control statement). In your source program you can access the reserved registers by register number, or you can use the .EQUATE control statement to assign them labels.

The binary registers that are not reserved by the RGLT parameter of the .START control statement can be labeled and initialized by declare control statements in your source program. Use the .DC control statement to label and initialize one binary register, or the .DCLBR control statement to label several uninitialized binary registers.

Although binary registers are 2 bytes in length, you can access either 1 or 4 bytes by defining the byte length, in parentheses, following the register number or label. If you specify a length of 1 byte (BR40(1)), only the *rightmost* byte of BR40 is accessed. If you specify a length of 4 bytes (BR40(4)), the 2 bytes of BR40 and the 2 bytes of BR41 are accessed. A binary register specification with a length of 4 bytes is referred to as a binary double register.

Binary registers are often used to hold addresses. The instructions to load a binary register are described in *Load Binary Register* under *Data Movement Instructions* in Chapter 4. In your source program, you can load a 2-byte binary register with:

- An unsigned decimal integer (0-65535)
- Two EBCDIC characters

Figure 1-3 shows the hex representation of binary data in two binary registers.

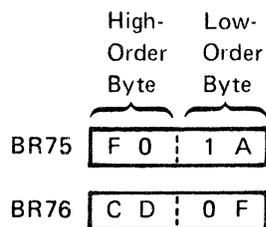


Figure 1-3. Binary Registers

The following examples illustrate the different ways you can refer to BR75 if you assign it the label BREG1.

|                                      |   |   |
|--------------------------------------|---|---|
| BR75<br>BR75(2)<br>BREG1<br>BREG1(2) | } | specifies the full 2-byte binary register, which contains hexadecimal F01A. |
| BREG1(1)<br>BR75(1)                  | } | specifies the low-order byte of BR75, which contains hexadecimal 1A.        |
| BR75(4)<br>BREG1(4)                  | } | specifies the 4 bytes of BR75 and BR76, which contain hexadecimal F01ACD0F. |

### *Decimal Registers*

The 3840 bytes of this area may be used for up to 240 sixteen-byte decimal registers. Decimal registers can be represented by the letter R and the register number. The registers are numbered sequentially from R0 to R239. The bytes within R0 and R1 are used for indicators; the bytes within R2 and R3 are used for system registers. You should not assign R0-R3 for any other purpose. In your source program, the decimal registers reserved by the RGLT parameter of the .START control statement can be accessed by register number. Or you can use the .EQUATE control statement to assign them labels.

Decimal registers not reserved by the RGLT parameter of the .START control statement can be labeled and initialized by the declare control statements in your source program. Use the .DC control statement to label and initialize one decimal register, or the .DCLDR control statement to label several uninitialized decimal registers.

Although a decimal register is 16 bytes in length, a double decimal register of 32 bytes may be specified by defining the byte length in parentheses, following the register number or label. Decimal registers and double decimal registers are valid in decimal arithmetic and shift operations, branch operations, and table operations. All data in decimal registers is stored in EBCDIC notation. The instructions to load a decimal register are in *Load Decimal Register* under *Data Movement Instructions* in Chapter 4. In your source program, you can load a 16-byte decimal register with:

- A positive or negative decimal number ( $\pm 0$  to  $10^{16}-1$ )
- Up to 16 EBCDIC characters

The following examples illustrate the different ways that you can refer to R120 if you assign it the label REGX.

|                      |   |  |
|----------------------|---|--|
| R120<br>REGX         | } | specifies the 16 bytes of R120.          |
| R120(32)<br>REGX(32) | } | specifies the 32 bytes of R120 and R121. |

The contents of a decimal register may be positive or negative; the sign is determined by the zone half of the byte in the units position (byte 15) of the decimal register. If the register contains a positive number, hex F is in the zone half; if it contains a negative number, hex D is in the zone half. Figure 1-4 illustrates the sign control position in a decimal register.

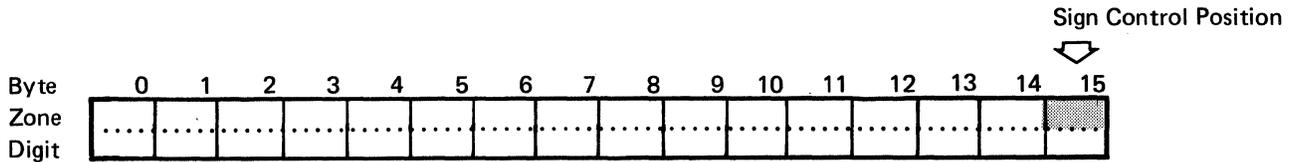


Figure 1-4. The Sign Control Position in a Decimal Register

The zone halves of the bytes are used for sign control; however, no checking is done by the IBM 5280 to determine whether the register contents are numeric or alphabetic.

### Storage

Following the registers is a variable-length area of storage. The size of this area is the size of the partition, less the 256 bytes of the partition control area and the bytes used for indicators and registers. The instruction object code is stored in this area, with the buffers, tables, formats, messages, device IOBs, control tables, data, and data structures necessary for the program.

### Addressing Methods

In your source program, each byte of storage within a partition can be addressed directly, using an assigned label, or indirectly, using a displacement and a base address.

*Direct labeled addressing* of a storage location is accomplished by using a declare control statement to assign a label to a storage area of any length. To access this labeled area in a source program instruction, the following format is used.

label [(length)]

where:

*label* is the assigned label from the declare control statement. The label addresses the leftmost byte of the storage area.

*length* is the length, in bytes, of the storage area. If length is not specified in the instruction, the length defaults to the length assigned to that area by the declare statement.

*Indirect base displacement addressing* of a storage location is accomplished by specifying in the instruction (1) the location of the base address and (2) the displacement from that base address at which the storage area is located. The length may be specified for many, but not all, instructions. To access a storage location by indirect addressing, one of the following formats is used.

[displacement] ([length], BRn)  
[displacement] (BRn)

where:

*displacement* is the number of bytes (0-255) from the base address at which the storage area is located. If the displacement is not specified, it defaults to 0.

*length* is the length, in bytes, of the storage area. The instruction descriptions indicate whether or not length is allowed in the address. If a length specification is allowed, it must be followed by a comma. If length is omitted from an instruction that allows a length specification, the comma must be retained. If the instruction does not allow a length specification, the comma must *not* be included in the address.

*BRn* is a binary register that contains the base address. The base address is relative to the start of the partition.

When a source program instruction that has an indirect storage address is assembled, the displacement is added to the base address in the binary register. The result is the relative address of the leftmost byte of the data area. This address is placed in the object code.

*Examples:*

Direct: BIN1 = STOR1(2)  
The contents of the byte at STOR1 and the next byte (length is 2) are loaded into the binary register labeled BIN1.

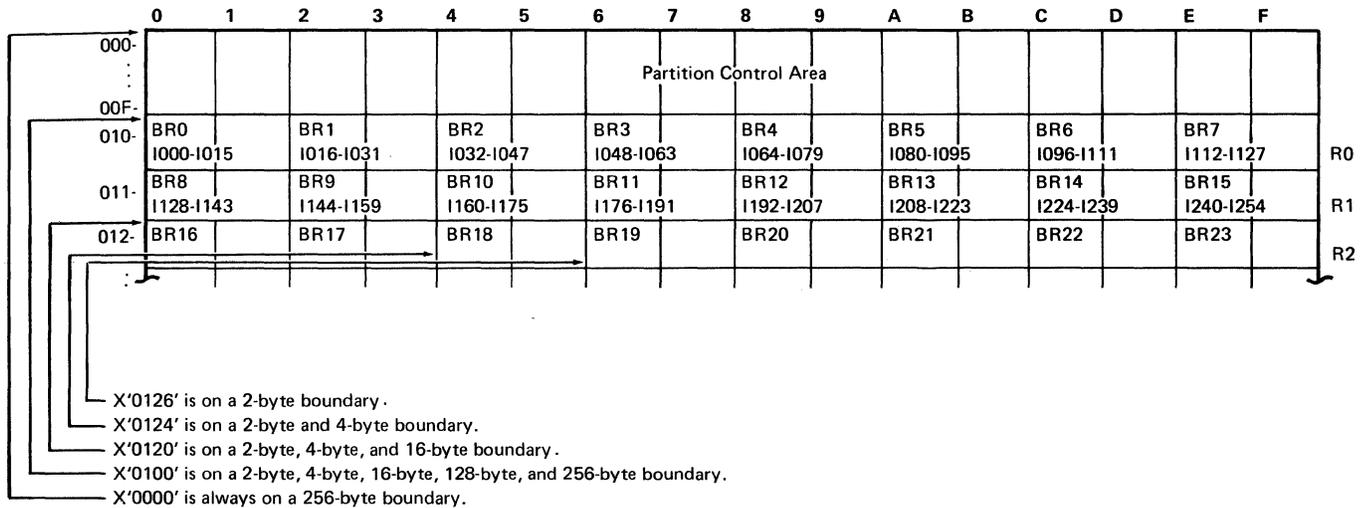
Indirect: BIN2 = 1(2, BREG)  
The displacement of 1 is added to the address stored in the binary register labeled BREG. The contents of the byte at the resulting address and the contents of the next byte (length is 2) are loaded into the binary register labeled BIN2.

### Partition Work Area

Following the variable-length storage area is a 256-byte work area. This area is set up by the assembler, and it is used by the IBM 5280 during program load and program execution. Your assembler program should not access or change the bytes of this work area.

## MAIN STORAGE BOUNDARY ALIGNMENT

Main storage is divided by several types of boundaries. Each type of boundary encloses an area of a specific number of bytes. Many data areas must begin on a certain type of boundary. Figure 1-5 represents a main storage partition and points out the different types of boundaries. The system configuration portion of the SCP begins each partition on a 256-byte boundary and measures the length of each partition in multiples of 256 bytes.



The address of a 2-byte boundary ends in 0, 2, 4, 6, 8, A, or C.

The address of a 4-byte boundary ends in 0, 4, 8, or C.

The address of a 16-byte boundary ends in 0.

The address of a 128-byte boundary ends in 00 or 80.

The address of a 256-byte boundary ends in 00.

**Figure 1-5. Main Storage Boundaries**

When you declare a register in your source program, the assembler places it on the next sequential boundary appropriate for the type; it places a binary register on a 2-byte boundary and a decimal register on a 16-byte boundary. It places all other data types on 1-byte boundaries unless you specify a boundary. When you are building a storage structure, you may want to specify a boundary. When the IBM 5280 assembler processes your source control statements and sets up these data areas, it skips over any storage bytes between the current location and the next appropriate boundary in order to observe the boundary restrictions. These bytes cannot be used by your program. Your assembly listing indicates how many storage bytes are lost due to boundary alignment. See the examples following the description of the .DC control statement in Chapter 3 for an illustration of boundary alignment.

## BLANKS, CONSTANTS, AND CODING SYMBOLS

In your source program, you may specify optional blanks before or after an equal sign, arithmetic operator, or parenthesis. Blanks may follow a comma but must not precede a comma. Blanks are not allowed within a field; however, one or more blanks must separate fields if no other delimiter is used.

A constant may be specified as a decimal value, a hexadecimal value, a binary value, or a character. A constant may also be equated to a label; the label can be specified in an instruction that accepts a constant. Decimal digits are simply written as digits. Binary, hexadecimal, and character data are prefixed by a capital letter (B, X, and C respectively) and enclosed in single quotes. For character data the capital C is optional. Do not leave a blank between the capital letter and the first quote.

|      |   |
|------|---|
| n    | Decimal digits                                    |
| X'I' | Hexadecimal digits; I = 0-F                       |
| B'I' | Binary digits; I = 1 or 0                         |
| C'I' | EBCDIC characters; I = any valid EBCDIC character |

To specify the single quote character, use two quotes (C'IT''S').

## Symbols Used in This Manual

The symbols used in this manual are of two types, syntax symbols and statement symbols. The syntax symbols are used to illustrate syntax and are not to be used in writing your source programs. The statement symbols are a part of the language and must be coded as shown.

### *Syntax Symbols*

Syntax symbols are not to be coded in the source program.

| <b>Symbol</b> | <b>Definition</b>   |
|---------------|---|
| [ ]           | <i>Brackets</i> enclose optional item(s) to be used or not, at your discretion.   |
| { }           | <i>Braces</i> enclose two or more items from which you must select one.   |
| ...           | <i>Three dots</i> indicate that the preceding can be repeated.  |
| b<br>label    | <i>Lowercase letters</i> represent information you must supply. (You must substitute your own values for the lowercase terms.)  |
| n             | Represents an unsigned decimal number.  |
| ±n            | Represents a signed decimal number.   |
| 0–9           | Represents a range of numbers from which one number can be selected. (The dash is not coded.)   |
| In            | Represents an indicator, which can be referred to by label or number.   |
| BRn           | Represents a binary register, which can be referred to by label or number.  |
| Rn            | Represents a decimal register, which can be referred to by label or number.   |
| BRa<br>Rb     | When more than one register may be used in a statement, the letters a, b, and c may replace the n to more clearly demonstrate the positions in the statement that the different registers may occupy. |
| constant      | Represents any form of constant as described in this chapter.   |

## Statement Symbols

Statement symbols must be used in an assembler source program as shown in the syntax illustrations:

| Symbol     | Definition  |
|------------|---|
| :          | <i>Colon</i> is used after symbolic labels.   |
| ;          | <i>Semicolon</i> delimits statements.   |
| .          | <i>Point</i> , or period, begins control statements.  |
| =          | <i>Equal sign</i> causes the value of the data area on the left of the equal sign to be changed according to the value of the data area on the right of the equal sign.   |
| ( )        | <i>Parentheses</i> enclose certain parameter values.  |
| ' '        | <i>Single quotes</i> enclose literals and are used to specify numeric data other than decimal. The use of single quotes is interchangeable with the use of capital C and single quotes. For example, C'abc' and 'abc' produce the same results. |
| ,          | <i>Comma</i> separates parameter values.  |
| S<br>LABEL | <i>Uppercase letters</i> are assembler language and must be coded as shown.   |

This chapter discusses various data areas that are set up according to your control statements and are used by the IBM 5280 during program execution. The discussions often refer to the control statements that generate the areas and the instructions that use the areas during program execution. Each control statement is described in Chapter 3; each instruction is described in Chapter 4.

**TABLES**

Tables play an important part in IBM 5280 assembler programming. Two kinds of tables may be set up and used by your program: data tables, which are set up by .TABLE control statements, and label tables, which are set up by .LABTAB control statements. Also, the assembler builds system tables, which are used by the IBM 5280 during program execution. These system tables allow you to refer to a data area with a label; the IBM 5280 converts the label to an index that points into a system table and locates the address and parameters of the area.

**System Tables**

When the assembler processes control statements that set up as tables, formats, or prompts, it places the address of each table, format, or prompt in a table. This table of addresses is a system table, and is used by the IBM 5280 during program execution. System tables are stored in the partition storage area. You can specify the address of the system tables by using the .SYSTAB control statement in your source program. Otherwise, when the assembler encounters the .END control statement, it stores the system tables at the addresses immediately following the last address that contains program object code. The address of each system table is stored in the partition control area. The control statements that generate a system table argument are listed below, with the system table into which the argument is entered.

| <b>System Table</b>         | <b>Control Statements</b>               |
|-----------------------------|---|
| Table control               | .TABLE                                  |
| Edit format control         | .FMT series (each series = 1 argument)  |
| Screen format control       | .SFMT series (each series = 1 argument) |
| Prompt control              | .DC TYPE=PRMT                           |
| Duplicate and store control | .DC TYPE=MDUP                           |

When a source instruction refers to a prompt, table, duplicate field, store field, or format, the instruction specifies only the label. The IBM 5280 uses this label to find the system table entry; the system table entry provides the address and other control parameters. The system table entries are stored sequentially, in the order they occur in the source program. Except for the prompt system table, the first entry in a system table is at index 0; for the prompt system table, the first entry is at index 1. The assembler places the table index into the object code instruction. This method requires only 8 to 10 bits of the 4-byte object code to provide the address and parameters of the requested data area. The .SYSTAB control statement description in Chapter 3 describes how to specify the labels and addresses of the system tables.

### Data Tables

Contiguous fields of related data can be referred to as a data table. In your source program, you can allocate and initialize the fields of a data table by using .DC control statements. After you have allocated the fields, you must use the .TABLE control statement to structure the fields into a table. The first argument in a data table is at index 1. You may have up to 128 tables within a partition. You must include one .TABLE control statement for each table in your source program.

You can use instructions in your source program to request that the IBM 5280 search, read, or write the entries in a data table. See *Table Instructions* in Chapter 4 for a description of these instructions.

Data tables can be fixed length or variable length, according to your .TABLE control statement. See the .TABLE control statement definition in Chapter 3 for an example of .DC and .TABLE control statements that build a variable length table.

### Label Tables

Label tables are tables that contain addresses; they are used by your program to make indexed branches and indexed subroutine calls. In your source program you use a .LABTAB control statement to set up a label table.

The parameters of the .LABTAB control statement specify the labels of the subroutines or instructions you wish to enter into the label table. The address of the first label specified in the .LABTAB statement is entered at index 0 in the label table, the address of the second label is entered at index 1, and so on. When you code a GOTAB or CALLTB instruction, you specify (1) the label of the label table and (2) the label table index of the subroutine or instruction you wish to execute.

The IBM 5280 makes similar indexed branches through the label table you use for your external status condition subroutines, if you code a separate subroutine to handle each condition. (See *Keyboard/Display External Status* in Chapter 6.) You specify this label table in the ETAB parameter of the .KBCRT control statement.

## DATA TYPES

Each source instruction and control statement requires specific types of data to be used as operands. For some operands only one type of data is accepted. For example, the format operand of the ENTR instruction requires a screen format specification; no other type of data is accepted. For other operands more than one type of data may be specified. For example, the operand of the ZONE instruction may be specified as a decimal register or as a constant.

The following data types can be used in the instruction and control statement operands.

- Label or number of an indicator
- Label or number of a binary register
- Label or number of a decimal register
- Label of an instruction
- Label of a data storage area (from a STOR type .DC)
- Label of a prompt (from a PRMT type .DC)
- Label of a duplicate area (from a MDUP type .DC)
- Label of an edit format
- Label of a screen format
- Number of a data set
- Index of a table
- Constant

## SUBROUTINES

A program can call any subroutine that is stored within the partition. Calls to routines in the common function area are discussed under *Common Function Routines* in Chapter 6.

Two source instructions can be used to call a subroutine: the CALL and CALLTB instructions. These instructions are described in Chapter 4 under the *Subroutine Call and Return* instructions. A CALL instruction must include a label or a binary register, or both. If the CALL instruction includes a label, a normal call is made to the statement at the specified label. If the CALL instruction specifies a binary register and no label, a call is made to the address contained in the register. If the CALL instruction specifies a binary register and a label, the contents of the binary register are added to the address of the specified label, and a branch is made to the resulting address.

The CALLTB instruction is used to make an indexed branch through a label table. The label table must be set up and labeled by a .LABTAB control statement. You include this label table and a binary register when you write the CALLTB instruction. The binary register contains the index of the subroutine you wish to call. The first entry in the label table is at index 0. When the CALLTB instruction is executed, the call is made to the subroutine at the specified index into the label table. If you use a separate subroutine for each external status condition, the IBM 5280 uses this method to call the appropriate external status subroutine. The IBM 5280 uses BR23 to hold the index into the external status subroutine label table.

### The Partition Subroutine Stack

Whenever a subroutine call instruction is executed, the address of the next sequential instruction is assumed to be the return address and is stored into the partition stack. When an external status condition is detected during an I/O operation, and the IBM 5280 branches to the external status routine, it checks byte 13 bit 5 of the DATASET IOB. If the bit is set, the address of the I/O operation will be used as the subroutine return address and is stored into the partition stack. Otherwise the address of the next sequential instruction following the I/O operation will be used as the return address. The partition stack is a system table with 2-byte entries, located in partition storage. You may use the .SYSTAB control statement in your source program to specify the address and size of the partition stack. Otherwise, when the assembler encounters the .END statement it locates the beginning of the partition stack in the address following the last address that contains program object code or system tables. In either case, it stores the address of the beginning of the partition stack in BR18, which is referred to as the stack pointer. When the first subroutine call is executed, the 2-byte return address is placed in storage at the address indicated by BR18. Then the address in BR18 is incremented by two, so that it points to the next available stack entry. If another call is executed before a return is made to the first call, the return address for the second call is placed in the address indicated by BR18, and BR18 is incremented by two. In this way, you can have nested subroutine calls. You must remember, however, that each nested call adds another 2-byte entry to the partition stack. If the partition stack extends beyond the end of the partition, a program check error results.

### Subroutine Returns

External status subroutine returns depend upon the particular external status condition and are described under *External Status and Error Conditions* in this chapter.

Other subroutines end with a RETURN instruction. When this instruction is executed, the content of BR18 is decremented by two so it points to the last address entered into the partition subroutine stack. If the RETURN instruction includes a binary register, an indexed return is made. The content of the binary register is added to the address pointed to by BR18, and control returns to the resulting address.

Figure 2-1 illustrates how the partition stack and stack pointer are used during subroutine calls and during returns.

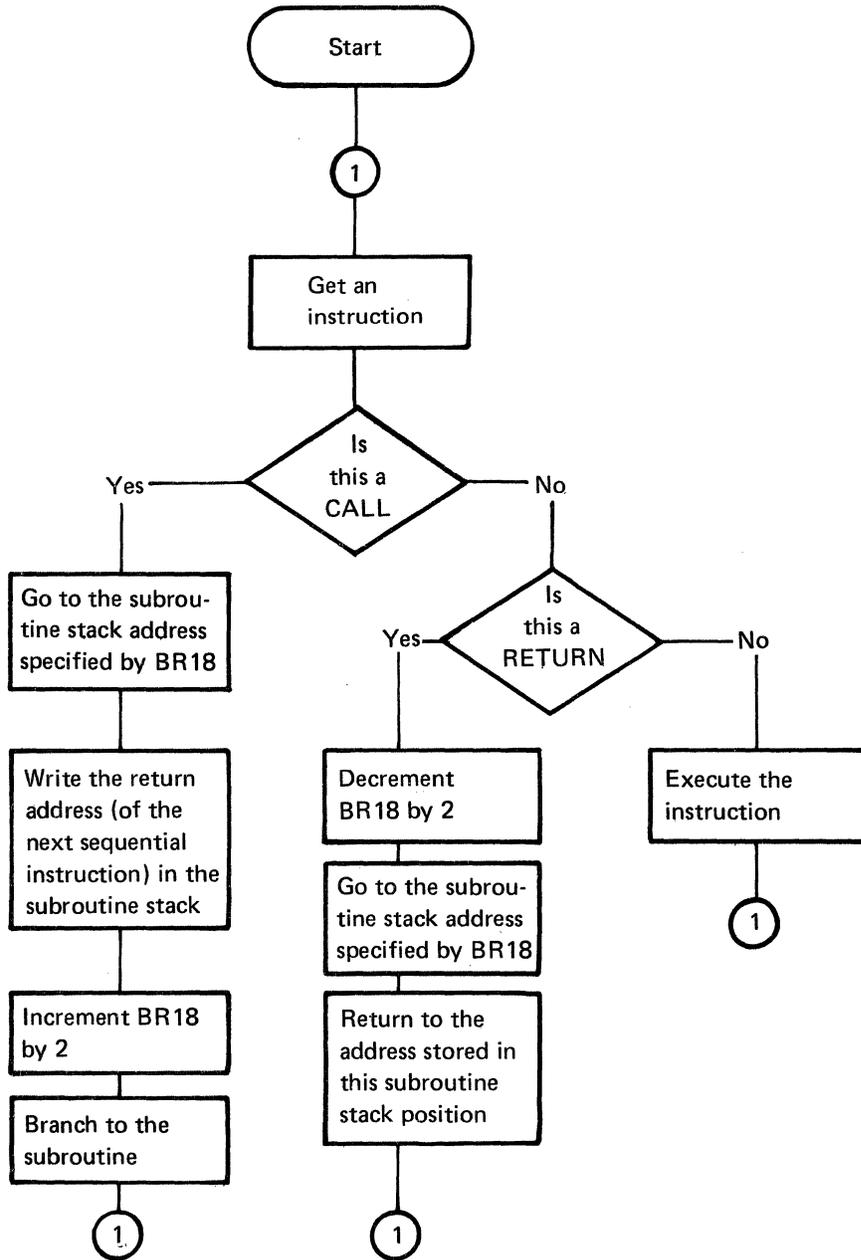


Figure 2-1. Overview of Subroutine Calls and Returns

## THE STATUS LINE

The top line of the data station screen normally displays the status line. The IBM 5280 maintains status line fields, which communicate status information to the operator. Figure 2-2 illustrates the status line fields.

|   |         | Position |   |   |   |   |   |   |   |   |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |     |    | Mode |  |           |                              |              |
|---|---------|----------|---|---|---|---|---|---|---|---|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|-----|----|------|--|-----------|------------------------------|--------------|
|   |         | 1        | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | ... | 32 |      |  |           |                              |              |
| P | C C C C |          |   |   |   |   |   |   |   |   |    | S  |    |    | R  | R  |    |    | H  | H  |    |    |    |    |    |    |    |     |    |      |  |           |                              | Normal Entry |
| P | C C C C |          |   |   |   |   |   |   |   |   |    | S  |    |    | R  | R  | >  |    | H  | H  |    |    |    |    |    |    |    |     |    |      |  |           | Normal Entry,<br>Insert Mode |              |
| P | C C C C | -        | E | E | E | E | - |   |   |   | S  |    |    | R  | R  |    |    | H  | H  |    |    |    |    |    |    |    |    |     |    |      |  |           | Keystroke<br>Error           |              |
| P | C C C C | -        | E | E | E | E | - | L | L |   |    |    |    | N  | N  | N  | N  | N  | N  | N  | N  | N  | N  | N  | N  | D  | D  | ... | D  |      |  | I/O Error |                              |              |

### Key

- P is the partition number.
- C is the current position counter.
- E is the error or condition code.
- S is the field shift.
- > is the insert mode symbol.
- R is the positions remaining in the field.
- H is the hex value of the current position.
- L is the logical device ID.
- N is the program name (first 8 characters).
- D is the data set name.

Figure 2-2. The Status Line Fields

### *The Partition Number*

The partition number is maintained only during an attach or detach operation. Upon completion of a successful attach operation, this status line field contains the partition number of the attached partition. Upon completion of a successful detach operation, this field contains the partition number of the foreground partition that is permanently associated with the keyboard.

### *The Current Position Counter*

The current position counter is maintained only during the processing of an ENTR command. This status line field contains the value of the position counter. The value is automatically updated with each keystroke. The value reflects the current position, relative to: (1) the beginning of the I/O buffer, (2) the first position on the screen, (3) the first position of the record, or (4) the first position of the field. The CNTR parameter of the .KBCRT control statement determines which value is maintained in the counter.

### *The Error Code*

The error code field of the status line contains the error code of the current error. It is maintained by the IBM 5280 to reflect all errors. If your program issues a keyboard operation to send an error code to the status line, you may place the code in positions 1-65 of the status line; however, the code is normally placed in positions 3-11.

### *The Field Shift*

The field shift position of the status line is maintained only while an ENTR command is being processed. It contains the symbol that reflects the keyboard shift for the current field or subfield.

### *The Insert Mode Symbol*

The insert mode symbol is maintained only during the processing of an ENTR command in insert mode, after the operator has pressed the Ins (Insert) key.

### *The Positions Remaining in the Field*

This status line field is maintained only while an ENTR command is being processed. It reflects the number of field positions remaining to be entered in the current input field. If the value is greater than 99, two asterisks (\*\*) are contained in the status line field.

### *The Hex Value of the Current Position*

The hex value is maintained on the status line only while an ENTR command is being processed. It is the hex value contained in the I/O buffer position that corresponds to the current position of the cursor.

### **Nondisplay of the Status Line**

Certain applications may require the use of every line on the screen. For these applications, the DISPEX instruction can remove the status line from the screen so the top line can be used to display data or prompts, or both. The IBM 5280 maintains the status line whether or not it is displayed on the screen. If an error occurs when the status line is not being displayed, the DISPST instruction can temporarily replace the current top line with the status line in order to communicate error information to the operator. Or the FUNC parameter of the .KBCRT control statement can specify that the IBM 5280 determines whether the status line is being displayed whenever an error occurs; if it is not, the IBM 5280 displays it, then returns the top line when the error is reset. The data from the top line is not lost and may be returned to the screen after appropriate error recovery has been accomplished.

The DISPEX and DISPST instructions are discussed under *Keyboard Operations* in Chapter 4. The .KBCRT control statement is discussed under *.KBCRT Control Statement* in Chapter 3.

## EXTERNAL STATUS AND ERROR CONDITIONS

When an I/O error condition or a condition that requires operator intervention occurs, the IBM 5280 generates an appropriate condition code and places it into the IOB of the data set that was being processed when the condition occurred. The condition code is made up of four digits that describe the condition:

- Device reporting the condition (first digit)
- Category of the condition (second digit)
- Condition number (third and fourth digits)

The device digits are:

| Digit | Device                           |
|-------|----------------------------------|
| 0     | IBM 5280 controller              |
| 1     | Keyboard/display                 |
| 2     | Printer                          |
| 3     | Diskette                         |
| 4     | SNA communications access method |
| 5     | BSC communications access method |
| 9     | Program                          |

The category digits are:

| Digit | Device  |
|-------|---|
| 0     | Communications completion codes                 |
| 1     | Operator intervention required                  |
| 2     | Hard error (has not been retried)               |
| 3     | Error has been unsuccessfully retried           |
| 4     | IOB error                                       |
| 5     | Soft error (has been retried)                   |
| 6     | Exception condition                             |
| 7     | Warning message, program execution may continue |
| 8     | Reserved  |
| 9     | Software termination                            |

The last two digits of the condition code are the condition number. The condition number specifies the condition and varies depending upon the device and category. All condition codes and messages will be described in the *Message Manual*.

The following information concerning the condition is placed into system binary registers within the partition when the condition occurs.

| Register | Information  |
|----------|--|
| BR19     | Used only with keyboard/display external status, this register contains the relative address of the field in the I/O buffer that holds the current record. The address is relative to the beginning of the partition and is valid only when BR21 contains a field specification.   |
| BR20     | Used only with keyboard/display external status, this register contains the absolute address of the current field in the screen refresh buffer and is valid only when BR21 contains a field specification. The screen refresh buffer is located within the keyboard/display unit and holds the data that appears on the screen.  |
| BR21     | Used only with keyboard/display external status, this register contains a control specification or a field specification. If it contains a field specification, it also contains the length minus one of the current field in the I/O buffer. See <i>Keyboard/Display External Status</i> in Chapter 6 for the format of the contents of this register.  |
| BR22     | Used with all external status <i>except</i> keyboard/display, this register contains the relative address of the last IOB to report external status.   |
| BR23     | Used with all external status, this register contains the index of the current external status condition. This index can be used by your program as the index into your external status subroutine label table. Except for keyboard/display external status, this index is the category digit from the condition code. See <i>Keyboard/Display External Status</i> in Chapter 6 for information about this index for keyboard/display external status. |

If you write subroutines to handle certain external status conditions, such as the keyboard/display external status conditions, your program may use the data in these registers. Do not change the data in the system registers.

## KEYBOARD DATA ENTRY

Keyboard entry of each input record is initiated with an ENTR command. The input record is formatted according to the screen format. The operator enters characters into the fields of the input record, and the IBM 5280 makes character and edit checks to make sure the characters are valid for the field, according to the specifications in the screen format. Data keys and many function keys sound a response click from the keyboard. The characters, as they are entered, are stored into the I/O buffer and are displayed on the screen. For enter, update, and verify modes a keystroke counter is incremented when each character is entered. (See *Keystroke Counters* in Chapter 6 for more information.) The cursor is moved to the screen position where the next character is to be entered. The operator can move the cursor forward and backward within the current record.

The status line displays data entry information such as the current keying position, the number of positions remaining to be filled in the current field, and the keyboard shift for the field.

The operator can select functions, such as duplicate or skip, by pressing the appropriate function keys. You can let the IBM 5280 process keyboard functions, or you can include your own routines to handle these functions. See the *Functions Reference Manual* for more information about the keyboard functions.

### Keystroke Buffering

Keystroke buffering gives the application program the ability to protect the operator from 1110 message codes during interrecord and return-to-program processing by postponing the processing of keys pressed during those time periods. Across record boundaries, the KB/CRT MPU can at a user option buffer keystrokes in a main storage buffer until processing on a new ENTR command begins. During return-to-program, the KB/CRT MPU can at a user option buffer keystrokes in a main storage buffer until processing on a RESUME operation begins.

For a further description of keystroke buffering, see the *IBM 5280 Functions Reference Manual*, GA21-9353.

### Modes of Entry

The IBM 5280 supports three basic modes for data entry:

- Enter mode, for initial data entry
- Update mode, for inspection and modification of previously entered data
- Verify mode, for having data checked for accuracy and making necessary corrections

In addition to these basic modes, rerun mode or display mode can be selected by your program to perform special functions. You can select one of these five modes with the MODE parameter of the .KBCRT control statement. (See *.KBCRT Control Statement* in Chapter 3.) Insert mode or field correct mode is automatically selected by the IBM 5280 when the appropriate keystroke is entered.

### Enter Mode

When the IBM 5280 executes an ENTR command in enter mode, each data character is displayed on the screen and placed into the I/O buffer as it is entered. Prompts, constant inserts, duplicate fields, skip fields, and display attributes are displayed when the cursor moves to the first position of the field or to the attribute position; these positions are specified in the screen format. Constant inserts are also placed into the I/O buffer as they are displayed. When the complete screen format has been processed, the I/O buffer holds the constant inserts and the newly entered data.

### Update Mode

When the IBM 5280 executes an ENTR command in update mode, prior instructions in your program must have placed a previously-entered record into the I/O buffer.

The IBM 5280 displays prompts, display attributes, and the contents of the I/O buffer. The display attributes and prompts are determined by the screen format. The operator can enter a new data character into any record position to replace the data character currently in the record. The new data character is displayed on the screen and placed into the I/O buffer as it is entered. When the operator has completed all necessary modifications, the I/O buffer contains the original data in all positions that were not modified and the new data in the positions that were modified.

### *Verify Mode*

When the IBM 5280 executes an ENTR command in verify mode, prior instructions in your program must have placed a previously-entered record into the I/O buffer. The IBM 5280 displays the prompts and display attributes as for enter mode, according to the screen format. It does not display the contents of the I/O buffer. As the operator enters a data character into a record field position, it is verified against the contents of the corresponding field position in the I/O buffer. If the newly entered character matches the original character, which is in the I/O buffer, it is displayed on the screen and the cursor moves to the next position. If the newly entered character does not match the original character, the cursor remains at the character position, the original character and the remainder of the field in the I/O buffer are displayed, and a verify mismatch error is reported. The operator must press the Reset key, then enter either the character displayed above the cursor or reenter the character that caused the mismatch. If the character that is displayed above the cursor is entered, the remainder of the field is removed from the screen and the cursor moves to the next position. If the character that caused the mismatch is reentered, that character is displayed above the cursor and replaces the original character in the I/O buffer. A verify-correction keystroke counter is incremented (see *Keystroke Counters* in Chapter 6) and the cursor moves to the next position. If the character entered is neither the original character nor the character that caused the mismatch, another verify mismatch error is reported. If the operator backspaces over a data position on the screen, the position is blanked and must be reentered and reverified.

### *Rerun Mode*

When the IBM 5280 executes an ENTR command in rerun mode, no data or prompts are displayed on the screen. The status line counters, keyboard shift, and hex display information is not maintained. The entire screen format is processed, except that a clear-screen function that is specified at the start or end of the format is ignored. Character and field edit checks are bypassed. Auto duplicate, auto skip, and main storage duplicate and store functions are performed if the auto-dup/skip switch is turned on or if the field has the AA (absolutely automatic) attribute specified in the screen format. Constant inserts are placed into the I/O buffer. When an RG (return to program) exit specification is encountered in the format, the appropriate external status condition occurs.

### *Rerun/Display Mode*

When the IBM 5280 processes an ENTR command in rerun/display mode, the prompts, display attributes and the contents of the I/O buffer are displayed as for update

mode. The status line information is maintained. Character and field edit checks are bypassed. Auto duplicate, auto skip, main storage duplicate and store, and RG functions are performed as for rerun mode.

Rerun mode requires less execution time than rerun/display mode, and is the mode usually selected for the rerun function. Rerun/display mode can be used when an error occurs when a record is being processed in rerun mode, and the operator must inspect the record data order to recover from the error.

### *Display Mode*

When the IBM 5280 executes an ENTR command in display mode, prior instructions in your program must have placed a previously-entered record into the I/O buffer. The IBM 5280 displays prompts, display attributes and the contents of the I/O buffer. The display attributes and prompts are determined by the screen format. The cursor is not displayed, and no data can be entered. If a buzz or clear-screen function is specified at the end of the screen format, it is ignored. When the IBM 5280 has processed the complete screen format, the external status condition for record advance (condition 6) occurs.

You can use display mode to inspect the prompts and display attributes of a screen format. Do not confuse display mode with rerun/display mode.

### *Insert Mode*

Insert mode is initiated when the operator presses the Ins key. Insert mode is valid only when an ENTR command is being processed. When the Ins key is pressed, the insert mode symbol is displayed on the status line. When the operator presses a data key, the data character is inserted into the field at the current cursor position. All field positions to the right of the cursor, and the cursor and character above the cursor, are shifted one position to the right. Insert mode is canceled when the operator presses the Reset key.

### *Field Correct Mode*

Field correct mode is selected by the IBM 5280 when it is processing an ENTR in verify mode and the operator presses the unshifted Corr key. The cursor moves to the first position of the field, and the field is filled with blanks in the I/O buffer and on the screen. The operator can then enter data into the entire field as for enter mode. The character and field edit checks are performed. When the cursor exits the field in the forward direction, the IBM 5280 returns to verify mode. The field can now be verified.

## **AUTOMATIC FUNCTIONS**

While the IBM 5280 is processing formatted data entry, certain functions may be initiated automatically as specified in your application program. These functions include auto enter, auto duplicate/skip, and alternate record advance. You can activate these automatic functions by including the FUNC parameter of the .KBCRT control statement. You can activate automatic functions by providing support in your program for the Auto Enter and Dup Skip keys. See the *Functions Reference Manual* for a detailed description of all keyboard functions.

### **Auto Enter**

If you specify auto enter in your .KBCRT control statement, the IBM 5280 automatically performs a record advance function when the operator enters the last input position of a record.

If you do not specify auto enter, the IBM 5280 sets the system in the awaiting record advance state when the operator enters the last position of a record. The operator must then press the Enter key or Rec Adv key to initiate a record advance function.

### **Auto Duplicate/Skip**

If you specify auto duplicate/skip in your .KBCRT control statement, the IBM 5280 automatically processes any field that is defined in your program as an auto duplicate or auto skip field. When the cursor moves to the first position of an auto duplicate field, the IBM 5280 duplicates data into the field from the area specified by your program. (See *Field Definitions* later in this chapter for more information about duplicate fields.) When the cursor moves to the first position of a skip field, the IBM 5280 fills the field with blanks and then moves the cursor to the first position of the next field.

If you do not specify auto duplicate/skip in the .KBCRT control statement, a duplicate field or skip field is processed as for a manual field. In order to initiate the duplicate or skip function, the field must have also been specified as absolutely automatic in the program, or the operator must press the Dup Skip key. (Software must provide support for this key.)

### **Alternate Record Advance**

If you specify alternate record advance in your .KBCRT control statement, when the operator presses the Enter key or Rec Adv (Record Advance) key the processing of the current record stops. Any specifications for fields or screen control that is defined in your program for positions between the cursor position and the end of the record are ignored.

If you do not specify alternate record advance, any specifications for fields or screen control defined for positions between the cursor position and the end of the record are processed. Input fields are processed as though a → (Field Advance) key were pressed for each field.

## **SCREEN FORMATS**

A screen format is a series of source program control statements that define each field of a record to be entered via the keyboard. The control statements also define the prompts and display attributes that appear on the screen while the record is being entered. The series of control statements must begin with a .SFMTST statement and must end with a .SFMTEND statement. You can write up to 256 screen formats for each partition.

When the assembler processes each series of control statements, it generates a string of object code referred to as a screen format control string. The assembler stores each screen format control string in the partition storage area, and places the address of each string in a system table.

During program execution, formatted key entry is initiated by a key entry command, the ENTR command. Each ENTR command specifies the format to be used to enter the record. The IBM 5280 searches the system table for the address of the screen format control string generated from the specified format. The IBM 5280 then processes the screen format control string in the same order that the source program control statements were written.

Besides the .SFMTST and .SFMTEND control statements, the screen format control series includes the following control statements:

| <b>Control Statement</b> | <b>Purpose</b>                                  |
|--------------------------|---|
| .SFMTPMT                 | To specify prompts                              |
| .SFMTCNS                 | To specify constant insert data                 |
| .SFMTFLD                 | To define each field and subfield of the record |
| .SFMTCTL                 | To specify keyboard, screen, and format control |

The sequence of a screen format for a typical key entry job could begin with a statement to display a prompt requesting the operator to enter a field of data. The next format statement could define the valid characters that the operator may enter into the field. As the operator enters the field, the IBM 5280 checks each input character to make sure it is valid according to the screen format statement that defines the field. Each valid input character is placed into an I/O buffer. The next screen format statement could move the cursor or the pointer in the I/O buffer that contains the current record. Then another prompt could request the operator to enter the next field.

When all the specifications of a screen format are processed, the complete input record is in the current record buffer. The IBM 5280 must then execute object code instructions to move the data from the current record buffer to registers or other storage areas. When the IBM 5280 has moved the data from the buffer, another ENTR may be issued, with the same or a different screen format specification.

## Prompts

In your source program, you label and initialize each prompt by using a .DC (declare) control statement. You must specify PRMT for the TYPE parameter and define the prompt message with the INIT parameter. Then, when you write a screen format using the .SFMT control statements, you specify the label of the prompt with a .SFMTPMT statement.

When the assembler processes your source statements, it stores the prompt labels in a system table. It stores the system table index for the prompt in the screen format control string.

During program execution, when the IBM 5280 encounters a prompt index while processing a screen format control string, it finds the address of the appropriate prompt at that index into the system table. It takes the prompt message from the storage address and displays it on the screen. The prompt message is not inserted into the current record buffer.

You can specify the screen position where each of your prompts are displayed. A prompt can be displayed in the standard fixed prompt location, which begins in column one of line two. You can specify a different line for the fixed prompt position by including the FPLC parameter of the .KBCRT statement. Each current fixed prompt replaces the previous fixed prompt on the prompt line. You can also have the prompt displayed at the current cursor position, or at a specified number of positions to the right or left of the current cursor position, or on the beginning of the next line. All of these options for the placement of your prompt are described under *.SFMT<sub>P</sub>MT Control Statement* in Chapter 3.

### **Constant Insert Data**

In your source program, you can specify constant data to be inserted into the current record buffer and displayed upon the screen during program execution. The constant data is labeled and initialized with a .DC control statement, with PRMT specified for the TYPE parameter. It is specified in a source screen format with a .SFMTCNS statement. The IBM 5280 finds the appropriate constant insert data by using the prompt system table. The insert is processed as if it were a prompt, except that the constant is displayed on the screen and inserted into the current record buffer.

### **Field Definitions**

You can define the individual fields of the record by including a .SFMTFLD statement for every field. The parameters of the .SFMTFLD statement specify the field length and the character set that is valid for the field. Other parameters can break a field down into a number of subfields, or indicate that the field is a data required, automatic duplicate, or right adjust field. Parameters can also specify display attributes that effect the individual field, such as blink, highlight, or underscore.

### ***Main Storage Duplicate and Store***

You can specify a main storage duplicate field, or a main storage store field, by including an MD or MS parameter in the .SFMTFLD statement that defines the field. The MD or MS parameter specifies the label of the main storage data area. This main storage data area must be allocated and labeled with a .DC statement that specifies MDUP for the TYPE parameter.

When a main storage duplicate field (MD) is entered, the contents of the specified main storage area are automatically copied into the field in the current record buffer if one of the following is true:

- The field is also specified as auto duplicate and absolutely automatic (AD, AA in the third FLDF position).
- The field is also specified as auto duplicate (AD in the third FLDF position) and the auto dup/skip mode is active.

If the field is defined only as MD, duplication can be initiated by pressing the Dup key. When the Dup key is pressed, the duplication starts at the current field position and continues to the end of the field.

When a main storage store field (MS) is exited, the contents of the field are automatically copied into the specified main storage location if one of the following is true:

- The field is also defined as absolutely automatic (AA in the third FLDF position).
- The auto dup/skip mode is active.

*Example:* The following declare control statements allocate and initialize a prompt and a constant insert and allocate a data area in main storage. The screen format control statements use the prompt, constant insert, and data area to illustrate a main storage store and main storage duplicate.

```
.DC LABEL=PTNAME TYPE=PRMT INIT='Name: ' ;
.DC LABEL=CONST1 TYPE=PRMT INIT='Hello ' ;
.DC LABEL=DUPNAME TYPE=MDUP LEN=20;
.
.
.SFMTST LABEL=PFT04 CNTL=MV;
.SFMPMT PRMT=SP,PTNAME ; display a standard position prompt
.SFMTFLD FLDF=A,20,AA MS=DUPNAME;
*The operator enters a name into the 20-byte alphabetic field, which is
*specified as absolutely automatic. The characters are displayed and placed
*into the I/O buffer as they are entered. When the field is exited, the
*contents of the field are stored into the main storage data area labeled
*DUPNAME because the AA is specified.
.SFMTCONS CNST=CONST1 BFPS=1 CSPS=NL;
*The constant is displayed on a new line, the I/O buffer pointer is incremented
*1 to skip 1 position in the buffer, and the constant is placed into the I/O
*buffer.
.SFMTFLD FLDF=A,20,AD,AA MD=DUPNAME ;
*When the cursor moves to the first position of this field, the name is
*automatically copied from the data area labeled DUPNAME into the I/O
*buffer and is displayed on the screen.
.
.
.
```

## Field Control

You can specify control of the screen, of the keyboard, and of the format with a .SFMTCTL control statement. The parameters of this control statement can specify display attributes for the screen, such as blink, reverse image, and nondisplay. Other parameters can enable or disable the Dup key or specify whether a field exit key is required to exit the current field. Other parameters can cause a field to be duplicated or stored, cause a conditional bypass of a portion of the format, or cause a secondary format to be processed.

### *Secondary Screen Format*

You can specify a secondary screen format series by including a .SFMTCTL control statement at the position where you want the secondary screen format to begin. The .SFMTCTL statement must have an ES parameter that indicates the label (LABEL parameter of the .SFMTST statement) of the secondary screen format.

The secondary screen format specification acts in a way similar to a subroutine call. When an ES parameter is encountered while the primary screen format series specifications are being processed, control goes to the first specification of the secondary screen format. All specifications of the secondary screen format series are processed. Then control returns to the primary screen format, to the statement following the ES parameter.

Only one level of secondary formats is allowed.

*Example:* In the following example, three screen formats are used to enter a record: the primary format FMT04, the secondary screen format FMT16, and the secondary screen format FMT17.

```
.SFMTST LABEL = FMT04; begin primary screen format.  
.SFMTPUT LABEL = PROMPT6; primary format displays a prompt.  
.SFMTCTL ES = FMT16; process complete screen format FMT16.  
.SFMTPUT LABEL = PROMPT7; primary format displays a prompt.  
.SFMTCTL ES = FMT17; process complete screen format FMT17.  
.SFMTPUT LABEL = PMTEOR;  
.SFMTEND; primary screen format ends.
```

### *Conditional Bypass*

You can specify a conditional bypass for any section of a screen format. Include a .SFMTCTL control statement with a CI parameter at the position in the screen format series where the bypass begins. Then include a .SFMTCTL statement with a CNTL = CE parameter at the position where the bypass section ends.

For the CI parameter, you must specify an indicator label followed by either ON or OFF. Use the label assigned by a .DCLIND control statement. When the IBM 5280 encounters the bypass specification, it checks the specified indicator. If the indicator is 1 and the CI parameter specified ON, or if the indicator is 0 and the CI parameter specified OFF, the IBM 5280 bypasses all field, display attribute, and prompt specifications between the CI and the CE specifications. However, the cursor and current record buffer pointer are moved past the space on the screen and in the current record buffer where the bypassed fields, display attributes, or prompts would have appeared. If the bypass specifications are encountered in a forward direction, the current field counter is incremented by the number of fields bypassed. If it is encountered in a backward direction, the current field counter is decremented. If an RG (return to program), BFPS (change buffer position pointer), CSPA (change screen position pointer), or a control specification to change status is encountered during bypass, it is processed as normal. If an ES (execute secondary format) specification is encountered, the fields and control specifications of the secondary format are processed as described above for a bypass.

*Example:* The secondary format FMT06 is not executed if the indicator CHECK10 is 1:

```
.SFMTST LABEL = FMT05 CNTL = MV;  
.SFMTCTL CI = CHECK10, ON; the indicator is labeled CHECK10.  
.SFMTCTL ES = FMT06; a secondary format specification.  
.SFMTCTL CNTL = CE; end bypass section.  
.br/>.br/.
```

Only one level of conditional bypass is allowed. Do not follow a CI parameter with a second CI parameter before a CE specification is included. However, you can have more than one bypass within a screen format series if each CI parameter is followed with a CE specification. Do not follow a CI parameter with a .SFMTST or a .SFMTEND control statement before a CE specification is included.

### Returning (RG) Exits

When you write a screen format in your source program, you may wish to temporarily interrupt key entry in order to have program operations performed. You can do this by including a CNTL=RG parameter in any .SFMT control statement except the .SFMTEND statement. The assembler sets a bit in the screen format control string whenever it encounters an RG specification in the source screen format. Then, when the IBM 5280 executes the screen format control string and encounters this bit in a forward or backward direction, it interrupts key entry and reports external status condition 4 or 5, respectively.

You could include an RG exit immediately following an input field you want to self-check, or immediately following an input field you want to add to a running total.

### EDIT FORMATS

Edit formats are used to reformat the fields of a record as the record is moved between main storage and the current record buffer. Each edit format is set up by one .FMTST control statement followed by one .FMTFLD for every field in the record. The .FMTFLD statement specifies the length of the field and the registers to or from which the field is moved. It can also specify editing for the field, such as the placement of a currency sign, decimal point, or minus sign. Each edit format must end with a .FMTEND control statement.

The edit formats are used for several data movement instructions to move the fields of the record to or from the current record buffer after the record has been read, or before it is written.

Edit formats may be used for read and write instructions for the diskette drive, communications line, and printer. When an edit format is specified in a read instruction, the record is reformatted as it is read into the current record buffer. The format edits the record, removing currency symbols and punctuation. The edit format then specifies the registers or storage locations into which each field is moved. For a write instruction, the format moves the fields from the specified registers or storage locations to the current record buffer and replaces the currency symbols and punctuation. The reformatted record is then written from the current record buffer to the diskette, printer or communications line.

### **Data Directed Formatting**

For input records, you can specify that the formatting is data directed. The .FMTST statement allows you to specify a control character and where the control character is located in the record. When you write a data directed read instruction, you specify an asterisk (\*) rather than a format label. Then, during execution of a read operation, the IBM 5280 selects the appropriate format by matching the control character of the input record to the first format that has the same control character specification.

### **FIELD MODIFICATION INDICATORS**

There are 32 field modification indicators: 1160-1191. Each indicator represents a field in the screen format, up to 32 fields. If there are more than 32 fields in the format, each indicator represents every 32nd field. 1160 represents field 0, field 32, field 64 and so on. A format level zero specification is represented by one indicator for the entire group of 1-byte fields.

When the IBM 5280 encounters an ENTR command, it sets each field modification indicator to zero. Each time the cursor is advanced or backspaced into a field, the IBM 5280 sets a bit in the partition control area to zero. If data is entered into the field, the IBM 5280 sets the bit to 1. When the cursor exits the field, the IBM 5280 ORs the bit with the field modification indicator that represents the field.

Constant inserts are assigned field modification indicators. Whenever the insert is moved into the I/O buffer and onto the screen, the corresponding field modification indicator is turned on.

If an external status condition occurs while the cursor is within a field, the corresponding field modification indicator has not yet been ORed with the bit in the partition control area; therefore, it may not indicate that the field has been modified in the current record.

If your program makes a change to a field in the current record in the I/O buffer, it is your responsibility to update the corresponding field modification indicator.

## **DISKETTE DATA MANAGEMENT**

Diskette operations for the IBM 5280 include operations to read, write, search, insert, and delete records that are stored in diskette data sets. The data sets may be sequential or key indexed data sets. In a sequential or key data set, the IBM 5280 can access records sequentially, in the order they were entered. In a sequential data set, the IBM 5280 can access records directly, by relative record number. In a key indexed data set, the IBM 5280 can access records directly by key. When a key indexed data set is opened, an index table of the keys is built automatically, or you can choose to build your own key index table.

### **Label Update**

By specifying a label update type data set, you can update the HDR1 labels and sectors 1 through 7 of track 0 on the diskette index as though it were a sequential update data set. For label update, the IBM 5280 treats each 128-byte diskette data set label as a record. The record number of the last label is both the EOD (end of data) and the EOE (end of extent) record number. By specifying a label update, erase-type data set, you can create labels without reading the existent label values. Only the index cylinders are accessed for the label update access method.

A device opened for label update may not have any open data sets. Similarly, a data set may not be opened on a device already open for label update.

### **Physical and Logical Buffers**

You must set up at least one physical buffer in main storage for any program that has I/O instructions. The physical buffer length must be a multiple of 128 bytes and must begin on a 128-byte boundary. You can use double buffering for minimal delays in interactive programs; set up a second physical buffer so the IBM 5280 can process data in one while an input or output operation is being performed with the other. For keyboard/display I/O, double buffers are required to duplicate fields of a previous record into the same field of a current record. The IBM 5280 keeps track of the buffers and the records that are in the buffers.

You can block your data sets for faster execution; set up a logical buffer, and the blocking and deblocking functions are performed automatically by the IBM 5280. Or you can omit the logical buffer and use pointer I/O to block and deblock logical records directly to and from the physical buffer.

### **Automatic Logical Buffering**

When the IBM 5280 opens a data set, it finds the address of the physical buffer and the logical buffer in the data set IOB. During I/O operations to or from a diskette data set, the IBM 5280 maintains a record counter to keep track of the record number of the logical record currently being processed, relative to the first record of the data set.

When the IBM 5280 is processing a sequential data set and encounters the first READ instruction, it reads the logical records from the diskette into the physical buffer until the physical buffer is filled. It then moves the first logical record from the physical buffer to the logical buffer. If the READ instruction specified an edit format, the fields of the record are edited and moved from the logical buffer according to the format. When the IBM 5280 encounters the second READ instruction, it moves the second logical record from the physical buffer to the logical buffer. No more data is read from the diskette until all the logical records currently in the physical buffer have been processed.

Output of sequential logical records is managed in the same way. When the IBM 5280 encounters the first write instruction, it writes the contents of the logical buffer to the physical buffer at the record position specified by the current record counter. Subsequent write operations place logical records into the physical buffer. The IBM 5280 automatically writes the contents of the physical buffer to the diskette.

### Pointer I/O

When you use pointer I/O, your program can access logical records directly from the physical buffer. This saves storage that is required for a logical buffer, and saves the time involved in moving the logical record from the physical buffer. Omit the logical buffer specification in the .DATASET control statement, and specify pointer (PTR) for the data set attribute. When you process your source program with the IBM 5280 assembler, the assembler places the address of the physical buffer into the IOB location reserved for the logical buffer address.

When the IBM 5280 is processing a sequential data set that specifies pointer I/O, and encounters the first READ instruction, it reads the logical records from the diskette into the physical buffer until the physical buffer is filled. The logical buffer address in the data set IOB points to the first logical record in the physical buffer. The logical record is not moved to another storage location but is processed directly from the physical buffer.

When the IBM 5280 encounters subsequent READ instructions, it updates the logical buffer address in the data set IOB so the address always points to the current logical record in the physical buffer. For sequential write operations, the logical buffer address is the address of the next logical record to be written.

You cannot use an edit format to edit and move the record fields when you use pointer I/O for diskette operations. However, you can place the address of the logical record into a binary register and use it to access the individual fields of the record using base displacement addressing. The address of the logical record is in the IOB at displacement hex 0C-0D. The IBM 5280 does not update this base address in the binary register; you must replace the address before each I/O operation.

## Keyed Data Sets

Keyed data sets can be read according to a specified key. When a keyed data set is opened, an index table is built from the record keys. The index table can be built either by the application program or automatically by the IBM 5280, depending on the TYPE parameter of the .DATASET control statement. If a KR (key indexed read) or KU (key indexed update) data set is specified, the IBM 5280 automatically builds the index table. If a KRN (key indexed read, no table build) or a KUN (key indexed update, no table build) data set is specified, the application program must build the index table. The .TABLE control statement is used to define the table. The parameters of the .DATASET control statement and the .TABLE control statement that are used for keyed data sets are as follows:

### *.DATASET Control Statement*

|       |  |
|-------|--|
| TYPE= | Specifies one of the following keywords: KR, KU, KRN, or KUN. In addition, may also specify ORD (records in ascending key sequence). |
| KPOS= | Specifies the position of the key in the record.   |
| KLEN= | Specifies the length of the key.   |
| TLOC= | Specifies the location of the index table.   |
| DLTA= | Specifies the density of the index table; optional for KR and KU data sets but mandatory for KRN and KUN data sets.                  |

### *.TABLE Control Statement*

|          |  |
|----------|--|
| LABEL=   | Specifies the location of the index table (also specified for the TLOC parameter of the .DATASET control statement).   |
| MAXM=    | Specifies the maximum number of entries that can be placed into the index table.   |
| ENTRIES= | Specifies the actual number of entries used in the table.  |
| ARGL=    | Specifies the length of the index entry. This length can be less than or equal to the length of the key, but it cannot be greater than the length of the key.  |
| BYPAS=   | Specifies the number of bytes to be associated with each index entry that are not part of the index itself. Valid entries are 0, 1, 2, or 3. If a nonzero entry is specified, the byte or bytes are used to hold the relative record number of the record that corresponds to the index entry. If 0 is specified, the index table entry contains only the index entry, and the relative record number is calculated from the DLTA specification and the relative position of the index within the index table. |

**Note:** These parameters are in addition to the parameters required or normally used for a data set, as specified in Chapter 3.

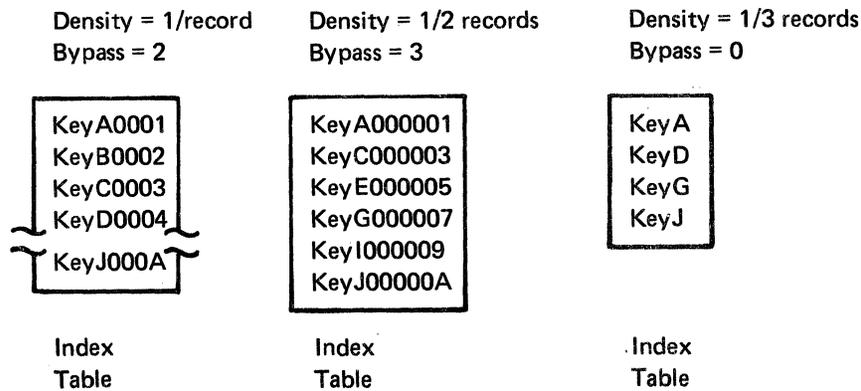
### Density of the Index Table

The density of the index table is specified with the DLT A parameter of the .DATASET control statement. The density specifies the number of logical records between index entries. Density may be 1 entry per record, 1 entry per track, 1 entry per 10 records, and so on. For KR and KU data sets, DLT A may be omitted; the density is calculated by the IBM 5280, using the length of the data set and the length of the index table. The first entry in the index table is always for the first record; the last entry is always for the last record. The following examples show how the IBM 5280 sets up index tables for various density and bypass specifications, using the sample data set in Figure 2-3.

|                         | KeyA/Data | KeyB/Data | KeyC/Data | KeyD/Data | KeyE/Data | KeyF/Data | KeyG/Data | KeyH/Data | KeyI/Data | KeyJ/Data |
|-------------------------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|
| Relative Record Number: | 1         | 2         | 3         | 4         | 5         | 6         | 7         | 8         | 9         | A         |

Figure 2-3. Sample Keyed Data Set Records

#### Examples:



### Reading a Keyed Data Set

When the IBM 5280 encounters a READ instruction for a keyed data set, the instruction specifies a decimal register that contains the key of the record to read. The IBM 5280 determines which entry in the index table is the highest entry that is lower than the specified key, and which entry is the lowest that is not higher than the specified key. The IBM 5280 determines the relative record number for the record that corresponds to each of these index entries, either by finding them in the index table (if a nonzero BYPAS is specified) or by calculating them (if BYPAS=0). The IBM 5280 searches the key positions of the records between these two relative record numbers until a matching key is found, and reads the record with the matching key. If the .DATASET control statement specifies the ORD parameter, the IBM 5280 uses a binary search; otherwise the IBM 5280 uses a sequential search. If the .DATASET control statement specifies a nonzero BYPAS and omits the ORD parameter, and if the index has one entry per record, the IBM 5280 searches the index for the first entry that matches the key, then reads the data set record that corresponds to that index entry.

### ***Updating a Keyed Data Set***

A keyed data set can be updated by using a READ instruction with a key specified to find the desired relative record number, then using a WRT instruction to update the record at that record position. The IBM 5280 does not update the index table.

### ***Adding to a Keyed Data Set***

A record can be: (1) inserted or (2) added to the end of a keyed data set. The IBM 5280 does not update the index table. Therefore, records must not be added or inserted unless the application program provides instructions to update or rebuild the index before reading the data set again.

### **Shared Data Sets**

Data sets that have share attributes specified in the .DATASET control statement can be used simultaneously by more than one program. Corresponding share attributes must be specified in every program that shares the data set. Improperly specified share attributes result in an access error external status.

If a record is added to a shared data set, the EOD of all IOBs for that data set are updated to the new increased value. Record inserts are not allowed for a shared data set. Any operation that reduces the EOD or EOE value of a shared data set is not allowed.

To make logical records more quickly available to be shared, you can use the quick release (QR) and early write (EW) parameters in the .DATASET control statement. These functions make direct access to update data sets more efficient because the logical record to be read or written will be released so that other programs can use it as soon as the operation is complete.

### **SCS Conversion Data Sets**

When you store a data set on a diskette, you can save diskette space by using an SCS conversion data set. An SCS conversion data set is defined by specifying SCS in the TYPE parameter of the .DATASET control statement. When an SCS conversion data set is processed, SCS (standard character string) control characters are inserted by the IBM 5280 to replace blank characters creating an SCS data set. When you write the SCS data set created by SCS conversion to a diskette, the record length on the HDR1 label must equal the block length. See *SCS Conversion*, under *Printer Instructions* in Chapter 4 for information about using SCS conversion.

### **Extending Data Sets**

When the data set type specified in the TYPE parameter of the .DATASET statement is sequential update (SU), sequential write (SW), or key indexed update (KU or KUN), records may be added at EOD to extend the data set. This requires that there be unused space between EOD and EOE.

In order to extend the data set's IOB, the current record pointer must first be pointed to EOD. This can be done by using the position (POSN) instruction, or reading sequentially until EOD is exceeded and a 3701 external status results. It is done automatically by the OPEN if either the sequential write (SW) or erase (ERS) parameters are included in the .DATASET statement. For binary searches, it is also automatically done if a 3703 (record not found) external status occurs.

Once positioned at EOD, write (WRT) instructions with the current record specification are used to actually add the records. After each such write the IOB current record pointer will be incremented to the new EOD so that it will not need to be repositioned prior to the next write. The EOD field of the IOB is also incremented and will be used to update the data set label by the CLOZ.

## SELF-CHECK

The IBM 5280 self-check facilities allow you to verify an input field at the time it is entered. The self-check function can detect incorrect keystrokes and character transpositions. It can also detect fraudulent entries.

The self-check facilities include the:

- .SELFCHK control statement, to define the self-check field, register, modulus and algorithm.
- GSCK instruction, to generate a unique self-check number for each self-check field.
- IF . . . CHK instruction, to verify the self-check field each time it is entered.

### *The Self-Check Field*

A self-check field consists of the self-check number, which may be one or two digits long, and the foundation. The foundation may consist of any characters available to your IBM 5280 keyboard. The self-check number may be assigned to any position in the field. If the self-check number is two digits long, the two digits must be adjacent.

The maximum length of a self-check field is 32 bytes.

### *The Self-Check Register*

The self-check field must be placed into a decimal register, or if the self-check field is larger than 16 bytes, a decimal double register. The IBM 5280 right-adjusts the field in the register. All unused register bytes are bypassed.

Figure 2-4 illustrates a self-check field with a self-check number one digit in length. The position assigned for the self-check number is the rightmost position of the decimal double register that acts as the self-check register.

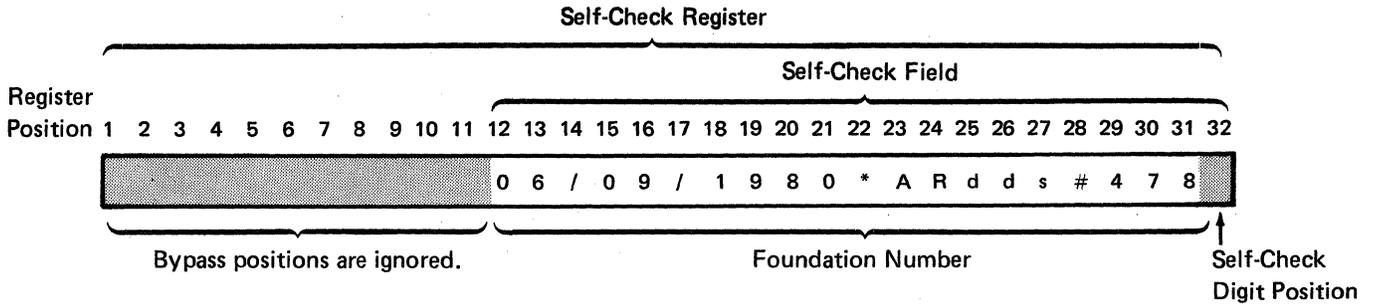
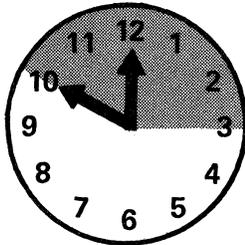


Figure 2-4. A Self-Check Field in a Decimal Double Register

### The Modulus

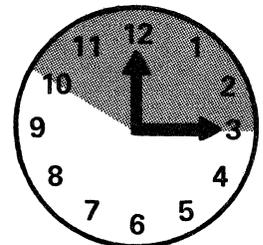
The self-check function uses a modular arithmetic. Modular arithmetic replaces a number with its remainder after it is divided by a fixed number. This fixed number is the modulus.

For example, clocks follow a modular arithmetic with modulus 12. If you add 5 to 10 o'clock, the sum is 15 modulus 12. Replace 15 with the remainder after it is divided by 12:



10 o'clock + 5

$$15 \text{ modulus } 12 = 12 \overline{)15} \begin{array}{r} 1 \\ \underline{12} \\ 3 \end{array} \begin{array}{l} \text{Remainder} \\ \downarrow \\ 3 \text{ Replaces } 15. \end{array}$$

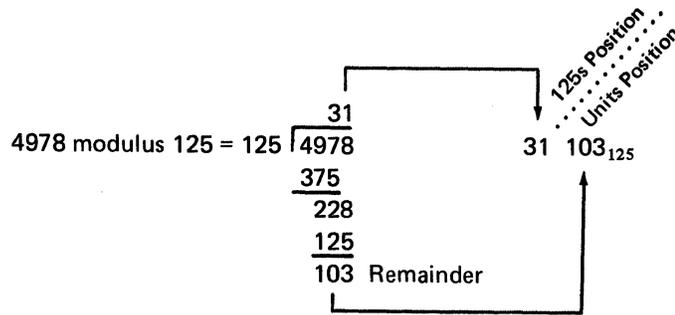


3 o'clock

Stated another way, modular arithmetic replaces a number with the value in the units position when the number is expressed in the base of the modulus. You can convert a number to the base of the modulus by dividing it by the modulus and using the remainder for the value of the units position:

$$15_{10} \begin{array}{l} \text{10s Position} \\ \dots \\ \text{Units Position} \end{array} \quad 15 \text{ modulus } 12 = 12 \overline{)15} \begin{array}{r} 1 \\ \underline{12} \\ 3 \end{array} \begin{array}{l} \text{Remainder} \\ \uparrow \\ 1 \quad 3_{12} \end{array} \begin{array}{l} \text{12s Position} \\ \dots \\ \text{Units Position} \end{array}$$

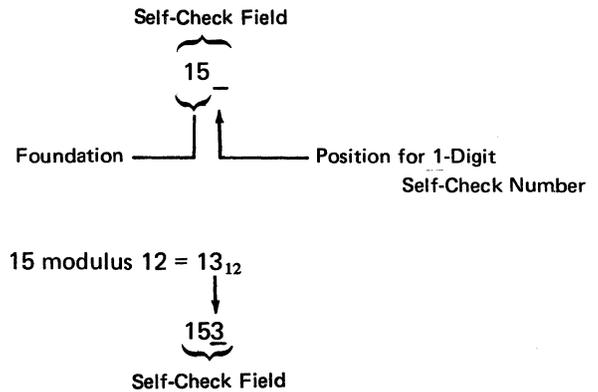
The value in the units position can be any number up to the modulus minus one. For example, if the modulus is 125, the units position can contain any number up to 124:



### The Self-Check Algorithm

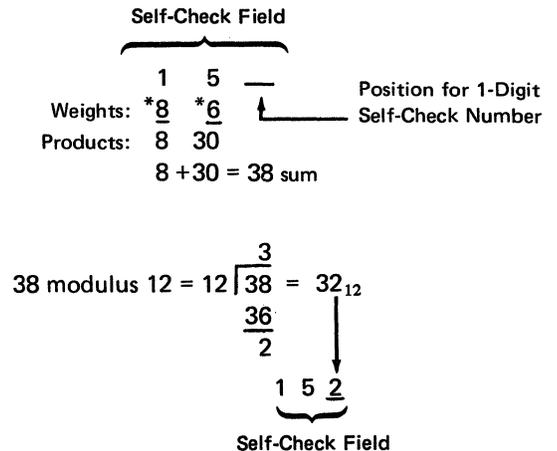
The basic steps of a self-check algorithm can be illustrated as follows, using 15 for the foundation and 12 for the modulus:

1. Convert a number to the base of the modulus.
2. The value in the units position is the self-check number.



Usually, however, the number converted to the base of the modulus is not the foundation, but a number derived from the individual characters of the foundation. The most common way to derive a number from the characters of the foundation is to:

1. Multiply the contents of each position in the foundation by a weight.
2. Add the resulting products.
3. Convert this sum of the products to the base of the modulus.
4. The value in the units position is the self-check number.



Numerous variations can be added to these basic steps to design a unique algorithm. These variations include:

1. **Input translation.** You can translate each character in the foundation to a specified numeric value by using an input translate table. Input translation is useful to assign a specific value to each alphabetic or special character in the foundation.
2. **Product table.** You can find the product for the numeric value of each foundation position by looking it up in a table rather than multiplying it by a weight.
3. **Output translation.** You can translate the generated self-check number to any specified output characters by using an output translate table. This is useful if the self-check number can be up to two digits long (for modulus 11-100) and only one self-check number position is used in the self-check field, or if the self-check number can be up to three digits long (for modulus 101-127).

### Choosing Your Algorithm

Appendix C shows how the IBM 5280 manipulates the self-check field according to various parameters of the .SELFCHK control statement. Use this appendix to select the variations you want for your unique algorithm.

If you do not want to design your own algorithm, you may use one of the IBM-supplied algorithms available with the IBM 5280. These algorithms are referred to as Standard Modulus 10 and Standard Modulus 11. If you use Standard Modulus 10 or 11, the IBM 5280 assumes that your self-check field is as follows:

- The self-check number is one digit long.
- The self-check number is in the rightmost position of the self-check field.
- The foundation may be from 2 to 31 characters long.

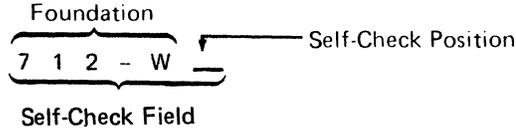
For Standard Modulus 10 or 11, any IBM 5280 keyboard character may be included in the foundation. The numeric value of each foundation position is determined by the low-order 4 bits of the EBCDIC representation of the contents of that position. If the low-order 4 bits are 0-9, the numeric value is 0-9. If the low-order 4 bits are A-F, the numeric value is 0.

*Example:*

|             |    |    |    |    |    |    |
|-------------|----|----|----|----|----|----|
| Foundation: | 6  | 9  | 7  | *  | X  | Y  |
|             | ↓  | ↓  | ↓  | ↓  | ↓  | ↓  |
| EBCDIC:     | F6 | F9 | F7 | 5C | E7 | E8 |
|             | ↓  | ↓  | ↓  | ↓  | ↓  | ↓  |
| Value:      | 6  | 9  | 7  | 0  | 7  | 8  |

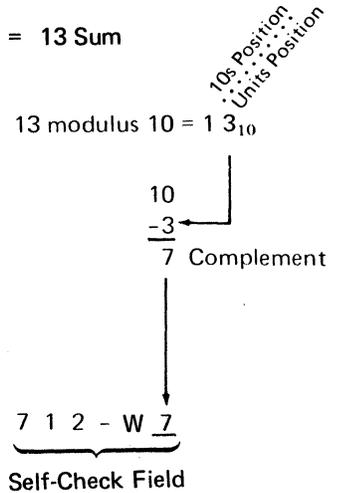
**Standard Modulus 10**

Standard Modulus 10 is designed to detect single incorrect keystrokes and single transpositions. If you use Standard Modulus 10, the IBM 5280 performs the following operations to generate a self-check number for this sample self-check field:



|   |         |         |         |         |         |             |
|---|---------|---------|---------|---------|---------|-------------|
|   | 7       | 1       | 2       | -       | W       | -Foundation |
| 1. Find the numeric value for each foundation position.   | ↓<br>F7 | ↓<br>F1 | ↓<br>F2 | ↓<br>60 | ↓<br>E6 | -EBCDIC     |
|   | ↓<br>7  | ↓<br>1  | ↓<br>2  | ↓<br>0  | ↓<br>6  | -Value      |
| 2. Multiply each value by the corresponding weight. The weights alternate between 2 and 1, with 2 always in the rightmost position of the foundation. | 7       | 1       | 2       | 0       | 6       |             |
|   | *2      | *1      | *2      | *1      | *2      | -Weights    |
|   | 14      | 1       | 4       | 0       | 12      | -Products   |
|   | 1+4     | +1      | +4      | +0      | +1+2    | = 13 Sum    |

3. Add each digit of the resulting products.
4. Convert the sum to the base of the modulus.
5. Subtract the value in the units position from the modulus to find the complement.

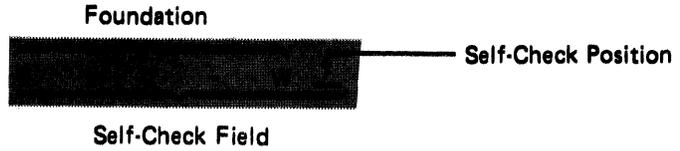


**Note:** If the remainder is 0, the complement is 0. If the remainder is 1, the character combination in the foundation does not have a valid self-check number. You must eliminate these character combinations when you generate self-check numbers.

6. The complement is the self-check number.

**Standard Modulus 11**

Standard Modulus 11 is designed to detect single incorrect keystrokes, single transpositions, and double transpositions. If you use Standard Modulus 11, the IBM 5280 performs the following operations to generate a self-check number for this sample self-check field:



- |    |   |    |    |    |    |    |             |
|----|---|----|----|----|----|----|-------------|
| 1. | Find the numeric value for each foundation position.  | 6  | 1  | 2  | -  | W  | -Foundation |
|    |   | ↓  | ↓  | ↓  | ↓  | ↓  |             |
|    |   | F6 | F1 | F2 | 60 | E6 | -EBCDIC     |
|    |   | ↓  | ↓  | ↓  | ↓  | ↓  |             |
| 2. | Multiply each value by the corresponding weight. Weights are the numbers from 2 to 7, starting in the rightmost position of the foundation and going leftward. Repeat the numbers if necessary. | 6  | 1  | 2  | 0  | 6  | -Value      |
|    |   | ↓  | ↓  | ↓  | ↓  | ↓  |             |
|    |   | *6 | *5 | *4 | *3 | *2 | -Weights    |
|    |   | 36 | 5  | 8  | 0  | 12 | -Products   |

3. Add the whole number products.  $36 + 5 + 8 + 0 + 12 = 61$  Sum

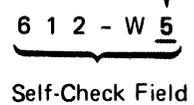
4. Convert the sum to the base of the modulus.

$61 \text{ modulus } 11 = 5 \text{ } 6_{11}$

5. Subtract the value in the units position from the modulus to find the complement.

$11 - 6 = 5$  -Complement

**Note:** If the remainder is 0, the complement is 0. If the remainder is 10, the character combination in the foundation does not have a valid self-check number. You must eliminate these character combinations when you generate self-check numbers.



6. The complement is the self-check number.

11s Position  
Units Position

### **Using the GSCK Instruction**

The GSCK instruction is used to create a new file of fixed numbers, such as account numbers that require self-checking each time an operator enters them. Your program must move the foundation into the self-check register before it issues the GSCK instruction. You must include a .SELFCHK control statement in your program if you use this instruction, or unpredictable results may occur.

During the execution of a GSCK operation, the IBM 5280 (using the algorithm you defined with the .SELFCHK control statement) performs manipulations upon the foundation to generate a self-check number. When the self-check number is generated from the foundation, the IBM 5280 can place the self-check number into the self-check register. The foundation and its self-check number made up the complete self-check field.

### **Using the IF . . . CHK Instruction**

The IF . . . CHK instruction is used to verify a self-check field. When an operator enters a self-check field, your program must move the field into the self-check register before it issues the IF . . . CHK instruction. You must include a .SELFCHK control statement in your program if you use this instruction, or unpredictable results may occur.

During the execution of an IF . . . CHK operation, the IBM 5280 again uses your algorithm to generate a self-check number from the foundation currently in the self-check register. The IBM 5280 then compares this self-check number with the current contents of the self-check number position in the self-check register. The comparison fails when the characters now in the self-check register do not match the characters that were in the same register positions when the GSCK operation was performed.

1. The first part of the document discusses the importance of maintaining accurate records of all transactions and activities. It emphasizes that this is crucial for ensuring transparency and accountability in the organization's operations.

2. The second part of the document outlines the specific procedures and protocols that must be followed when recording transactions. It details the steps from initial entry to final review and approval, ensuring that all data is entered correctly and consistently.

3. The third part of the document addresses the role of technology in record-keeping. It discusses the benefits of using digital systems for data storage and retrieval, as well as the necessary security measures to protect sensitive information from unauthorized access.

4. The fourth part of the document focuses on the importance of regular audits and reviews. It explains how these processes help identify discrepancies, correct errors, and ensure that the records remain up-to-date and accurate over time.

5. The fifth part of the document provides a summary of the key points discussed and offers final recommendations for implementing and maintaining an effective record-keeping system. It stresses the need for ongoing training and support for all staff involved in the process.

The IBM 5280 assembler language control statements provide control information to the system and allocate registers and data areas in main storage. Certain control statements must occur in a specific order. Others may occur anywhere throughout the program, interspersed with instruction statements. The prescribed order is indicated in Figure 3-1.

### To Initialize the Partition Control Area

|        |  |
|--------|--|
| .START | One is required as the first statement of a program, except for optional listing control statements, to indicate the start of the program. |
| .KBCRT | One is required as the second statement, to specify keyboard/display control parameters.   |
| .EDITC | One is allowed as the third statement, to specify the edit control characters.   |

### To Declare and Label Data Areas

|         |   |
|---------|---|
| .DC     | One or more are allowed to allocate, label, and initialize any kind of data area.                         |
| .DCLBR  | One or more are allowed following the .KBCRT statement, to allocate and label binary registers.           |
| .DCLDR  | One or more are allowed to allocate and label decimal registers, and should follow any .DCLBR statements. |
| .DCLIND | One or more are allowed anywhere to label user indicators.  |
| .EQUATE | One or more are allowed anywhere to label registers, constants, expressions, and system registers.        |

### To Set Up and Initialize Device Control Areas

|          |   |
|----------|---|
| .COMM    | One is allowed to specify the characteristics of the communications line. |
| .DATASET | One or more are allowed to specify the characteristics of a data set.     |

### To Set Up and Label Tables

|         |   |
|---------|---|
| .TABLE  | One or more are allowed anywhere to define the parameters of a data table. <i>All .TABLE statements must be contiguous.</i> |
| .LABTAB | One or more are allowed anywhere to set up a label table of specified subroutine addresses.                                 |
| .SYSTAB | One or more are allowed anywhere to specify the label and location of system tables or the subroutine stack.                |

Figure 3-1 (Part 1 of 2). Control Statement Summary

### To Set Up Edit Formats

|         |  |
|---------|--|
| .FMTST  | One or more allowed to begin each edit format specification.   |
| .FMTFLD | One or more allowed following a .FMTST or another .FMTFLD statement, to define an edit format field. |
| .FMTEND | One is required as the last statement of an edit format specification.                               |

### To Set Up Screen Formats

|          |  |
|----------|--|
| .SFMTST  | One is required for each screen format, to indicate the start of a screen format control specification.  |
| .SFMTCTL | One or more are allowed within a screen format series, to specify control operations.  |
| .SFMPMT  | One or more are allowed within a screen format series, to specify a prompt to display.   |
| .SFMTCNS | One or more are allowed within a screen format series, to specify a constant to display and to place into the I/O buffer for the current record. |
| .SFMTFLD | One or more are allowed within a screen format series, to define a key entry field.  |
| .SFMTEND | One is required as the last statement of each screen format series, to end the screen format.  |

### To Control the Assembly Listing

|           |  |
|-----------|--|
| .TITLE    | One is allowed to specify a heading to be printed on each page of the listing. |
| .EJECT    | One or more are allowed to begin printing on the next page.                    |
| .SPACE    | One or more are allowed to skip one or more lines.                             |
| .PRINTOFF | One or more are allowed to suppress assembly listing.                          |
| .PRINTON  | One or more are allowed to continue assembly listing after PRINTOFF.           |

### Miscellaneous Control

|          |  |
|----------|--|
| .INCLUDE | One or more are allowed to insert another data set into the program.                                     |
| .SELFCHK | One is allowed to set up the self-check control area and define the self-check algorithm.                |
| .XTRN    | One is required if the program uses any common functions, to specify the labels of the common functions. |
| .END     | One is required following all control statements and instructions, to indicate the end of the program.   |

Figure 3-1 (Part 2 of 2). Control Statement Summary

The .TABLE control statements are the only ones that must be consecutive in your source program. However, for best performance from the assembler, place statements of the same kind together. For example, place all .FMT statements together, all .LABTAB statements together, and all .DC statements together. Place the .XTRN statement toward the beginning of the source program.

Control statements are listed in this chapter by the type of function the control statement performs. The control statements are divided as indicated in Figure 3-1. The syntax, parameters, and parameter keywords are described for each control statement.

## FORMAT

Each control statement is written for a length of 72 positions per line. Each control statement must begin with a period (.) in the first position, followed by an uppercase statement name and its associated parameters. The parameters may be in any order and are separated by one or more blanks. Each parameter consists of an uppercase parameter name, an equal sign, and a parameter value. The parameter value may be one or more fields, with each field identified by the order of its appearance on the line. The parameter fields are separated by commas or parentheses. Each control statement must end with a semicolon (;). The general format of a control statement is as follows:

```
.NAME PARAM1=XX PARAM2=XX,XX PARAM3=(XX,XX),(XX,XX) ;
```

Control statements may be continued from one line to the next by stopping between two parameters. The statement is continued on the next line. Each parameter should be complete on one line unless the parameter contains a sublist of keywords. In this case, the sublist may be interrupted after a comma between keywords as in the following example.

*Example:*

```
.NAME PARAM1=XX PARAM2=XX,XX PARAM3=XX,  
XX PARAM4=XX ;
```

The end of the statement is always determined by the semicolon.

## Blanks

Optional blanks may be placed before or after an equal sign or parentheses. Blanks may follow a comma but must not precede a comma. Blanks are not allowed within a control statement field; however, one or more blanks must separate fields if no other delimiter is used.

## Comments

A comment may be included on any line, following the semicolon. An entire line may be designated as a comment line by placing an asterisk (\*) in the first position of the line. A comment line may be included before the .START statement and before print control statements.

*Examples:*

```
.NAME PARAM1=XX ; This is a comment on a statement line.
```

```
*This is a comment line.
```

## INITIALIZE THE PARTITION CONTROL AREA

The IBM 5280 provides, uses, and updates much of the partition control information during program execution. However, the assembler initializes certain control areas during assembly, using the following control statements. This control information is used by the IBM 5280, but it is not changed during program execution.

The `.START` control statement is mandatory for every program, and the `.KBCRT` statement is mandatory for every program that uses keyboard/display I/O. The `.EDITC` statement is optional.

### `.START` Control Statement

```
.START [ENTRY=  PNAM=  OPTION=  ORG=
        MCHK=   TMSL=  RGLT=] ;
```

The mandatory `.START` control statement must be the first control statement of every program. It specifies program name, origin, error routines, register usage limits, and time slice factor.

| Parameter Name | Description   |
|----------------|---|
| ENTRY          | Entry point; the label of the instruction to be executed first. If omitted, it defaults to the first executable instruction encountered.  |
| PNAM           | Program name; 8-character field of alphabetic or numeric characters, enclosed in single quotes. The first character must be alphabetic. It defaults to PROGRAM.<br><br>This name is printed in the listing header and included in the object code. Do not specify any unprintable code for this parameter.  |
| OPTION         | Option; informs the assembler whether the code being assembled is a main program (MAIN) or a separately assembled subroutine (SUB). It defaults to MAIN.  |
| ORG            | Origin; mandatory with parameter <code>OPTION=SUB</code> , to assemble a transient overlay. It specifies the actual location (relative to start of partition) to place the first instruction of executable code. Control statements preceding it are used to define shared labels but do not produce object code. Given in decimal or hex, ORG does not have to be a 256-byte boundary unless you are writing a partial overlay.            |
| MCHK           | Main program check errors; the label of the routine that has been coded to handle program errors. If omitted, the program is terminated if a program check error occurs. If you use the common function from the common area, specify the name <code>CFPGMCHK</code> , and include this name in an <code>.XTRN</code> control statement. See <i>Program Check Errors</i> in Chapter 6 for more information about main program check errors. |

|      |  |
|------|--|
| TMSL | Time slice factor; a 1-byte field that specifies the maximum length of time the controller executes instructions within the current partition. Acceptable time limits are 4-60 milliseconds. Default is 12 milliseconds.   |
| RGLT | Register limit; the number of 16-byte blocks you want to reserve to use as undeclared decimal or binary registers. The assembler skips over these 16-byte blocks during allocation of space. The default of four reserves the first four decimal registers, which includes all indicators, or the first 32 binary registers. |

**Note:** You do not have to reserve registers that you declare and label with a .DC, .DCLBR, or .DCLDR control statement. Reserve only the registers you use in .EQUATE control statements or the registers that you refer to by register number rather than by label in instructions.

*Examples:*

```
.START PNAM='PAYROL'
ENTRY=BEGIN MCHK=MCHKER RGLT=8;
.START;
```

#### **.KBCRT Control Statement**

```
.KBCRT CRBA= { ETAB=
               ELAB= }
        [PRBA= TRAP= MODE= AFIL= NFIL= BLCK =
        FPLC= NMIN= HLIN= CNTR= FUNC= SCREEN=
        RGBUFF= ] ;
```

The .KBCRT statement specifies the location of the keyboard/display record buffer, the handling of keyboard/display error conditions, and the initialization of keyboard/display parameters.

This control statement is mandatory in each main program.

#### **Parameter**

| <b>Name</b> | <b>Description</b>   |
|-------------|--|
| CRBA        | Current record buffer; the label assigned by a .DC statement for the I/O buffer that holds the current record. This parameter is mandatory.  |
| ELAB        | Exception label (either ELAB or ETAB must be specified); the label of a subroutine that handles all external status conditions.  |
| ETAB        | Exception table (either ELAB or ETAB must be specified); the label of a label table set up by a .LABTAB statement. Each entry in the label table is the address of a subroutine that handles one specific external status condition. |

| <b>Parameter Name</b> | <b>Description</b>  |                |             |   |       |   |        |   |        |   |         |   |       |   |               |
|-----------------------|---|----------------|-------------|---|-------|---|--------|---|--------|---|---------|---|-------|---|---------------|
| PRBA                  | Previous record buffer; the label assigned by a .DC statement for the I/O buffer that holds the previous record. This parameter is optional; if omitted, the IBM 5280 sets the address for the PRBA to the CRBA address.  |                |             |   |       |   |        |   |        |   |         |   |       |   |               |
| TRAP                  | <p>Trap; one or more numbers that represent certain functions normally handled by the IBM 5280 that you want to process with your own subroutine. The number assignments are described in Appendix D under <i>Keyboard Functions: EBCDIC Codes and Bit Numbers</i>.</p> <p>The format of the input is TRAP=BITn, . . . as follows:</p> <p style="text-align: center;">TRAP=BIT1,BIT5</p> <p>This code causes an external status condition to occur whenever a shifted Cmd key (BIT1) or an Ins key (BIT5) is pressed.</p>   |                |             |   |       |   |        |   |        |   |         |   |       |   |               |
| MODE                  | <p>Mode of entry; one of the following keywords to specify the mode of entry.</p> <table border="0" style="margin-left: 40px;"> <thead> <tr> <th style="text-align: left;"><b>Keyword</b></th> <th style="text-align: left;"><b>Mode</b></th> </tr> </thead> <tbody> <tr> <td>E</td> <td>Enter</td> </tr> <tr> <td>U</td> <td>Update</td> </tr> <tr> <td>V</td> <td>Verify</td> </tr> <tr> <td>D</td> <td>Display</td> </tr> <tr> <td>R</td> <td>Rerun</td> </tr> <tr> <td>Y</td> <td>Rerun/display</td> </tr> </tbody> </table> <p>See <i>Modes of Entry</i> in Chapter 2 for a description of the modes. Default is enter (E) mode.</p> | <b>Keyword</b> | <b>Mode</b> | E | Enter | U | Update | V | Verify | D | Display | R | Rerun | Y | Rerun/display |
| <b>Keyword</b>        | <b>Mode</b>   |                |             |   |       |   |        |   |        |   |         |   |       |   |               |
| E                     | Enter   |                |             |   |       |   |        |   |        |   |         |   |       |   |               |
| U                     | Update  |                |             |   |       |   |        |   |        |   |         |   |       |   |               |
| V                     | Verify  |                |             |   |       |   |        |   |        |   |         |   |       |   |               |
| D                     | Display   |                |             |   |       |   |        |   |        |   |         |   |       |   |               |
| R                     | Rerun   |                |             |   |       |   |        |   |        |   |         |   |       |   |               |
| Y                     | Rerun/display   |                |             |   |       |   |        |   |        |   |         |   |       |   |               |
| AFIL                  | Alphabetic fill character; used for padding the left end of the right-adjust alphabetic fill fields. It defaults to blank (hex 40).   |                |             |   |       |   |        |   |        |   |         |   |       |   |               |
| NFIL                  | Numeric fill character; used for filling the left end of right-adjust numeric fill fields. It defaults to zero (hex F0).  |                |             |   |       |   |        |   |        |   |         |   |       |   |               |
| BLCK                  | Blank check character; the character that is not permitted in a blank checked field. It defaults to blank (hex 40).   |                |             |   |       |   |        |   |        |   |         |   |       |   |               |
| FPLC                  | Fixed prompt location; indicates the row at which the fixed prompts occur on the screen. It defaults to row 2.  |                |             |   |       |   |        |   |        |   |         |   |       |   |               |

**Parameter  
Name**

**Description**

**NMIN**

Normal display attribute; specified as a 1-byte constant, this entry determines the display attributes of fields not currently being processed; the attribute is in effect after the cursor exits the field. This parameter is normally omitted if you specify display attributes with the DSPLY parameter in your screen format control statements. Each bit specifies an attribute as follows:

| <b>Bit</b> | <b>Meaning If 1</b>            |
|------------|--------------------------------|
| 0-2        | Not assigned (Bit 2 always on) |
| 3          | Column separators displayed    |
| 4          | Blink screen                   |
| 5          | Underline                      |
| 6          | High intensity                 |
| 7          | Reverse image                  |

The value for NMIN defaults to no high intensity. If NMIN is specified, HLIN should also be specified.

**HLIN**

Display attributes; specified as a 1-byte hex constant, this entry determines the display attributes of the input field being processed; the attribute is in effect when the cursor moves to the first position of the field. This parameter is normally omitted if you specify display attributes with the DSPLY parameter in your screen format control statements. Each bit specifies an attribute, as described for NMIN. The value for HLIN defaults to no highlighting. If HLIN is specified, NMIN should also be specified.

**CNTR**

Current-position counter; the value of the internal register maintained by the system during formatted data entry. This value is displayed in columns 3-6 of the status line while an ENTR is being processed. The counter reflects one of the four values listed below. It defaults to K.

- K — current keying position of next keystroke to enter, relative to first position of the record.
- B — current position in the current record buffer, relative to the beginning of the buffer, where the next data character entered will be stored.
- C — current relative position of the cursor on the screen.
- F — current position within the current field, relative to the beginning of the field, where the next data character entered will be stored.

- FUNC** Functions; one to three functions may be specified. If more than one is specified, separate them with a comma:
- A — auto duplicate/skip (see *Auto Duplicate/Skip* in Chapter 2).
  - C — click software function keys.
  - D — do not display fixed prompts on the screen.
  - R — auto enter (see *Auto Enter* in Chapter 2).
  - S — on keystroke error, determine if status line is currently displayed. If not, display it and on error reset remove it from screen. (See *Nondisplay of the Status Line* in Chapter 2.)
  - X — alternate record advance (see *Alternate Record Advance* in Chapter 2).
- SCREEN** Screen positions; the number of valid screen positions you wish to use in the current program. Valid entries are 480, 960, or 1920. Default is 480.
- RGBUFF** Keystroke buffering during RG exit; determines if keystrokes will be buffered when RG exit code is active. This is an optional parameter. Specify either YES (buffering will occur) or NO (buffering will not occur). The default is NO.

Examples of input:

```
.KBCRT CRBA=BUFR01 PRBA=BUFR02 ELAB=ERRTN AFIL=/ NFIL=X'40';
.KBCRT CRBA=BUFFX PRBA=BUFFY ETAB=RTN1 TRAP=BIT4,BIT8
      FPLC=4 CNTR=B FUNC=C, S, D;
```

#### **.EDITC Control Statement**

```
.EDITC [EDCUR= EDDEC= EDCOM= EDCNT=] ;
```

The .EDITC control statement specifies the edit control characters. These characters are stored in the partition I/O control block and are used by other control statements.

| Parameter Name | Description   |
|----------------|---|
| EDCUR          | Edit currency sign; a two-character field that specifies the currency symbol. It defaults to C '\$'.  |
| EDDEC          | Edit decimal character; a one-character field that separates the decimal portion of a number from the fraction. (It may be specified by X' ', but numerics or hex codes of hex F0 or greater must not be used.) It defaults to C'.' |

|       |  |
|-------|--|
| EDCOM | Edit separator character; a one-character field that specifies the symbol used to separate groups of digits in an edited field. (It may also be specified as X' ', but numerics or hex codes of hex F0 or greater must not be used.) It defaults to C',. |
| EDCNT | Edit control count; a one-character field that specifies the number of digits between the occurrence of the edit separator character. It defaults to 3.  |

**Note:** All characters must be enclosed in single quotes; they may also be specified as hex values.

*Example:*

```
.EDITC EDCUR=C'FR' EDDEC=',' EDCOM=X'40' ;
```

## DECLARE AND LABEL DATA AREAS

Several different control statements assign labels to data areas, but each statement has a specific purpose. Use the following to help decide which control statement to use.

- Use .DC to label and initialize one data area or register.
- Use .DCLIND to label up to 30 program indicators.
- Use .DCLBR to label up to 30 uninitialized binary registers.
- Use .DCLDR to label up to 30 uninitialized decimal registers.
- Use .EQUATE to label initialized decimal or binary registers reserved by the RGLT parameter of the .START statement. You must specify register numbers.
- Use .EQUATE to label system indicators. You must specify the indicator number.
- Use .EQUATE to label a constant or expression.

### .DC Control Statement

```
.DC [LABEL=   TYPE=   LEN=   LEVL=   LOC=
     BDY=     PREFIX=  DISP=  INIT=] ;
```

The .DC control statement specifies the allocation of data areas and storage structures, and assigns labels to decimal and binary registers. It also allows you to initialize the data set.

When declaring registers, it is important to declare all binary registers first. If you declare decimal registers or storage areas before you declare all binary registers, you can fill the 256 bytes of the partition that contains the 128 two-byte binary registers. An attempt to declare a binary register beyond the binary register limit causes an error message to be written on the assembly listing.

When using boundary alignment, remember that storage is assigned sequentially and any bytes which are unused between boundaries cannot be recovered. Declaring two 1-byte field, both on 256-byte boundaries, results in the loss of 255 bytes between the fields. These bytes cannot be recovered. Statistics are maintained to indicate how many bytes are lost due to boundary alignment. These statistics are written to the assembly listing.

**Note:** Any unassigned storage byte is initialized to hex 00.

| Parameter Name | Description   |
|----------------|---|
| LABEL          | Label; the name that refers to the storage area or register. This is an optional parameter, unless TYPE is PRMT or MDUP.  |
| TYPE           | Type; specified as DEC (decimal register), BIN (binary register), STOR (storage area), PRMT (prompt) or MDUP (duplication). It defaults to STOR.<br><br>The PRMT type allocates and initializes space for character strings that are used as either prompts or constant insert data during data entry.<br><br>The MDUP parameter allocates space in main storage for duplication during data entry.<br><br>A table containing the addresses for these PRMT and MDUP areas is created and the address of the table is stored in the keyboard/display I/O control block. A source statement refers to these areas by using the label, and the system finds the address for the area in this table.                |
| LEN            | Length; the length of the area being declared, specified in bytes. This is an optional parameter. If TYPE=DEC, it defaults to 16. If TYPE=BIN, it defaults to 2. For other types, the length defaults to the length of the INIT field, or to 1 byte if no INIT field is specified.<br><br>A length of zero is valid for a declare that uses no space. For example, a .DC that specifies 0 length, and specifies LOC (location) acts in the same way as the origin (ORG) parameters in the .START control statement.<br><br><b>Note:</b> The LEN specification overrides the actual length of an INIT entry. If LEN is less than the number of bytes necessary for the INIT data, some of the INIT data is lost. |

| Parameter Name | Description   |
|----------------|---|
| LEVL           | <p>Level; the structure level of the area being declared. Variable leveling builds a storage structure if a level-1 declare is followed by one or more level-2 declares; the level-2 areas are overlaid into the level-1 area. The level-2 declare does not affect the current location counter. If the level-2 area extends beyond the level-1 area an overflow condition message results. However, the overflow message is suppressed if the preceding level-1 declare had the length specified as zero; this allows you to assign labels without advancing the current location counter. If omitted, it defaults to level-1.</p>   |
| LOC            | <p>Location; a number that specifies the relative location for the area being declared, or the label of a previously defined location. When LOC is specified, the assembler will reset its internal location counter. After processing a .DC statement with LOC specified, the location counter will equal LOC + LEN. Subsequent storage will be assigned starting at this location. If this parameter is omitted, the area is declared at the next sequential location.</p> <p><b>Note:</b> LOC cannot be specified with a LEVL=2 declare.</p>   |
| BDY            | <p>Boundary alignment; the type of boundary on which to locate the first byte of the data area. Specify a numeric value that is valid for the particular type of data area. See <i>Main Storage Boundary Alignment</i> in Chapter 1 for more information. An easy way to remember boundary alignment requirements is: an n-byte boundary is evenly divisible by n. For example, a binary register can begin on any boundary that is evenly divisible by 2, a decimal register can begin on any boundary that is evenly divisible by 16, and a buffer can begin on any boundary that is evenly divisible by 128.</p> <p>If BDY is omitted, the boundary defaults to the next sequential boundary that is appropriate for the specified TYPE.</p> <p><b>Note:</b> BDY cannot be specified if LEVL=2 has been specified.</p> |
| PREFIX         | <p>Prefix; a 1 or 2 character prefix that may be added to a level-1 declare. All level-2 declares may then be copied from a single statement to define an identical structure several times. The character or characters specified for the level-1 area attaches to the level-2 labels as a prefix to prevent duplicate labels. This is an optional parameter valid only for LEVL=1 declares.</p>   |
| DISP           | <p>Displacement; used only with LEVL=2 declares. Displacement is the number of bytes into the last level-1 area where this level-2 area is defined. If omitted, the subfields are contiguous.</p>   |

| Parameter Name | Description   |
|----------------|---|
| INIT           | <p>Initialization; numeric or character data may be used to initialize the data area. The INIT parameter must be complete on a single line.</p> <p>Decimal data is specified as decimal numbers with no quotes.</p> <p>Character data must be enclosed in single quotes. It may be preceded by an uppercase C, or the C may be omitted.</p> <p><b>Note:</b> If you are sending data to a printer, do not use character data with an EBCDIC representation of lower than hex 40. If you must send data with an EBCDIC lower than hex 40 to the printer, specify it as hex data.</p> <p>Binary data must be enclosed in single quotes and must be preceded by an uppercase B. Binary data must be at least 8 digits in length, and the total number of binary digits must be a multiple of 8.</p> <p>Hex data must be enclosed in single quotes and must be preceded by an uppercase X. Hex data must be at least two hex digits in length, and the total number of hex digits must be a multiple of 2.</p> <p>If a constant has been equated with a label, the label may be used for the INIT parameter, and the equated constant is placed into the data area.</p> <p>The address of a data area may be initialized into storage by coding:</p> <p style="padding-left: 40px;">INIT=ADDR (label [<math>\pm</math> constant] );</p> <p>where label is the label of the data area.</p> <p>Initialization occurs as illustrated in Figure 3-2.</p> <p><b>Note:</b> Initialization of a decimal register with an equated constant fills the register with binary data.</p> <p>The LENG function cannot be used.</p> |

|            | Decimal Registers  | Binary Registers                                 | Stor, MDUP, and PRMT Areas                       |
|------------|--|--|--|
| Characters | Right adjust, pad with blanks.   | Left adjust, pad with blanks.                    | Left adjust, pad with blanks.                    |
| Hex        | Right adjust, pad with zeros.  | Right adjust, pad with zeros.                    | Right adjust, pad with zeros.                    |
| Binary     | Right adjust, pad with zeros.  | Right adjust, pad with zeros.                    | Right adjust, pad with zeros.                    |
| +Integer   | Right adjust, pad with zeros. Zone on low-order byte is X'F'; leave as EBCDIC. | Right adjust, pad with zeros, convert to binary. | Right adjust, pad with zeros, convert to binary. |
| -Integer   | Right adjust, pad with zeros. Zone on low-order byte is X'D'; leave as EBCDIC. | Error.   | Error.   |

Figure 3-2. Initialization of Data Areas

The following examples illustrate how the assembler allocates data areas as it assembles the .DC control statements.

Figure 3-3 represents an area of storage in a partition. The assembler has assigned the 2 bytes of BR120 and the high-order byte of BR121 to a previously allocated data area. The next sequential byte that is available to the assembler is the low-order byte of BR121. The bytes of BR127 are the last bytes that can be allocated as a binary register.

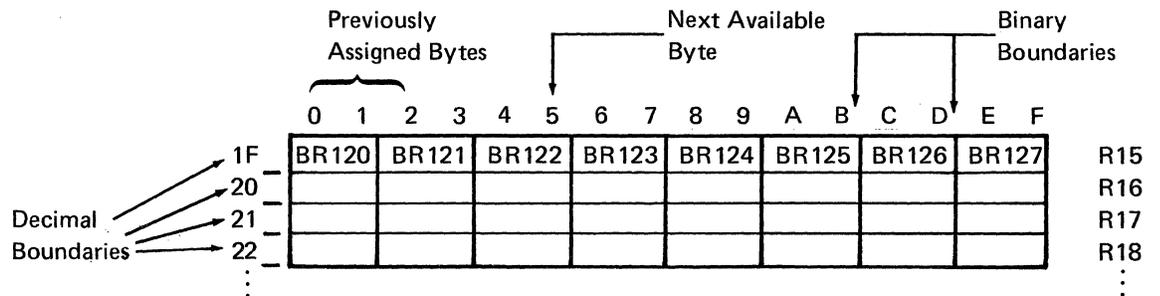


Figure 3-3. Storage Bytes



Figure 3-5 represents an area of overlapped storage in a partition. The assembler has assigned the 2 bytes of BR96, the 2 bytes of BR97, and the high-order byte of BR98 to a previously allocated data area. The next sequential byte that is available to the assembler is the low-order byte of BR98.

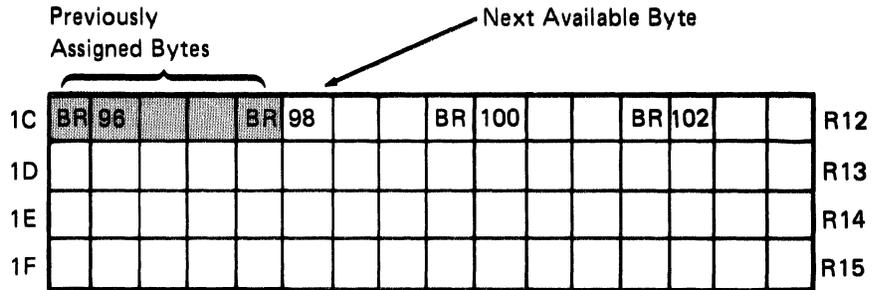
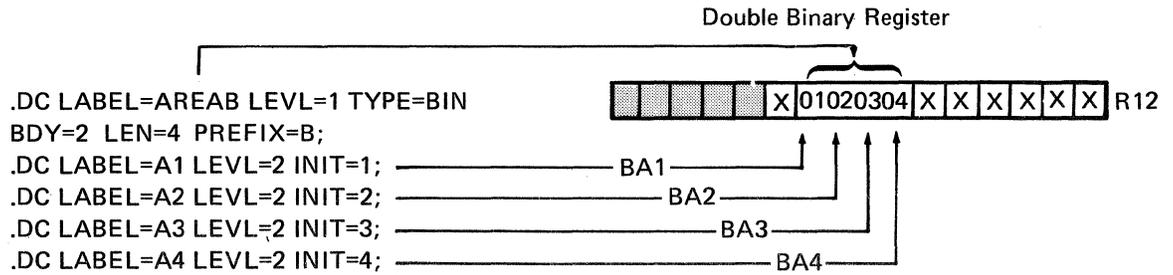


Figure 3-5. Storage Bytes

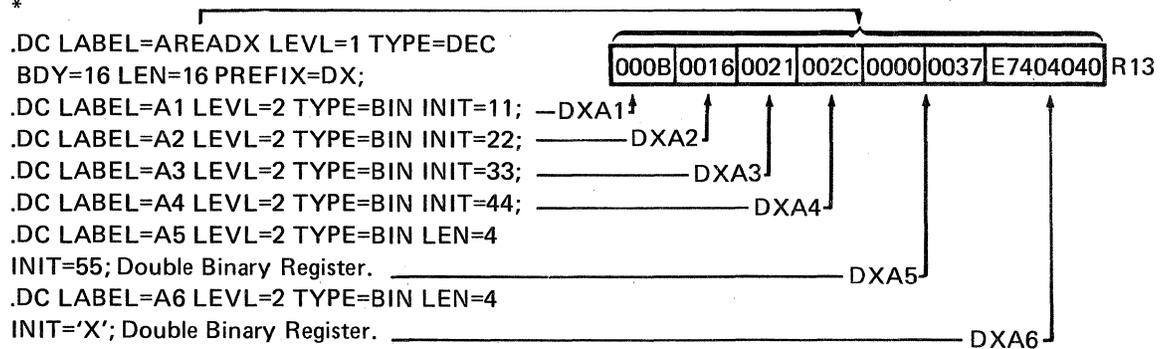
Example 2 shows .DC statements that use the LEVL and PREFIX parameters and illustrates how the assembler allocates the bytes in Figure 3-5.

**Example 2: Storage Structures; LEVL, PREFIX**

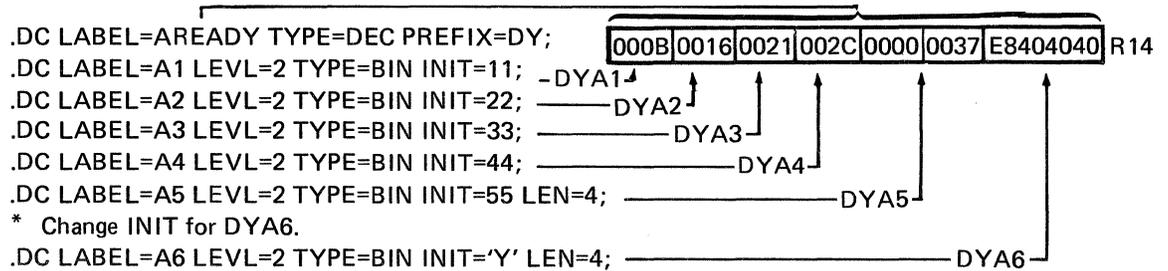


\* DISP may be omitted for contiguous fields.  
 \* The level 1 area is referred to as AREAB;  
 \* the level 2 areas as BA1, BA2, BA3, BA4.  
 \*  
 \*

\* The following shows how to use PREFIX  
 \* to define two decimal registers and  
 \* overlay them with eight binary registers  
 \* each. The coding for the AREADX registers  
 \* can be copied for the AREADY registers.  
 \*



\* The level 1 decimal register is referred to as AREADX,  
 \* the level 2 binary registers as DXA1, DXA2, and so on.



\* Change INIT for DY A6.

### **.DCLBR Control Statement**

```
.DCLBR LABEL= [,...] ;
```

The .DCLBR control statement declares and labels one or more binary registers with a single statement. It may be used only for declaring registers which are not initialized. The registers will contain hex 00s. You do not specify a register number. The system assigns the labels to the next available binary registers. You can declare up to a maximum of 30 binary registers with each .DCLBR control statement.

#### **Parameter**

| <b>Name</b> | <b>Description</b>  |
|-------------|---|
| LABEL       | Label; lists the names of the binary registers, each name separated by a comma. |

#### *Example:*

```
.DCLBR LABEL=PATRN,REGB1,REGB2,CHK4;
```

### **.DCLDR Control Statement**

```
.DCLDR LABEL= [,...] ;
```

The .DCLDR statement declares and labels one or more decimal registers with a single statement. It may be used only for declaring registers which are not initialized. The decimal registers will contain hex 00s. You do not specify a register number. The system assigns the labels to the next available decimal registers. You can declare up to a maximum of 30 decimal registers with each .DCLDR control statement.

#### **Parameter**

| <b>Name</b> | <b>Description</b>   |
|-------------|--|
| LABEL       | Label; lists the names of the decimal registers; each name separated by a comma. |

#### *Example:*

```
.DCLDR LABEL=PAYMT,BAL1,RATE,EXCH,TOTAL,SEND;
```

### **.DCLIND Control Statement**

```
.DCLIND LABEL= [,...] ;
```

The DCLIND statement is used to declare and label one or more user indicators (10-199) with a single statement. You do not specify indicator numbers. The system assigns the labels to available user indicators. You can declare up to a maximum of 30 indicators with each .DCLIND control statement.

| Parameter Name | Description  |
|----------------|--|
| LABEL          | Label; lists the names you assign to the indicators, each name separated by a comma. |

*Example:*

```
.DCLIND LABEL=LABL1,LABL2,LABL3,LABL4,LABL6;
```

**.EQUATE Control Statement**

```
.EQUATE [REG=   NUMB=   IND=
        LABEL=  EXPR=] ;
```

This control statement equates labels to registers, constants, indicators (10-199), or the value of an expression. You must specify the register, constant, or indicator number along with the label you wish to use. You may equate up to 30 register, constant, or indicator labels with each .EQUATE control statement, or one arithmetic expression. This statement is useful to label system registers and indicators. It is the only means to specify an arithmetic expression in the control statements.

A labeled expression may be used in subsequent instructions as a constant or a storage specification. If it is used as a storage label, the result of the expression must be a valid storage address or an error results. Otherwise, it may be used in any instruction that requires immediate data, such as a length or displacement specification.

**Note:** Only reserved registers, previously equated terms, and self-defining terms may be used in the equate statement. EQUATE does not check to see if the specified register already has a label. The current EQUATE label does not override any previously assigned labels.

| Parameter Name | Description  |
|----------------|--|
| REG            | Registers; ordered pairs of a register (binary or decimal) followed by a label. Separate the register and label with a comma, enclose each pair in parentheses, and separate the pairs with a comma.   |
| NUMB           | Numbers; ordered pairs of a constant and a label. Separate the constant and label with a comma, enclose each pair in parentheses, and separate the pairs with a comma. If used as the length parameter of a .DC TYPE = PRMT or .DC TYPE = MDUP, the .DC statement must follow the .EQUATE statement. |
| IND            | Ordered pairs of an indicator and a label. Separate the indicator and label with a comma, enclose each pair in parentheses, and separate pairs with a comma.   |
| LABEL          | Expression label; used only with the EXPR parameter, it specifies the name you want to assign to the result of the expression. You may specify this label as a parameter in a subsequent instruction.  |

**Note:** This label should not be used:

- Prior to its declaration.
- Within a screen format control string.
- With an I/O format control string.
- As the LENGTH parameter of .DC TYPE = PRMT or .DC TYPE = MDUP.

**Parameter  
Name**

**Description**

EXPR

Expression; an expression of up to eight terms, separated by arithmetic symbols. The following symbols may be used:

| Symbol | Meaning  |
|--------|----------|
| +      | Add      |
| -      | Subtract |
| *      | Multiply |
| /      | Divide   |

The arithmetic operations are performed with 2-byte integers; fractions and overflows are ignored. The arithmetic operations are performed from left to right, although the multiply and divide operations are performed before the add and subtract operations. Do *not* include parentheses in the expression.

The terms may be any type of previously defined labels. The length attribute of the first term of the expression will be assigned to the expression label.

*Examples:*

```
.EQUATE REG=(BR5,XREG),(BR6,YREG) NUMB=(22,INDEX) IND=(17,SWITCH);
```

```
.EQUATE REG=(R4,BUF7),(R5,BUF8),(R8,BUF9);
```

```
.EQUATE LABEL=AREA2 EXPR=BUF7+INDEX/3-4;
```

Expression operands can be relocatable terms or self-defining terms. Relocatable terms are labels used for addressing storage locations relative to the beginning of the partition. These include labels of instructions and data areas. Relocatable terms can only be used as operands of the add and subtract operators. Self-defining terms are absolute values or labels that have been equated to such values. Self-defining terms can be used as operands of the add, subtract, multiply, and divide operators.

There are only two valid forms of expressions. Self-defining terms have no effect on the validity of an expression. The validity of an expression can thus be determined by comparing the number of add operators immediately followed by a relocatable term (quantity A) with the number of subtract operators immediately followed by a relocatable term (quantity S). The first term in an expression is assumed to immediately follow an implied add operator. If A is equal to S then the expression is valid and defines a self-defining term; the pairs of relocatable terms of opposite operator type producing a result of type self-defining term. If A exceeds S by exactly one then the expression is valid and defines a relocatable term because of the single unpaired relocatable term immediately preceded by an add operator. All other comparative values of A and S define invalid expression forms.

*Examples:*

Where R represents a relocatable term and Q represents a self-defining term.

.EQUATE LABEL=EXPR1 EXPR=Q+Q+R-Q\*Q/Q-R;  
A=1 S=1  
Therefore EXPR1 is a valid expression of type self-defining term.

.EQUATE LABEL=EXPR2 EXPR=R+Q+R+R-R+Q/Q-R;  
A=3 S=2  
Therefore EXPR2 is a valid expression of type relocatable term.

.EQUATE LABEL=EXPR3 EXPR=R+Q+R-Q;  
A=2 S=0  
Therefore EXPR3 is an invalid expression.

### SET UP AND INITIALIZE DEVICE CONTROL BLOCKS

One device I/O control block (IOB) must be set up and initialized for each data set your program uses. Use the .COMM statement for a data set that uses the communications line. Use a .DATASET control statement for a data set that uses a printer or diskette.

#### .COMM Control Statement

```
.COMM [CAM=] DSN= {ELAB=} TYPE=  
[RECL= BSIZ= ETAB=} LABEL=  
HTAB= VTAB=] ; LBUF=
```

*Parameters unique to BSC:*

```
[SIDL= SIDH= RECFM=]
```

*Parameter unique to SNA:*

```
PLUNAME=
```

The .COMM statement specifies the characteristics of the communications line for the current communications session. It also sets up the device I/O control block in the current partition. The *IBM 5280 Communications Utilities Reference Manual, SC34-0247*, describes the functions of the SNA and the BSC versions of the communications access method, how to determine which version to use, and how to load the access method.

Although most .COMM parameters apply to both the BSC and SNA versions of the communication access method, some parameters are unique to one of the versions.

| <b>Parameter Name</b> | <b>Description</b>   |
|-----------------------|--|
| CAM                   | Communications access method; defines the CAM as either BSC or SNA. If omitted, it defaults to BSC.  |
| DSN                   | Data-set number; a decimal number from 1 to 15 that identifies the communications device I/O control block. This is a mandatory parameter.   |
| ETAB                  | Error table label (either ETAB or ELAB must be specified); the label of the label table set up by a .LABTAB statement. Each entry in the label table is the address of a subroutine that handles one specific external status condition. |
| ELAB                  | Error subroutine label (either ETAB or ELAB must be specified); the label of the subroutine that handles all external status conditions.   |
| TYPE                  | Type; specifies attributes of the data to be processed. One entry, file type, is mandatory. For BSC, one optional attribute key-word may follow the type. Separate the type and attribute with a comma.                                  |

#### **File Type, First Position**

- SR Sequential read, records can be received only.
- SW Sequential write; records can be transmitted but not received.
- COM General communications; sequential read, sequential write, or sequential read and write. (BSC only)
- CN Conversational; transmit one message, receive n messages.

#### **Optional Attributes, Second Position (for BSC only)**

- CB Compressed blanks, to expand blank-compressed data that is received. (The IBM 5280 does not transmit compressed data.)
- BT Blank Truncation, to truncate trailing blanks in data to be transmitted and to insert trailing blanks in data that is received.

| <b>Parameter Name</b> | <b>Description</b>   |
|-----------------------|--|
| <b>RECL</b>           | Record length; length of the logical records. Although this is an optional parameter in the .COMM statement, the RECL field of the I/O control block must contain a valid RECL value when the TINIT instruction is issued.<br><br>If RECL is not coded on the .COMM statement, you must initialize the record size field of the communications I/O control block as appropriate. |
| <b>BSIZ</b>           | Block size; maximum block size of the data to be transmitted. Include this parameter if you specify FB (fixed length and blocked) or VB (variable length and blocked) for the RECFM (record format) parameter. It defaults to 256.   |
| <b>LBUF</b>           | Logical buffer; the label assigned to the buffer by a .DC control statement.   |
| <b>LABEL</b>          | Label; a name to identify the I/O control block for communications. If omitted, the I/O control block is assumed to be not labeled.  |
| <b>HTAB</b>           | Horizontal tab table; the label of a horizontal tab table that specifies the printer tab settings.   |
| <b>VTAB</b>           | Vertical tab table; the label of a vertical tab table that specifies printer tab settings.   |

**Parameters that are unique to BSC**

|             |   |
|-------------|---|
| <b>SIDL</b> | Security ID local; character string to be sent to the host from a local terminal on switched lines. If specified, this ID will override the value in the communications control block, which is specified during execution of the communications configuration utility. If neither security ID is specified, no local ID will be sent to this host. |
| <b>SIDH</b> | Security ID host; a character string ID sent to local terminals from the host on switched lines. If specified, this ID will override the value in the communications control block, which is specified during execution of the communications configuration utility. If neither security ID is specified, no ID will be checked.                    |

**Parameter Name**

**Description**

RECFM Record format; describes the record to be processed, using one of the following keywords. If omitted, it defaults to fixed length (F).

| Keyword | Meaning                     |
|---------|-----------------------------|
| F       | Fixed length                |
| V       | Variable length             |
| FB      | Fixed length and blocked    |
| VB      | Variable length and blocked |

**Parameters that are unique to SNA/SDLC**

PLUNAME Primary logical unit name; a character string enclosed in single quotes. This is an optional parameter. If specified, it will be checked by the communications access method during initialization.

*Examples:*

.COMM CAM=BSC DSN=2 TYPE=COM,BT LABEL=COMIOB ETAB=TABL01;

.COMM CAM=SNA DSN=3 TYPE=SW, LABEL=SNAIOB ELAB=ERRTN  
PLUNAME=LOG01;

**.DATASET Control Statement**

```
.DATASET NAME= DSN= TYPE= { ETAB= }
          { DEVID= }
          { DEV= } PB1= [PB2= RECL=
          BSIZ= LBUF= LABEL= TRANS= DFLG=] ;
```

*Parameters unique to SCS conversion data sets:*

PGSIZ= LINSZ= LSTLN= [SGEA=]

*Parameters unique to keyed data sets:*

KPOS= KLEN= TLOC= DLTA=

The .DATASET control statement specifies the characteristics of a data set to be referred to by the program. It generates the device I/O control block which describes the characteristics of the data set to the I/O device.

| Parameter Name | Description   |
|----------------|---|
| NAME           | Data set name; mandatory for diskette, optional for printer. Enter the label, which was assigned by a .DC control statement, of the area where the data set name is stored. The data set name is described in Chapter 4 under <i>Partition Load and Exit Instructions</i> . Do not enter the data set name here. For label update data sets (type = 1), the data set name must be blank, and the period that follows the volume identifier is optional. |
| DSN            | Data set number; a decimal number from 1 to 15 which identifies the data set being described. This is a mandatory parameter.  |
| ETAB           | Error table label (either ETAB or ELAB must be specified); the label of the table set up by a .LABTAB statement. Each entry in the label table is the address of a subroutine that handles one specific external status condition.  |
| ELAB           | Error subroutine label (either ETAB or ELAB must be specified); the label of the subroutine that handles all external status conditions.  |
| DEV            | Device address (either DEV or DEVID must be specified); the physical address of the device to which this data set information is directed. A physical address is expressed as four hexadecimal digits.  |
| DEVID          | Device identifier (either DEV or DEVID must be specified); the 2-character logical device identifier that identifies the logical device to which this data set information is directed. (See <i>Logical Device Identifiers</i> in Chapter 1.)   |
| TYPE           | Type; specifies the data set type, and may specify other attributes. The first positional entry of this parameter (type) is mandatory. It may be followed by as many optional second position entries as needed. See <i>Diskette Data Management</i> in Chapter 2 for more information about the data set types and attributes.   |

If an optional attribute is included after the type specification, separate the type and the optional attribute with a comma.

#### Data Set Type, Mandatory, First Position

- |    |   |
|----|---|
| SR | Sequential read; records can be read sequentially, or directly by relative record number. Records cannot be written.  |
| SW | Sequential write; records can be written but not read.<br><br>The records are accessed starting at the next available record space, which is determined by the EOD and offset to the next record values on the HDR1 label on the diskettes.<br><br>This is the only valid type for printer data sets. |

| <b>Parameter Name</b> | <b>Description</b>  |
|-----------------------|---|
| TYPE<br>(continued)   | <b>SU</b> Sequential update; records can be read and written sequentially, or directly by relative record number. Records can be added at EOD.  |
|                       | <b>KR</b> Key indexed read; key indexed records can be read only, sequentially or directly by key. An index table is automatically built on OPEN.                                     |
|                       | <b>KRN</b> Key indexed read, no table build, records can be read only, sequentially or directly by key, and you must build or supply your own index table.                            |
|                       | <b>KU</b> Key indexed update; key indexed records can be read sequentially or directly by key, and written or added sequentially. An index table is automatically built on OPEN.      |
|                       | <b>KUN</b> Key indexed update, no table build, records can be read sequentially or directly by key, and written or added sequentially. You must build or supply your own index table. |
|                       | Label update; data set labels can be read and updated. The diskette index cylinder is accessed at track 0, sector 1. Otherwise, the operation is as for sequential update (SU).       |
| <b>INI</b>            | Initialize diskette.  |

**Optional Attributes, Second Position**

|             |  |
|-------------|--|
| <b>SHR</b>  | Shared read; shares the data set with other jobs that specify shared read.   |
| <b>SHW</b>  | Shared write; for diskette, shares the data set with other jobs that specify shared write. For printer, shares the same printer with other data sets that specify shared write.  |
| <b>SHRW</b> | Shared read/write; shares the data set with any other jobs that specify shared attributes. The data set can be read and written to if the other job specifies SHRW; the data set is write-only if the other job specifies SHW, or is read-only if the other job specifies SHR. |

| <b>Parameter Name</b> | <b>Description</b>   |
|-----------------------|--|
| TYPE<br>(continued)   | <p><b>EW</b> Early write; for diskette, this option allows logical records to be updated; then the entire physical buffer is written to the diskette immediately, without waiting until a full physical buffer has been updated. The entire physical buffer is freed upon completion of the write operation. This releases the logical records in the buffer so they can be used by other jobs.</p> <p>For printer, the data in one logical record is sent to the printer and printed immediately, without blocking.</p> |
|                       | <p><b>QR</b> Quick release; like EW, but affects only read operations. The entire physical buffer is freed upon completion of the read operation. This releases the logical records in the buffer to be used by other jobs. Do not use QR when writing with pointer I/O.</p>   |
|                       | <p><b>SCS</b> Standard character string; indicates SCS conversion.</p>   |
|                       | <p><b>ERS</b> Erase; existing data is erased at open. You may not specify ERS for a shared file. You may specify ERS for a label update data set to erase the label area.</p>  |
|                       | <p><b>PTR</b> Pointer I/O; indicates pointer I/O is being done. (See <i>Pointer I/O</i> in Chapter 2.)</p>   |
|                       | <p><b>EXTC</b> Overlapped extent check; during an open or allocate operation, checks all extents of all other data sets to ensure that they do not overlap this data set.</p>  |
|                       | <p><b>TLBL</b> Translate label; translation applies to diskette HDR1 label as well as to the data.</p>   |
|                       | <p><b>ORD</b> Ordered; may be specified only with a keyed data set to indicate that key indexed records are in ascending key sequence.</p>   |

| <b>Parameter Name</b> | <b>Description</b> |
|-----------------------|--------------------|
|-----------------------|--------------------|

**Note:** If you specify a keyed data set and include attributes for another file type, the system assumes a keyed data set.

|            |  |
|------------|--|
| <b>PB1</b> | Physical buffer 1; the label assigned to the buffer by a .DC control statement. This parameter is mandatory. |
|------------|--|

**Note:** Physical buffer lengths must begin on a 128-byte boundary, and must be a multiple of 128, regardless of block size or record format. This is because buffer size is specified in the I/O control block as a multiple of 128. For data sets requesting conversion to SCS (standard character string) data sets, the maximum size is 256 bytes.

|            |  |
|------------|--|
| <b>PB2</b> | Physical buffer 2; the label assigned by the .DC statement that set up the buffer. This buffer is used for double buffering. This parameter is optional. |
|------------|--|

|             |   |
|-------------|---|
| <b>RECL</b> | Record length; the length of the logical record to be handled. This parameter is required for printer output and for SCS conversion data sets; it is optional for diskette I/O. If this parameter is omitted for diskette, the value from the data set label is used. If this parameter is specified for diskette, it must match the value in the HDR1 label. |
|-------------|---|

| Parameter Name | Description   |
|----------------|---|
| BSIZ           | Block size; the length of the block to be handled. This parameter is required for SCS conversion data sets; it is optional for printer and diskette I/O. If this parameter is omitted for diskette, the value from the data set label is used. If this parameter is specified for diskette, it must match the value in the HDR1 label.  |
| LBUF           | Logical buffer; the label assigned to the buffer by a .DC control statement. This parameter should be omitted when using pointer I/O (locate mode). If omitted, the logical buffer address is assumed to be the same as the physical buffer (PB1) address, and the pointer I/O flag in the data set IOB is set on.  |
| LABEL          | Label; the name you wish to assign to the device I/O control block for this data set. This is an optional parameter. If this parameter is omitted, the I/O control block is assumed to be not labeled.  |
| TRANS          | Translate tables label; the label specified in a .TABLE control statement that describes a pair of 256-byte tables used for code translation (substitution). The first table is used for input translation, and the second for output. The content of each table is identical to tables used with the equivalent instruction TRANS. This optional parameter is used only when translation is desired.   |
| DFLG           | Delete flag; the character that is placed in the HDR1 label during an allocate, which will be used to indicate a deleted record. This character is placed in the last byte of a record that is operated upon by a WRTS (write delete) instruction. A deleted record is skipped on a READ (sequential read) and overwritten on a WRTI (write insert) or WRT (write current) instruction. During an OPEN, if the delete character on the HDR1 label is not blank or X '00', then the delete character in the IOB is overridden. |

This parameter may be used only for an I or E exchange data set.

| <b>Parameter Name</b> | <b>Description</b> |
|-----------------------|--------------------|
|-----------------------|--------------------|

These parameters are only for standard character string conversion of printer output (when the second TYPE entry is SCS). These parameters are not allowed for keyed data sets.

|              |   |
|--------------|---|
| <b>PGSIZ</b> | Page size; the number of lines per page. This parameter is mandatory with SCS conversion.   |
| <b>LINSZ</b> | Line width; number of characters per line. This parameter is mandatory with SCS conversion.   |
| <b>LSTLN</b> | Last line; line number of the last line to print. This parameter is mandatory with SCS conversion.  |
| <b>SGEA</b>  | Set graphics error action; the symbol to represent unprintable values, in the form (character, code). The default is (-,1), which prints one dash and continues printing. The only other valid entry is (-,3), which prints one dash and stops printing at the end of the line. |

The following parameters are only for keyed files.

|             |  |
|-------------|--|
| <b>KPOS</b> | Key position; the position of the key in the record (first column = 1). This parameter is required for keyed data sets.  |
| <b>KLEN</b> | Key length; the number of positions in the key. This parameter is mandatory for keyed data sets.   |
| <b>TLOC</b> | Table location; the label of the table you have set up for the file index parameters. Use the label assigned by the .TABLE statement. This parameter is mandatory for keyed data sets, whether you build your own index table or let the IBM 5280 automatically build the table. |
| <b>DLTA</b> | Delta of index table; the number of logical records between each index entry. This parameter is required for KRN and KUN data sets. If this parameter is omitted, the delta is calculated from data set length and table length.   |

*Examples:*

```
.DATASET NAME=TXDATA DSN=3 RECL=80 BSIZ=3120 LBUF=MYAREA  
PB1=WKBUF TYPE=KR,SHR ELAB=ERORTN2 DEVID=D1  
KPOS=12 KLEN=8 TLOC=KEYTBL;
```

## SET UP AND LABEL TABLES

The table control statements organize and assign labels to tables, but do not initialize storage. The .LABTAB control statement organizes the labels of subroutines, which are used in indexed subroutine calls. The .TABLE statement organizes data areas initialized by .DC control statements. These data areas are used by the TABLE instructions. The .SYSTAB statement assigns labels and determines locations of the system tables, which are set up and used by the IBM 5280.

### .TABLE Control Statement

```
.TABLE LABEL= DCLBL= ARGL= { MAXM=
                          ENTRIES= } [BYPASS]
                          { MAXM= ENTRIES= }
```

A table consists of a group of contiguous fields of the same length. The content of each field is the table argument, and the position of the field within the table is the index of the field. The index of the first table field is one.

Table arguments may be in an ordered or unordered sequence. An ordered table has arguments arranged in ascending or descending order according to the standard EBCDIC collating sequence.

The .TABLE control statements build a system table that will be referred to each time a table instruction is encountered during program execution. The system table contains all the parameters of each table you use in your program. You must include one .TABLE statement for each of your tables. These .TABLE statements must be consecutive. The address of the system table is stored in the partition I/O control block. When a table instruction refers to the label of a table, the address and all the parameters of that table are provided by the system table.

Space for the tables you use in your program must be allocated by .DC statements.

| Parameter Name | Description   |
|----------------|---|
| LABEL          | Label; the label of the table this statement defines.   |
| DCLBL          | DC label; the LABEL parameter from the declare .DC statement that assigned space for this table.  |
| ARGL           | Argument length; the number of bytes in the table argument.   |
| MAXM           | Maximum; the maximum number of entries allowed in this table. Either the MAXM or ENTRIES parameter must be included for fixed-length tables. For variable-length tables both MAXM and ENTRIES are used. |

| Parameter Name | Description   |
|----------------|---|
| ENTRIES        | Entries; the number of entries in the table that have been initialized. In variable length tables this may not be the same as the maximum number of entries. If MAXM is not included, the maximum number of entries is equal to the number specified by ENTRIES, and a fully initialized table is assumed. If MAXM is included and ENTRIES is omitted, the table is a fixed-length table that is not initialized. |
| BYPAS          | Bypass; the number of bytes per table entry that are not part of the table argument. This may be used in conjunction with LEVL and DISP on the .DC statement to define a two-dimensional table.   |

*Examples:* In Figure 3-6, the following .DC statements allocate space for two tables. Then the .TABLE statements define the parameters for the two tables. The parameters are stored in the system table, which the IBM 5280 builds in another area of storage, and refers to during program execution.

```
.DC LABEL=TAB2NUMS LEN=400 LEVL=1 ; start of table of numbers
.DC LABEL=DAT21 LEN=8 LEVL=2 INIT='11111111' ;
.DC LABEL=DAT21END LEN=0 LEVL=2 ;
.DC LABEL=TAB2LETS LEN=400 LEVL=1 LOC=DAT21END ; start of table of letters
.DC LABEL=DAT22 LEN=2 LEVL=2 INIT='AA' ;
.DC LABEL=DAT23 LEN=8 LEVL=2 INIT='22222222' ;
.DC LABEL=DAT24 LEN=2 LEVL=2 INIT='BB' ;
```

```
*TABLE statement defining a table with 8-byte arguments.
.TABLE LABEL=NUMBERS DCLBL=TAB2NUMS ARGL=8 BYPAS=2 ENTRIES=2 MAXM=40;

*TABLE statement defining a table with 2-byte arguments.
.TABLE LABEL=LETTERS DCLBL=TAB2LETS ARGL=2 BYPAS=8 ENTRIES=2 MAXM=40;
```

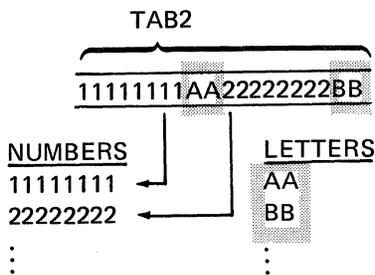


Figure 3-6. Data Tables

## .LABTAB Control Statement

```
.LABTAB LABEL= ENTRY=
```

The .LABTAB control statement specifies a group of subroutine or statement labels and creates a label table of the addresses of these subroutines or statements. This table of addresses is used to make indexed branches through a table as with the GOTAB and CALLTB instructions. You can specify up to a total of 30 ENTRY labels for each .LABTAB control statement.

**Note:** The .LABTAB tables are *not* used with the TABLE instructions that are described in Chapter 4.

### Parameter

| Name  | Description  |
|-------|--|
| LABEL | Label; the name you wish to assign to the table of addresses being created. This parameter is mandatory unless you are building a label table of more than 30 entries. In that case, include the LABEL parameter with the .LABTAB statement that specifies the first 30 entries, and follow it with one or more .LABTAB statements with the LABEL omitted. |
| ENTRY | Label of a subroutine or statement whose address is to be stored in the label table. At least one but no more than 30 labels must be entered. If two or more labels are entered, separate them with a comma. The first entry is at index 0 in the label table, the second entry is at index 1, and so on.  |

### Examples:

```
.LABTAB LABEL=MYTABLE ENTRY=EOF,EOJ,EOT;
```

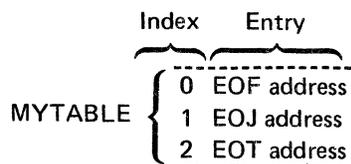


Figure 3-7. Label Table

## **.SYSTAB Control Statement**

**.SYSTAB [PRMT= FMT= MDUP= SFMT= STACK=] ;**

The **.SYSTAB** control statement allows you to assign labels and determine the location of the system control tables and the subroutine stack. When the assembler encounters a **.SYSTAB** statement, the indicated control tables and subroutine stack are assigned locations at the address in the current location counter. You may reserve space for the entries into the area by specifying the number of entries to be placed into the control table or subroutine stack. If you specify the number of entries you may omit the label; the space for the specified number of entries will be reserved in the table, and the table will be assigned the location in the current location counter. If you omit the number of entries, you must specify the label; the assembler will make the appropriate number of entries in the table but will reserve no extra space for additional entries. See Chapter 2 for more information about these control areas, under *System Tables* and *The Partition Subroutine Stack*.

If the **.SYSTAB** statement is not included, any necessary control tables are created and placed into the partition storage when the **.END** statement is encountered, and the address of each table is stored into an assigned location in the partition I/O control block. The control tables are assigned no labels; if you wish to refer to an entry in a control table, you must read the address from the appropriate area in the partition control area.

On the assembly listing, all system tables and the subroutine stack are written immediately following the **.END** statement. This is true regardless of their actual addresses. No system tables or subroutine stack is written on the listing with the **.SYSTAB** control statement.

If the **.SYSTAB** statement is used for a separately assembled subroutine (**OPTION=SUB** in the **.START** statement), you must initialize the control table addresses in the partition control area at execution time. The location of each table address is included in the following parameters.

### **Parameter**

| <b>Name</b> | <b>Description</b>  |
|-------------|---|
| <b>PRMT</b> | <p>Prompt control table label; specifies the name you wish to assign to the prompt control table. You may optionally follow the label with a comma and the number of entries to be placed into the table.</p> <p>The number specified must include one extra entry to be used for the first table entry, which the assembler always creates and fills with zeros; specify the number of your prompts plus one.</p> <p>Each entry in the table is 2 bytes long and, except for the first entry, contains the address of a prompt created by a <b>.DC TYPE=PRMT</b> control statement. You must specify the label, or the number of entries, or both. The address of the prompt table is placed into hex 8D-8E of the partition control area.</p> |

| <b>Parameter Name</b> | <b>Description</b>   |
|-----------------------|--|
| <b>FMT</b>            | <p>Format control table label; specifies the name you wish to assign to the format control table. You may optionally follow the label with a comma and the number of entries to be placed into the table.</p> <p>The number specified must include one extra entry to be used for the last entry, which the assembler creates and fills with hex Fs; specify the number of your edit formats plus one. The assembler always creates this system table and the entry filled with Fs whether or not you write any edit formats.</p> <p>Each table entry is 2 bytes long (3 bytes long for global formats), and contains the address of a format created by a .FMTST control statement and its associated .FMTFLD statements. The address of the table is placed into hex 24-25 of the partition control area. You must specify the label, or the number of entries, or both.</p> |
| <b>MDUP</b>           | <p>Main storage duplication control table; specifies the name you wish to assign to the MDUP control table. You may optionally follow the label with a comma and the number of entries to be placed in the table.</p> <p>Each table entry is 2 bytes long, and contains the address of a MDUP area created by a .DC TYPE=MDUP control statement. The address of the table is placed into hex B6-B7 of the partition control area. You must specify the label, or the number of entries, or both.</p>   |
| <b>SFMT</b>           | <p>Screen format control table label; assigns a name to the SFMT control table. You may optionally follow the label with a comma and the number of entries to be placed in the table.</p> <p>Each table entry is 2 bytes long and contains the address of a screen format. Each screen format is created by a .SFMTST statement and its associated .SFMT statements up to and including the .SFMTEND statement. The address of the table is placed into hex F9-FA of the partition control area. You must specify the label, or the number of entries, or both.</p>  |
| <b>STACK</b>          | <p>Stack label; assigns a name to the subroutine stack. For the STACK parameter, you must specify the number of entries to be placed in the table whether or not you specify the label. The number required for the partition stack entries is the number of levels your program uses for nested subroutine calls. Include calls to common function routines, calls to external status routines, and a call to the program check routine. It is your responsibility to control overflow of the partition stack; the system does not check for such overflow to prevent the stack from extending beyond the end of the partition.</p>   |

| Parameter Name       | Description   |
|----------------------|---|
| STACK<br>(continued) | Each stack entry is 2 bytes long and contains the return address of the most recent subroutine call. The address of the next available space in the subroutine stack is placed into BR18. |

**Note:** When the number of table entries is included in a .SYSTAB parameter, it overrides the actual number of entries generated from your control statements. Therefore, you may reserve extra space for future entries by specifying a number greater than the present number of entries.

*Examples:*

If the following two statements are the only .SYSTAB control statements in the program, the edit format system table is automatically built when the .END statement is processed.

```
.SYSTAB PRMT=PTABLE MDUP=MTABLE,4 STACK=SUBSTACK,20;
.SYSTAB SFMT = ,22;
```

### SET UP EDIT FORMATS

Certain I/O instructions and data movement instructions may include an edit format. Edit formats describe punctuation, data types, and other editing requirements for the individual fields of a record. The instructions that may include an edit format are:

- READ
- WRT
- WFMCRT
- REBF
- WRTI
- WRBF

For the READ or REBF instruction, the edit format specifies the data area to which each input field is moved. For the READ instruction, the fields are moved from the I/O buffer after the read operation occurs. For the REBF instruction, the fields are moved from the storage area specified in the instruction.

Data directed formatting may be used with the READ or REBF instruction.

As the field moves to the data area, specified punctuation and edit characters are removed or specified conversion occurs.

For the write instructions, the edit format specifies the data area from which each field is moved. For the WRT and WRTI instructions, the fields are moved to the I/O buffer before the write operation occurs. For the WRBF instruction, the fields are moved to the storage area specified in the instruction. For the WFMCR instruction, the fields are moved to the screen, at the screen position specified in the instruction.

As the field moves from the data area, specified punctuation and edit characters are inserted into the field or specified conversion occurs.

In your source program, each edit format must begin with a .FMTST statement and end with a .FMTEND statement. Any number of .FMTFLD statements necessary may be placed between the .FMTST and .FMTEND statements. The parameters for the .FMTFLD and .FMTEND statements are identical.

### **.FMTST Control Statement**

```
.FMTST LABEL= [CCHAR= COL= ] ;
```

The .FMTST control statement identifies the start of an edit format specification. This statement must be followed by one or more .FMTFLD statements and a .FMTEND statement, or by a .FMTEND statement alone.

| <b>Parameter Name</b> | <b>Description</b>  |
|-----------------------|---|
| LABEL                 | Format label; the name you want to assign to this format description. This label is used as the format parameter in instructions that allow edit specifications format. This is a mandatory parameter.            |
| CCHAR                 | Condition character; the character used in data directed formatting. Any single character is acceptable. This is an optional parameter used only for data directed formatting.                                    |
| COL                   | Character column; the column in which the condition character is located. Valid entries range from 1 to the maximum size of the I/O buffer. This is an optional parameter used only for data directed formatting. |

### **.FMTFLD Control Statement**

```
.FMTFLD DCLBL= [LEN= TYPE= COL= { EDIT= } PIC= ] ;
```

Each .FMTFLD control statement defines a data field.

| Parameter Name                      | Description   |
|-------------------------------------|---|
| (Read) (Write)<br>(to) DCLBL (from) | Declared label; the label from the control statement that declared the register or labeled storage area into which (for READ or REBF instructions), or from which (for WRT, WRTI, WRBF, or WFMCRN instructions) the field is moved. The label may be followed by an optional comma and a length specification of 1 to 256 bytes. If the length of the storage area is omitted, it defaults to the length specified in the declare statement. The label is mandatory.  |
| (to) TYPE (from)                    | Type; the type of the declared area specified by DCLBL. Valid entries are DEC for decimal, and BIN for binary; defaults to BIN.   |
| (from) LEN (to)                     | Length; the number of bytes (1-256) to access for this field. This is an optional parameter that defaults to 1.   |
| (from) COL (to)                     | Column; the position of the first byte of this field, relative to the leftmost byte of the I/O buffer. The valid range begins at 1, which indicates the first byte of the buffer. This is an optional parameter that defaults to the next available column to the right.  |
| EDIT                                | <p>Edit specification; indicates how to edit the contents of the field during the move. This is an optional parameter that may be included if PIC is omitted. You may specify HX, C, W, or an edit string.</p> <p><b>HX</b> Converts EBCDIC to binary or binary to EBCDIC. For example, B'1001' converts to or from hex F9, and B'1010' to or from hex C1. Do not use HX for a storage area declared with TYPE = DEC.</p> <p><b>C</b> Specifies date edit. Slashes are inserted between each two positions as illustrated:</p> <p style="text-align: center;">xx/xx/xx</p> <p>The length of the DCLBL parameter should be 6.</p> <p><b>W</b> Specifies alternate date edit. Periods are inserted between each two positions as illustrated:</p> <p style="text-align: center;">xx.xx.xx</p> <p>The length of the DCLBL parameter should be 6.</p> <p>Edit string characters may be specified only when the field is being moved to (for READ and REBF) or from (for write instructions) a decimal area, or a binary area with the CV parameter specified.</p> <p>You may specify more than one of the following, separated by commas.</p> |

| <b>Parameter Name</b>      | <b>Description</b> |   |
|----------------------------|--------------------|---|
| <b>EDIT</b><br>(continued) | <b>Character</b>   | <b>Meaning</b>  |
|                            | CV                 | Converts decimal to binary or binary to decimal. For example, B'1001' converts to or from 9 and B'1010' to or from 10. If you convert to binary, you cannot specify any other edit string characters. Not more than 2 bytes of binary data can be converted to decimal.   |
|                            | FX                 | Inserts the currency sign in the field, according to the .EDITC parameter EDCUR. It is a fixed sign and appears on the left side of the field. It is mutually exclusive with FL. It uses two character positions; the default is a blank to the left of a dollar sign (\$).   |
|                            | ZS                 | Indicates zero suppress. When coded, all numerics and any accompanying punctuation will be suppressed for a data value of zero. Currency sign (FX or FL) and sign edit chars (NS, S, CR or DB), if specified, will not be suppressed. When ZS is not coded, the data value zero will appear as a single zero to the left of the decimal point and as many zeros as specified to the right of the decimal point if decimal point punctuation is specified. |
|                            | CP                 | Indicates comma punctuation as specified by the .EDITC parameter EDCOM. It defaults to a comma.   |
|                            | DP                 | Indicates decimal point punctuation as specified by the .EDITC parameter EDDEC. It defaults to a period.  |
|                            | n                  | Specifies the number of digits that appear to the right of the decimal point. Valid range is 1-15.  |

One of the following edit string characters may be specified. Default is blank fill (BF).

| <b>Character</b> | <b>Meaning</b>   |
|------------------|--|
| BF               | Indicates blank fill.  |
| AF               | Indicates asterisk (*) fill.   |
| ZF               | Indicates zero fill.   |
| FL               | Indicates a floating currency sign. It is mutually exclusive with FX and uses two character positions. It defaults to a blank to the left of a dollar sign (\$). |

**Parameter  
Name**

**Description**

EDIT  
(continued)

One of the following edit string characters may be specified. The term *zone* refers to the zone portion of the rightmost byte of the decimal register. The term *field* refers to the field that is moved to or from the decimal register. Default is decimal sign (DS).

**Character      Meaning**

- DS              Indicates decimal sign. The zone is hex F for positive data, and hex D for negative data.
- SS              Indicates stripped sign. The zone is always replaced with hex F.
- NS              Indicates negative sign. If the data is negative, the zone has a hex D and the rightmost position of the field has a minus sign (-). If the data is positive, the zone has a hex F and the rightmost position of the field is blank.
- S                Indicates signs. Same as NS except the rightmost position of the field has a plus sign (+) for positive data.
- CR              Indicates credit. If the data is negative, the zone is hex D and the rightmost two positions of the field have C'CR'. If the data is positive, the zone has hex F and the rightmost two positions of the field are blank.
- DB              Indicates debit. If the data is negative, the zone is hex D and the rightmost two positions of the field have C'DB'. If the data is positive, the zone is hex F and the rightmost two positions of the field are blank.

PIC

Picture definition; a string of symbols that defines the format of the field. This is an optional parameter that may be specified *only* for a WFMCR, WRT, or WRBF instruction that moves the field to a decimal buffer; it is mutually exclusive with the EDIT parameter. The number of digits represented by the PIC string (not including punctuation) must exactly equal the number of digits in the input field in the decimal buffer. Specify one or more of the following, enclosed in single quotes.

**Symbol              Represents**

- 9                A decimal digit. A decimal digit is accepted for output to the corresponding position of the buffer.
- Z                Suppress leading zeros. The corresponding position in the buffer is blanked if the character is a leading zero.

| Parameter Name  | Description |  |
|-----------------|-------------|--|
| PIC (continued) | Symbol      | Represents   |
|                 | V           | Stop zero suppress. Zero suppression is stopped in the corresponding position of the buffer. V must be followed by a comma, slash, or period.<br><br><b>Note:</b> The picture definition must not end with V.  |
|                 | Y           | Insert a blank if the value is 0. A blank is inserted into the corresponding position of the buffer if the position contains a zero.   |
|                 | *           | Insert an asterisk. An asterisk is inserted into the corresponding position of the buffer if the position contains a leading zero.   |
|                 | ,           | Insert a comma. A comma is inserted into the corresponding position of the buffer unless zero suppression has occurred.  |
|                 | /           | Insert a slash. A slash is inserted into the corresponding position of the buffer unless zero suppression has occurred.  |
|                 | .           | Insert a period. A period is inserted into the corresponding position of the buffer unless zero suppression has occurred.  |
|                 | B           | Insert a blank. A blank is inserted into the corresponding position of the buffer.   |
|                 | M           | Insert a currency symbol. The currency symbol can be at a fixed position or placed to the left of the most significant digit. To insert a currency symbol at a fixed position, place one M at the desired position. To insert a floating currency symbol, place an M in all leading digit positions of the associated field. |
|                 | -           | Insert a minus sign. A minus sign is inserted into the corresponding position of the buffer if the field is negative.  |
|                 | +           | Insert a plus sign. A plus sign is inserted into the corresponding position of the buffer if the field is positive.  |

| Parameter Name     | Description   |   |            |   |   |   |   |   |   |
|--------------------|---|---|------------|---|---|---|---|---|---|
| PIC<br>(continued) | <table border="1"> <thead> <tr> <th>Symbol</th> <th>Represents</th> </tr> </thead> <tbody> <tr> <td>S</td> <td>Insert the appropriate sign. The appropriate sign (+ or -) is inserted into the corresponding position of the buffer.</td> </tr> <tr> <td>C</td> <td>Insert CR. The characters CR are inserted into the corresponding positions of the buffer if the field is negative; otherwise the corresponding positions are blank.</td> </tr> <tr> <td>D</td> <td>Insert DB. The characters DB are inserted into the corresponding positions of the buffer if the field is negative; otherwise the positions are blank.</td> </tr> </tbody> </table> | Symbol  | Represents | S | Insert the appropriate sign. The appropriate sign (+ or -) is inserted into the corresponding position of the buffer. | C | Insert CR. The characters CR are inserted into the corresponding positions of the buffer if the field is negative; otherwise the corresponding positions are blank. | D | Insert DB. The characters DB are inserted into the corresponding positions of the buffer if the field is negative; otherwise the positions are blank. |
|                    | Symbol  | Represents  |            |   |   |   |   |   |   |
|                    | S   | Insert the appropriate sign. The appropriate sign (+ or -) is inserted into the corresponding position of the buffer. |            |   |   |   |   |   |   |
| C                  | Insert CR. The characters CR are inserted into the corresponding positions of the buffer if the field is negative; otherwise the corresponding positions are blank.   |   |            |   |   |   |   |   |   |
| D                  | Insert DB. The characters DB are inserted into the corresponding positions of the buffer if the field is negative; otherwise the positions are blank.   |   |            |   |   |   |   |   |   |

*Example:*

To read the following data from data set 4 and move it from the I/O buffer to three decimal registers labeled NAME, SS#, and RATE, the following statements may be used:

```
3White Elmer J. 404772310 3.36
```

```
.FMTST LABEL=F2;
.FMTFLD DCLBL=NAME,16 LEN=16 COL=2; skip condition
character position
.FMTFLD DCLBL=SS# TYPE=DEC LEN=9 COL=20;
.FMTEND DCLBL=RATE LEN=6 COL=31 EDIT=DP, ZF;
READ (4,F2);
```

After the execution of the above statements, the decimal registers contain the following data.

```
NAME = WHITE ELMER J
SS# = 404772310
RATE = 000000000000336
```

A write instruction that specifies F2 as the format parameter would move the data from these registers back into the I/O buffer; all punctuation and edit characters would be replaced before the record is written to the I/O device.

The following tables show examples of PIC edit strings.

### Examples of Zero Suppression

| Input Field | Edit Word | Output Field |
|-------------|-----------|--------------|
| 12345       | ZZZ99     | 12345        |
| 00100       | ZZZ99     | bb100        |
| 00000       | ZZZ99     | bbbb00       |
| 00100       | ZZZZZ     | bb100        |
| 00000       | ZZZZZ     | bbbbb        |
| 00100       | *****     | **100        |
| 00000       | *****     | *****        |
| 00100       | YYYYY     | bb1bb        |
| 10203       | 9Y9Y9     | 1b2b3        |

### Examples of Character Insertion

| Input Field | Edit Word    | Output Field |
|-------------|--------------|--------------|
| 1234        | 9,999        | 1,234        |
| 123456      | 9,999.99     | 1,234.56     |
| 1234        | ZZ.ZZ        | 12.34        |
| 1234        | ZZV.99       | 12.34        |
| 0003        | ZZ.ZZ        | bbbbb3       |
| 0003        | ZZV.99       | bb.03        |
| 0000        | ZZ.ZZ        | bbbbb        |
| 0000        | ZZV.99       | bb.00        |
| 123456789   | 9,999,999.99 | 1,234,567.89 |
| 1234567     | ** ,999.99   | 12,345.67    |
| 0012345     | ** ,999.99   | ***123.45    |
| 123456789   | 9.999.999,99 | 1.234.567,89 |
| 123456      | 99/99/99     | 12/34/56     |
| 123456      | 99.9/99.9    | 12.3/45.6    |
| 001234      | ZZ/ZZ/ZZ     | bbbb12/34    |
| 000012      | ZZ/ZZ/ZZ     | bbbbb12      |
| 000000      | ZZ/ZZ/ZZ     | bbbbbbb      |
| 000000      | **/**/**     | *****        |
| 123456      | 99B99B99     | 12b34b56     |
| 123         | 9BB9BB9      | 1bb2bb3      |
| 12          | 9BB/9BB      | 1bb/2bb      |

## **.FMTEND Control Statement**

```
.FMTEND DCLBL= [TYPE=   LEN=  COL=  {EDIT=} ] ;  
                               {PIC=}
```

The .FMTEND control statement indicates the last field of an edit format. The parameters are identical to the .FMTFLD control statement.

## **SET UP SCREEN CONTROL FORMATS**

A screen control format describes a record that is entered via the keyboard/display. One screen control format must be specified with each ENTR instruction. When the IBM 5280 encounters an ENTR instruction during program execution, it directs the keyboard/display to allow the fields of one record to be entered from the keyboard. The screen control format specifies the length of each field and the type of data that may be entered. The IBM 5280 checks the characters entered into each field to make sure it meets the specifications of the screen control format. Valid data for each field is placed into the I/O buffer as it is entered, according to the current mode of entry. (See *Modes of Entry* in Chapter 2.) The screen control format can also specify prompts and display attributes to be displayed as the record is entered. The prompts and display attributes are moved to specified positions on the screen.

During an ENTR operation, the keyboard/display maintains two pointers. The buffer position pointer always contains the position, relative to the first byte of the I/O buffer, of the next available buffer position. The screen position pointer always contains the next available screen position.

In your source program, each screen control format must begin with a .SFMTST statement and end with a .SFMTEND statement. Between these two statements, you may include as many of the following statements as necessary:

| <b>Statement</b> | <b>Purpose</b>  |
|------------------|---|
| .SFMTCTL         | Specify control of screen attributes, data movement, keyboard functions, or format execution. |
| .SFMPMT          | Specify prompts to move to the screen.  |
| .SFMTFLD         | Describe the display attributes, field type, and keyboard functions of an input field.        |
| .SFMCNS          | Specify constant insert data to place in the I/O buffer and to also move to the screen.       |

Certain parameters or parameter keywords are common to more than one .SFMT control statement. These are as follows:

| <b>Parameter</b> | <b>Description</b>   |                |                |           |   |           |   |           |  |
|------------------|--|----------------|----------------|-----------|---|-----------|---|-----------|--|
| <b>CNTL</b>      | <p>Control; one or more keywords that specify control of the screen attributes, keyboard functions or format execution. The CNTL parameter may be specified in any of the .SFMT control statements. One or more of the following keywords may be specified for the CNTL parameter, depending on the particular control statement.</p>  |                |                |           |   |           |   |           |  |
|                  | <table><thead><tr><th><b>Keyword</b></th><th><b>Meaning</b></th></tr></thead><tbody><tr><td><b>RG</b></td><td><p>Return to program; the keyboard/display suspends processing key entry under the current ENTR command and sets on an external status indicator in the KB/CRT I/O control block.</p><p>If the key entry is processing in a forward direction when this parameter specification is encountered, the resulting external status condition is condition 4. The current .SFMT statement, which contains this RG parameter, is processed before key entry is suspended.</p><p>If key entry is processing in a backward direction such as during a backspace when this parameter is encountered, the resulting external status condition is condition 5. The current .SFMT statement, which contains this RG parameter, is not processed before key entry is suspended.</p></td></tr><tr><td><b>DE</b></td><td><p>Dup key status; changes the status that determines whether pressing the Dup key is allowed within a field. The Dup key is initially enabled at the start of each screen format control series. If it is enabled when this statement is encountered, it becomes disabled. If it is disabled when this statement is encountered, it becomes enabled.</p></td></tr><tr><td><b>MC</b></td><td><p>Monocase conversion status; changes the status that determines whether all lowercase alphabetic characters (and any other characters in the extended international character set for which an uppercase-lowercase relationship is defined) are converted to their uppercase equivalents as they are keyed, before they are inserted into the record and displayed upon the screen. The monocase conversion feature is initially disabled at the start of each screen format control series. If it is disabled when this statement is encountered, it becomes enabled. If it is enabled when this statement is encountered, it becomes disabled.</p></td></tr></tbody></table> | <b>Keyword</b> | <b>Meaning</b> | <b>RG</b> | <p>Return to program; the keyboard/display suspends processing key entry under the current ENTR command and sets on an external status indicator in the KB/CRT I/O control block.</p> <p>If the key entry is processing in a forward direction when this parameter specification is encountered, the resulting external status condition is condition 4. The current .SFMT statement, which contains this RG parameter, is processed before key entry is suspended.</p> <p>If key entry is processing in a backward direction such as during a backspace when this parameter is encountered, the resulting external status condition is condition 5. The current .SFMT statement, which contains this RG parameter, is not processed before key entry is suspended.</p> | <b>DE</b> | <p>Dup key status; changes the status that determines whether pressing the Dup key is allowed within a field. The Dup key is initially enabled at the start of each screen format control series. If it is enabled when this statement is encountered, it becomes disabled. If it is disabled when this statement is encountered, it becomes enabled.</p> | <b>MC</b> | <p>Monocase conversion status; changes the status that determines whether all lowercase alphabetic characters (and any other characters in the extended international character set for which an uppercase-lowercase relationship is defined) are converted to their uppercase equivalents as they are keyed, before they are inserted into the record and displayed upon the screen. The monocase conversion feature is initially disabled at the start of each screen format control series. If it is disabled when this statement is encountered, it becomes enabled. If it is enabled when this statement is encountered, it becomes disabled.</p> |
| <b>Keyword</b>   | <b>Meaning</b>   |                |                |           |   |           |   |           |  |
| <b>RG</b>        | <p>Return to program; the keyboard/display suspends processing key entry under the current ENTR command and sets on an external status indicator in the KB/CRT I/O control block.</p> <p>If the key entry is processing in a forward direction when this parameter specification is encountered, the resulting external status condition is condition 4. The current .SFMT statement, which contains this RG parameter, is processed before key entry is suspended.</p> <p>If key entry is processing in a backward direction such as during a backspace when this parameter is encountered, the resulting external status condition is condition 5. The current .SFMT statement, which contains this RG parameter, is not processed before key entry is suspended.</p>  |                |                |           |   |           |   |           |  |
| <b>DE</b>        | <p>Dup key status; changes the status that determines whether pressing the Dup key is allowed within a field. The Dup key is initially enabled at the start of each screen format control series. If it is enabled when this statement is encountered, it becomes disabled. If it is disabled when this statement is encountered, it becomes enabled.</p>  |                |                |           |   |           |   |           |  |
| <b>MC</b>        | <p>Monocase conversion status; changes the status that determines whether all lowercase alphabetic characters (and any other characters in the extended international character set for which an uppercase-lowercase relationship is defined) are converted to their uppercase equivalents as they are keyed, before they are inserted into the record and displayed upon the screen. The monocase conversion feature is initially disabled at the start of each screen format control series. If it is disabled when this statement is encountered, it becomes enabled. If it is enabled when this statement is encountered, it becomes disabled.</p>   |                |                |           |   |           |   |           |  |

**Parameter  
Name**

**Description**

**CNTL**  
(continued)

**Keyword      Meaning**

**FX**      **Field- status; changes the status that determines whether the Field- (Field Exit Minus) key is permitted in a field defined as a numeric shift field. The Field- key function is initially enabled at the start of each screen format control series. If it is enabled when this statement is encountered, it becomes disabled. If it is disabled when this statement is encountered, it becomes enabled.**

**SV**      **Specify verify status; changes the status that determines whether the mode is changed from verify mode to special verify mode for this field. If special verify mode is enabled when the cursor enters the field, the mode is changed to special verify mode. Special verify mode allows the operator to enter data into the field without the normal verify checking against the contents of the field in the I/O buffer. When the field is exited in the forward or backward direction, the mode is restored to verify mode. The special verify mode status is disabled at the start of each screen control format. If it is disabled when this statement is encountered, it becomes enabled. If it is enabled when this statement is encountered, it becomes disabled.**

**CS**      **Clear screen; the screen is cleared, except for the status line, prior to processing any other CNTL specifications within this statement.**

**DSPLY**

**Display attributes; specifies display attributes to affect this field only. The IBM 5280 replaces the display attributes currently in effect by moving the attributes you specify in this DSPLY parameter to the screen as immediate data. The IBM 5280 uses the cursor position immediately preceding the field to move the attributes to the screen. It also uses the cursor position immediately following the field to return to the screen the attributes in effect before the change. Remember to include these two cursors if you are counting positions for the CSPS parameter. For example, if you change display attributes for a field that is 8 positions long, the IBM 5280 uses 10 cursor positions.**

**This parameter is optional and should not be used if you specified the display attributes with the HLIN and NMIN parameters of the .KBCRT control statement. You may specify one or more of the following attributes, separated by commas.**

| Parameter Name | Description |
|----------------|-------------|
|----------------|-------------|

| DSPLY<br>(continued) | Symbol | Meaning                                 |
|----------------------|--------|---|
|                      | ND     | Nondisplay of the field                 |
|                      | NM     | Normal display of the field             |
|                      | BL     | Blink the field                         |
|                      | CS     | Display column separators for the field |
|                      | HI     | High intensity for the field            |
|                      | RI     | Reverse image for the field             |
|                      | UL     | Underline the field                     |

**Notes:**

1. The ND (nondisplay) and the NM (normal display) attributes are incompatible with any other specification.
2. If you specify UL (underline), RI (reverse image), and HI (high intensity) for the same field, display of the field is inhibited.
3. The DSPLY parameter has a different effect when specified in a .SFMTCTL control statement than when specified in one of the other statements. Check the DSPLY description for the .SFMTCTL control statement.

|             |  |
|-------------|--|
| <b>BFPS</b> | Buffer position pointer changed; specifies a signed number ( $\pm n$ ) that specifies the direction and number to change the current record buffer pointer position. This pointer determines where the next keystroke is to be placed within the current record buffer. This is an optional parameter. |
|-------------|--|

|             |   |
|-------------|---|
| <b>CSPS</b> | Screen position pointer changed; changes the pointer before any prompts, constant inserts, or input data is displayed on the screen. This is an optional parameter. You may enter one of the following: |
|-------------|---|

| Entry | Meaning |
|-------|---------|
|-------|---------|

|         |   |
|---------|---|
| $\pm n$ | A signed number; indicates the direction and number of the change.  |
| NL      | Next line; places the pointer in the first column of the next line. |

**Notes to CSPS:**

1. You must be careful not to move any data to the screen at a position that will allow the data to extend beyond the available screen positions. If this happens, the data may overwrite the data on another screen.
2. NL is incompatible with ES (execute secondary format), CI (conditional bypass), and CP (continue at current cursor position) specifications.

The IBM 5280 assembler generates a string of object code for each screen control format in your source program. Each series of control statements, which begins with a .SFMTST statement and ends with a .SFMTEND statement, is used to generate one string of object code. This string of object code is referred to as a screen format control string.

The assembler converts the control statements to object code sequentially, so you must code your .SFMT statements in the order you wish them to be executed. Within each source control statement, you may specify parameters in any order. However, there is a prescribed order in which certain parameter keyword specifications are placed in the object code string. This affects the order in which the parameter specifications are processed during program execution. The control statement descriptions specify the order in which the parameter specifications are processed.

The RG (return to program) specification is always placed in the object code string in association with another parameter specification. Therefore, if you specify CNTL=RG in a control statement, you must specify at least one other parameter.

The RG specification is usually associated with the parameter that is first processed in each control statement. When the object code string is being processed in a forward direction and an RG specification is encountered, an external status 4 condition results. The RG specification is encountered *after* the parameter with which it is associated is processed. However, if the object code string is being processed in a backward direction when an RG specification is encountered, an external status 5 condition results. The RG specification is encountered *before* the parameter with which it is associated is processed.

Therefore, if you include a .SFMTCTL control statement that contains only an RG specification and an ES (execute secondary screen format) parameter, or a CI (conditional bypass) parameter, the RG specification is encountered after the ES or CI parameter has been processed. If you want the return (external status 4 condition) to be made before the secondary screen format or the conditional bypass is processed, you must include another parameter in the control statement. The other parameter must be higher in the processing order than the ES or CI parameter. Then the RG return is made, after the other parameter is processed and before the ES or CI parameter is processed.

An exception is made when an RG specification is included in a .SFMTFLD or a .SFMTCNS control statement. For these control statements, the RG specification is always associated with the FLDF parameter or the CNST parameter. This is true even if another parameter (such as BFPS) is included that is higher in the processing order. The return (external status 4 condition) is made *after* the FLDF input field or CNST constant has been entered into the I/O buffer. If the object code string is being processed in a backward direction, the return (external status 5 condition) is made before the cursor enters the FLDF or CNST field.

## **.SFMTST Control Statement**

```
.SFMTST LABEL= [CNTL=] ;
```

The **.SFMTST** control statement identifies the start of a screen format control series. Each screen format control series specifies one format for data input via the keyboard/display unit. Each series must start with a **.SFMTST** statement and must end with a **.SFMTEND** statement. No other **.SFMTST** statements are valid before a **.SFMTEND** statement is encountered.

| <b>Parameter Name</b> | <b>Description</b> |
|-----------------------|--------------------|
|-----------------------|--------------------|

|              |  |
|--------------|--|
| <b>LABEL</b> | Label; identifies this screen format series, which includes all <b>.SFMT</b> statements before a <b>.SFMTEND</b> statement is encountered. This parameter is mandatory. This label is specified as the format operand in an <b>ENTR</b> command. |
|--------------|--|

|             |   |
|-------------|---|
| <b>CNTL</b> | Control keywords; up to four keywords are acceptable; separate them with commas. This is an optional parameter. |
|-------------|---|

| <b>Keyword</b> | <b>Meaning</b> |
|----------------|----------------|
|----------------|----------------|

|           |                    |
|-----------|--------------------|
| <b>RG</b> | Return to program. |
|-----------|--------------------|

|           |  |
|-----------|--|
| <b>MV</b> | Move data; before data entry keystrokes are accepted, the contents of the current record buffer are moved into the appropriate data input and constant insert fields. Prompts and display attributes specified within the screen format series are moved to the screen as specified. |
|-----------|--|

In update, rerun/display, or display mode, this function is automatically invoked.

In verify mode, this function is automatically invoked except that data fields on the screen are blanked.

In enter mode, you must include the **MV** parameter in the **.SFMTST** statement if you want the move data function to be invoked. Otherwise, the prompts and display attributes appear on the screen when they are passed over by the cursor for the first time. The content of the current record buffer does not appear on the screen unless it is rekeyed by the operator.

|           |               |
|-----------|---------------|
| <b>CS</b> | Clear screen. |
|-----------|---------------|

|           |   |
|-----------|---|
| <b>CP</b> | Continue at current cursor position; begins this screen format at the current cursor position. If this keyword is omitted, the cursor position is reset to line two, column one at the beginning of data entry under the enter statement. |
|-----------|---|

*Example:*

```
.SFMTST LABEL=FORMAT1 CNTL=CP,MV;
```

**.SFMTCTL Control Statement**

```
.SFMTCTL [CI= CNTL= DSPLY= CSPS= BFPS= ES=] ;
```

The .SFMTCTL control statement specifies screen or keyboard control operations. Although all parameters are optional, at least one parameter must be specified each time you include the .SFMTCTL statement.

**Parameter  
Name**

**Description**

**CI** Conditional bypass indicator; the label of an indicator, followed by OFF or ON, for a conditional bypass of a portion of this .SFMT series. This must be followed by an end of bypass control parameter (CNTL=CE) in a .SFMTCTL statement within this .SFMT series. All statements between this statement and the .SFMTCTL statement with the CE parameter are bypassed if the specified indicator is on and ON is indicated, or if it is off and OFF is coded. This is an optional parameter. Only one level of bypass is allowed. See *Conditional Bypass* under *Screen Formats* in Chapter 2 for more information.

**CNTL** Control keywords; up to five keywords are allowed. If more than one keyword is entered, separate them with a comma.

**Keyword      Meaning**

**RG** Return to program. Whenever the RG keyword is included for the CNTL parameter of a .SFMTCTL statement, at least one more keyword must also be included.

**CE** End of bypass; concludes the bypass portion within the format control series. A CE specification must have been preceded by a start of bypass (CI) parameter in a previous control statement within this .SFMT series.

**DE** Dup key status change.

**MC** Monocase conversion status.

**FX** Field exit status change.

**SV** Special verify mode status change.

**Parameter  
Name**

**Description**

**DSPLY** Display attribute; specifies a display screen attribute, which is moved to the screen at the current cursor position. The fields preceding the cursor position are not changed. The fields following the cursor position are displayed with the display attributes specified in the .SFMTCTL statement DSPLY parameter until another DSPLY parameter is encountered. Remember to include this cursor position if you are counting positions for the CSPA parameter. This is an optional parameter and should not be used if you specified the display attributes with the HLIN and NMIN parameters of the .KBCRT control statement. One or more of the following keywords, separated by commas, may be entered.

**Keyword      Meaning**

|    |                                 |
|----|---------------------------------|
| BL | Blink                           |
| CS | Column separators are displayed |
| HI | High intensity                  |
| ND | Nondisplay                      |
| RI | Reverse image                   |
| UL | Underline                       |
| NM | Normal display                  |

**Note:** The ND (nondisplay) and the NM (normal display) attributes are incompatible with any other specification. If you specify UL (underline), RI (reverse image) and HI (high intensity) for the same field, display of the field is inhibited.

**CSPA** Screen position pointer changed.

**BFPS** Buffer positions pointer changed.

**ES** Execute secondary screen format; specify the label (LABEL parameter from the .SFMTST statement) of the secondary screen format. When the IBM 5280 encounters this specification, it executes the entire secondary screen format, then returns to the primary screen format at the specification following the ES specification. See *Secondary Screen Format* under *Screen Formats* in Chapter 2 for more information. This is an optional parameter.  
Only one level of secondary format is permitted. The secondary format specified by this parameter must not have a secondary format specification within its statements.

Although the parameters may be specified in any order, the order in which they are processed is:

1. BFPS, CSPS
2. DSPLY
3. FX, MC, DE
4. ES
5. CI
6. CE

If RG is included, it is associated with the parameter that is first in processing order. The return (external status 4 or 5 condition) is made after that parameter is processed in the forward or backward direction (respectively).

*Example:*

```
.SFMTCTL CNTL=DE,FX CSPS=20 BFPS=+15 ES=FMT6 CI=CHK,OFF;
```

#### **.SFMPMT Control Statement**

```
.SFMPMT PRMT= [CSPS= BFPS= DSPLY= CNTL=];
```

The .SFMPMT control statement specifies a prompt to display on the screen. You set up and initialize the prompts with .DC control statements.

When you specify prompts for a screen format, be careful that the prompt length is not greater than the available screen positions. If such a prompt is moved to the screen, the prompt may extend beyond the screen into the screen area used by another keyboard. This may overwrite and destroy the original contents of another screen or destroy control information for your screen.

| <b>Parameter Name</b> | <b>Description</b>   |
|-----------------------|--|
| PRMT                  | Prompt; you must specify one of the following for this mandatory parameter.  |
| SP, label             | Standard position prompt; specifies the LABEL parameter of the .DC statement that set up and initialized the prompt. The indicated prompt is displayed at the current location of the screen position pointer. A standard position prompt is not redisplayed if it is encountered during backspace operations. |

| Parameter Name      | Description   |
|---------------------|---|
| PRMT<br>(continued) | <p>FP, label Fixed position prompt; specifies the LABEL parameter of the .DC statement that set up and initialized the prompt. The indicated prompt is displayed in column 1 of the fixed prompt line. You can specify the fixed prompt line by including the FPLC parameter in the .KBCRT control statement. If you omit the FPLC parameter, the fixed prompt line defaults to line two. The line is cleared before the prompt is displayed. A fixed prompt is redisplayed if it is encountered during backspace operations.</p> <p>CP,nn Clear fixed prompt line. No label may be specified with CP, although an optional length may be specified. If a length (nn) is specified, only the indicated number of positions are cleared on the fixed prompt line. If no length is specified, the entire line is cleared from the screen.</p> |
| CSPS                | Change screen position pointer.   |
| BFPS                | Change buffer position pointer.   |
| DSPLY               | Display attribute specification.  |
| CNTL                | Control; only RG may be specified.  |

Although the parameters may be specified in any order; the order in which they are processed is:

1. BFPS, CSPS
2. DSPLY (turn on an attribute)
3. PRMT
4. DSPLY (turn off the attribute)

If RG is included, it is associated with the parameter that is first in processing order. The return (external status 4 or 5 condition) is made after the parameter with which it is associated is processed in the forward or backward direction (respectively).

*Examples:*

```
.SFMTPMT PRMT=SP,PMESGE3 DSPLY=BL,HI CSPS=+7 CNTL=RG;
```

```
.SFMTPMT PRMT=CP; this clears the fixed prompt line.
```

```
.SFMTPMT PRMT=CP,20; this clears the first 20 positions of the fixed prompt line.
```

## **.SFMTCNS Control Statement**

**.SFMTCNS CNST= [CSPS= BFPS= DSPLY= CNTL=] ;**

The **.SFMTCNS** control statement specifies a constant insert field.

In enter, update, and rerun modes, a constant insert is placed into the I/O buffer and displayed on the screen in the current field position. The cursor does not appear within the constant insert field and the operator cannot enter data into the field.

In verify mode, verification of a constant insert field is done automatically. The contents of the field are compared to the contents of the main storage area declared with a **.DC** control statement and specified with the **CNST** parameter described below. If the verification is successful, a field advance is performed. If the verification is not successful, the cursor is positioned in the leftmost position of the constant insert field and the contents of the field in the I/O buffer are displayed on the screen. A constant insert verify mismatch error is reported. The operator must press the Reset key, then press either the →I(Field Advance) key, or a field correct key. If the →Ikey is pressed, the contents of the field in the I/O buffer remain unchanged and a field advance is performed. If a field correct key is pressed, the contents of the field in the I/O buffer and on the screen are replaced with the contents of the main storage area, then a field advance is performed. Any data key or any function key handled by the IBM 5280 (except a shift key) is invalid after the Reset key is pressed.

| <b>Parameter Name</b> | <b>Description</b>  |
|-----------------------|---|
| <b>CNST</b>           | Constant name; the label assigned to the character string by a <b>.DC</b> statement, which includes the <b>TYPE=PRMT</b> parameter. |
| <b>CSPS</b>           | Change screen position pointer.   |
| <b>BFPS</b>           | Change buffer position pointer.   |
| <b>DSPLY</b>          | Display attributes.   |
| <b>CNTL</b>           | Control; only <b>RG</b> may be specified.   |

Although the parameters may be specified in any order, the order in which they are processed is as follows:

1. **BFPS, CSPS**
2. **DSPLY (turn on attribute)**
3. **CNST**
4. **DSPLY (turn off attribute)**

If **RG** is specified, it is associated with the **CNST** parameter. The return (external status 4 condition) is made after the constant is entered into the I/O buffer.

**Example:**

```
.SFMTCNS CNST=PAYRATE CSPS=NL BFPS=+2 DSPLY=CS,HI,UL CNTL=RG;
```

**.SFMTFLD Control Statement**

```
.SFMTFLD FLDF= [CNTL= CSPS= BFPS= PIC= DSPLY=] ;
```

The .SFMTFLD control statement defines a data field for data entry. Field type and field definition keywords for the FLDF parameter are defined following the .SFMT control statement descriptions.

**Parameter**

**Name**

**Description**

**FLDF**

Field Definition; three positional keywords that specify field type, field length, and field definition. Field type is omitted if the PIC parameter is used.

Field type and field definition keywords are described following the .SFMTEND control statement under *Field Type Keywords* and *Field Definition Keywords*. Following the keyword descriptions is a chart (Figure 3-8) that shows which keywords are mutually exclusive and which require other keywords.

**Note:** Characters considered valid in any field type may be redefined at system configuration.

**Field type, mandatory first position:** Mandatory unless you use the PIC parameter. If you use PIC, omit the field type entry in this position and retain the comma before the length entry. If you do not use PIC, enter one or more of the following, separated by commas.

| Symbol | Meaning                  |
|--------|--------------------------|
| A      | Alphabetic shift         |
| N      | Numeric shift            |
| W      | Special characters shift |
| X      | Alphabetic only          |
| Y      | Numeric only             |
| V      | Special characters only  |
| D      | Digits only              |
| H      | Hex field                |
| S      | Signed numeric           |
| F      | Format level zero        |

**Field Length, mandatory second position:** Enter a number from 1 to the maximum number of valid positions remaining on the screen.

**Parameter  
Name**

**Description**

*Field definition, optional third position:* Enter one or more of the following, separated by commas.

| <b>Symbol</b> | <b>Meaning</b>   |
|---------------|--|
| FE            | Field exit required                                    |
| RB            | Right adjust, blank fill (not allowed if PIC is coded) |
| RZ            | Right adjust, zero fill (not allowed if PIC is coded)  |
| AD            | Auto dup   |
| AS            | Auto skip  |
| ME            | Mandatory enter  |
| DR            | Data required  |
| MF            | Mandatory fill   |
| BC            | Blank check  |
| RL            | Right-to-left (not allowed if PIC is coded)            |
| BV            | Verify bypass  |
| AA            | Absolutely automatic                                   |
| BY            | Bypass   |

**CNTL**

Control keywords; up to four optional keywords are acceptable; separate them with commas.

| <b>Keyword</b> | <b>Meaning</b>                     |
|----------------|------------------------------------|
| RG             | Return to program.                 |
| DE             | Dup key status change.             |
| MC             | Monocase conversion status change. |
| FX             | Field- key status change.          |
| SV             | Special verify mode status change. |

**MD**

Main storage duplication; the label of a storage location defined for duplication by a .DC control statement, using the TYPE = MDUP parameter. In enter, update, or field correct mode, the data from the storage location is duplicated into the field when the Dup key is pressed. Duplication starts at the current position within the field and continues to the end of the field.

In verify mode, the contents of the field are verified against the contents of the storage location. If a mismatch error occurs during verification, the cursor stops at that position, the entire field is displayed, and a verify mismatch error is reported. If the operator presses the Reset key, then again presses the Dup key, the character in the storage location replaces the character in the I/O buffer. Verification then continues to the end of the field.

If the field is also defined as auto dup (AD), the duplication from storage is done automatically (except in field correct mode) for the entire field if the auto dup/skip mode is active or if the absolutely automatic (AA) attribute is also specified for the field.

| Parameter Name | Description  |        |         |   |                  |   |             |   |             |   |               |   |                          |   |                 |   |              |   |                         |
|----------------|--|--------|---------|---|------------------|---|-------------|---|-------------|---|---------------|---|--------------------------|---|-----------------|---|--------------|---|-------------------------|
| MS             | Main storage store; the label of a storage location defined by a .DC using the TYPE = MDUP parameter. In enter, update, verify, or rerun mode, the content of the field is moved into this storage location when the field is exited and if the auto dup switch is on, or if the absolutely automatic (AA) attribute is also specified for the field.  |        |         |   |                  |   |             |   |             |   |               |   |                          |   |                 |   |              |   |                         |
| CSPS           | Change screen position pointer.  |        |         |   |                  |   |             |   |             |   |               |   |                          |   |                 |   |              |   |                         |
| BFPS           | Change buffer position pointer.  |        |         |   |                  |   |             |   |             |   |               |   |                          |   |                 |   |              |   |                         |
| PIC            | <p>Subfield picture definition; defines subfields within the original field.</p> <p>PIC consists of one or more ordered pairs of 2-positional entries; the first position is field length and the second position is field type. The positional entries are separated by commas; each pair of entries is enclosed by parentheses; and the pairs are separated by commas.</p> <p><b>Note:</b> If this parameter is specified, place a blank in the field type (first) position of the FLDF parameter for this .SFMTFLD control statement. The PIC parameter cannot be specified if RB (right adjust, blank fill) or RZ (right adjust, zero fill) is specified for the field definition keyword (third) position of the FLDF parameter for this .SFMTFLD control statement.</p> <p><i>Subfield Length; First Position</i></p> <p>A number from 1 to 8 that specifies the length of the subfield.</p> <p><i>Subfield Type; Second Position</i></p> <table border="1"> <thead> <tr> <th>Symbol</th> <th>Meaning</th> </tr> </thead> <tbody> <tr> <td>A</td> <td>Alphabetic shift</td> </tr> <tr> <td>D</td> <td>Digits only</td> </tr> <tr> <td>H</td> <td>Hexadecimal</td> </tr> <tr> <td>N</td> <td>Numeric shift</td> </tr> <tr> <td>V</td> <td>Special characters shift</td> </tr> <tr> <td>X</td> <td>Alphabetic only</td> </tr> <tr> <td>Y</td> <td>Numeric only</td> </tr> <tr> <td>Z</td> <td>Special characters only</td> </tr> </tbody> </table> | Symbol | Meaning | A | Alphabetic shift | D | Digits only | H | Hexadecimal | N | Numeric shift | V | Special characters shift | X | Alphabetic only | Y | Numeric only | Z | Special characters only |
| Symbol         | Meaning  |        |         |   |                  |   |             |   |             |   |               |   |                          |   |                 |   |              |   |                         |
| A              | Alphabetic shift   |        |         |   |                  |   |             |   |             |   |               |   |                          |   |                 |   |              |   |                         |
| D              | Digits only  |        |         |   |                  |   |             |   |             |   |               |   |                          |   |                 |   |              |   |                         |
| H              | Hexadecimal  |        |         |   |                  |   |             |   |             |   |               |   |                          |   |                 |   |              |   |                         |
| N              | Numeric shift  |        |         |   |                  |   |             |   |             |   |               |   |                          |   |                 |   |              |   |                         |
| V              | Special characters shift   |        |         |   |                  |   |             |   |             |   |               |   |                          |   |                 |   |              |   |                         |
| X              | Alphabetic only  |        |         |   |                  |   |             |   |             |   |               |   |                          |   |                 |   |              |   |                         |
| Y              | Numeric only   |        |         |   |                  |   |             |   |             |   |               |   |                          |   |                 |   |              |   |                         |
| Z              | Special characters only  |        |         |   |                  |   |             |   |             |   |               |   |                          |   |                 |   |              |   |                         |
| DSPLY          | Display attributes.  |        |         |   |                  |   |             |   |             |   |               |   |                          |   |                 |   |              |   |                         |

Although the parameters may be specified in any order, the order in which they are processed is as follows:

1. BFPS, CSPS
2. DSPLY (turn on attribute)
3. DE, MC, FX
4. FLDF
5. DSPLY (turn off attribute)

If RG is included, it is always associated with the FLDF parameter. The return (external status 4 condition) is made after the FLDF input field is entered into the I/O buffer.

*Examples:*

```
.SFMTFLD FLDF=A,10,FE,ME CNTL=RG,DE MS=STORS CSPS=+10 BFPS=+10  
DSPLY=HI;
```

```
.SFMTFLD FLDF= ,20 PIC=(8,A), (6,N), (6,A);
```

#### **.SFMTEND Control Statement**

```
.SFMTEND [CNTL=] ;
```

The .SFMTEND control statement identifies the end of a screen format series.

This must be the last statement of each screen format control series.

#### **Parameter**

| <b>Name</b> | <b>Description</b>  |
|-------------|---|
| <b>CNTL</b> | Control keywords; up to three keywords are acceptable, separated by a comma. This is an optional parameter. |

| <b>Keyword</b> | <b>Meaning</b>   |
|----------------|--|
| BZ             | Sound buzzer; when this statement is encountered at the end of a screen format control series, a buzzer is sounded in the keyboard/display unit. |
| CS             | Clear screen; the screen is cleared except for the status line.  |
| BU             | Keystroke buffering; keystrokes entered after the termination of the ENTR used with screen format control series will be buffered.               |

*Examples:*

```
.SFMTEND CNTL=BZ,CS;  
.SFMTEND;
```

Example of Screen Format Control Series:

```
.SFMTST LABEL=EXAMP1 CNTL=CS,MV;  
.SFMTPMT PRMPT=SP,MSG CSPS=10 DSPLY=HI;  
.SFMTFLD FLDF=,12,DR,BC CNTL=RG CSPS=NL BFPS=5 PIC=(5,A),(7,D);  
.SFMTCNS CNST=KONST CSPS=+5 DSPLY=UL;  
.SFMTFLD FLD=N,20,AD CNTL=DE MD=DUPDATE CSPS=+5 BFPS=1;  
.SFMTEND CNTL=BZ;
```

### Field Type Keywords

| Keyword | Meaning |
|---------|---------|
|---------|---------|

|   |  |
|---|--|
| A | Alphabetic shift. All characters are accepted under this character set definition. The keyboard shift is positioned to the lower symbol of each data key for non-Katakana data entry, proof, and typewriter keyboards. For Katakana keyboards, the shift is positioned to the lower left symbol if the keytop has more than 2 symbols. |
|---|--|

|   |  |
|---|--|
| D | Digits only. Only the digits 0-9 are accepted under this character set definition. If any other character is keyed, an error results. The keyboard shift is positioned as for a numeric shift field (N). |
|---|--|

|   |  |
|---|--|
| F | Format level zero. In enter, update, or verify mode, format level zero specifies a series of 1-byte alphabetic fields. The length of the field specified determines the number of 1-byte fields. |
|---|--|

**Note:** The field definition keywords of the FLDF parameter must not be used if type F is specified.

|   |   |
|---|---|
| H | Hexadecimal. Only hexadecimal characters (0-9, A-F) are accepted under this character set definition. If any other character is keyed, an error results. The keyboard shift is positioned to the upper symbol of each data key for data entry and proof keyboards, or on the lower symbol for typewriter keyboards. |
|---|---|

Each pair of hex digits entered is combined to form one position in the field. For example, if hex 3 and hex F are keyed, 3F is inserted into one record position. The Alpha shift key must be used to select A through F on the data entry or proof keyboards.

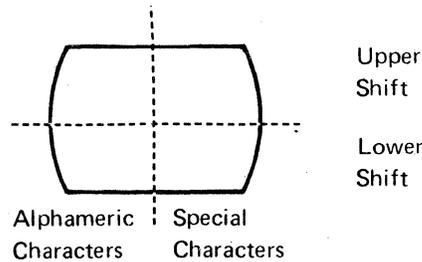
|   |   |
|---|---|
| N | Numeric shift. All characters are accepted under this character set definition. The keyboard shift is positioned to the upper symbol on each data key for non-Katakana data entry and proof keyboards, or to the lower symbol for typewriter keyboards. For Katakana data entry and proof keyboards, the shift is positioned to the upper symbol if the keytop has more than 2 symbols. |
|---|---|

**Keyword    Meaning**

**S**            Signed numeric. The rightmost position of the field on the screen is reserved for a sign; no data can be entered directly into this position. The .SFMTFLD statement that defines the field must not specify digits only (D) for the field type, and must not specify right adjust (RA) and field exit required (FE) for the field definition keywords. These specifications are already implied for a signed numeric field. To exit the field in enter, update, or field correct mode, the Field+, Field Exit, or Field- key must be used. If one of the first two keys is pressed, the data is right-adjusted, the sign position is set to blank on the screen, and the data in the record does not change. If the Field- key is pressed, the data is right-adjusted, the sign position is set to minus on the screen, and the zone portion of the low-order digit of the field is changed to hex D.

This field type cannot be specified for a one-byte field or for a PIC field specification.

**V**            Special characters shift. All characters are accepted under this character set definition. On some native language keyboards certain keytops are divided into quadrants. The two rightside quadrants represent special characters; the two leftside quadrants represent standard alphameric characters. The following illustrates the quadrants.



For a special characters shift field, the keyboard is initially positioned in lower shift. If a key is pressed that has both special and alphameric symbols, the lower rightside special symbol is selected unless a shift key is pressed.

This field type is valid only for the appropriate native language symbols. If it is specified for any other keyboard, a format-control-series-error external status condition (13) occurs.

**X**            Alphabetic only. Only the following characters are accepted under this character set definition: A-Z, comma, period, dash, and blank. If any other character is keyed, an error results. The keyboard shift is positioned to the lower symbol on each data key for data entry, proof, and typewriter keyboards.

**Y**            Numeric only. Only the following characters are accepted under this character set definition: 0-9, comma, period, dash, plus, and blank. If any other character is keyed, an error results. The keyboard shift is positioned to the upper symbol on each data key for data entry and proof keyboards, or on the lower symbol for typewriter keyboards.

**Keyword    Meaning**

**V**            Special characters only. Only characters defined via the character validity table as special characters are accepted under this character set definition. If any other character is keyed, an error results. The keyboard shift is positioned as in a special character shift field. This field type is valid only for the appropriate native language keyboards.

**Field Definition Keywords**

**Keyword    Meaning**

**AA**            Absolutely automatic. In enter, verify, field correct, update, or rerun mode, this keyword is used in conjunction with auto dup (AD), auto skip (AS), or main storage (MS). Automatic processing of the dup, skip, or store is done whether or not the auto dup/skip mode is active. If the field is defined as auto dup (AD), main storage dup (MD), and absolutely automatic (AA), automatic duplication from main storage occurs.

**AD**            Auto duplicate. AD may be specified alone or in conjunction with main storage duplicate (MD). In an enter or update mode, if a field is defined only as auto dup, the system automatically duplicates data from corresponding positions in the previous record if the auto dup/skip mode is active, or if the absolutely automatic (AA) attribute is also specified for the field. The duplication starts at the current position within the field and continues to the end of the field.

If a field is defined as auto dup (AD) and main storage dup (MD), the duplication occurs as described under the main storage dup (MD) function.

In verify mode, if a field is defined as auto dup (AD), the system automatically verifies that the data in the current record buffer matches the data in corresponding positions in the previous record buffer. Verify mismatch error recovery from an AD field is identical to the error recovery from manual dup.

If the auto dup function is invoked, all character and field edit checks are released for the field; the verify checks are not released.

**AS**            Auto skip. When an auto skip field is entered, the system automatically fills it with blanks and skips to the next field if the auto dup/skip mode is active or if the absolutely automatic (AA) attribute is specified in the field definition.

In verify mode, the positions of the field are verified for blanks. If a non-blank character is encountered, the cursor stops at the position, the remainder of the field is displayed, and a verify mismatch error is reported. If the operator presses the Reset key, then presses the Skip key, a blank is inserted into the I/O buffer at that position and the verification continues.

**Keyword    Meaning**

- BC**        Blank check. In enter, update, or field correct mode, the system checks each character as it is entered to ensure that it is not a blank character. When a field is exited, the system checks the entire field to ensure that if one character is entered into the field, no blanks are entered into the field. Therefore, should an operator move the cursor over a number of blank positions in the field before keying in the first character, the blank positions are detected before the field is exited.
- The Skip, Field Exit, or Field+ key is valid in the first position of the field; the entire field is filled with blanks.
- In verify mode, the BC is ignored except during a field correct.
- The code for the blank may be defined on the .KBCRT statement or defaulted to hex 40.
- BV**        Bypass in verify mode. In verify mode, the system bypasses this field. No verification is required, and the data in the record is left unchanged. BV overrides the auto dup (AD) and auto skip (AS) functions.
- In enter or update mode the field is processed normally.
- BY**        Bypass. In enter, update, verify, or rerun mode, the system passes over this field and allows no data to be entered into it. Data already in the field is left unchanged. (The auto dup/skip switch does not affect processing of bypass fields.)
- DR**        Data required. In enter, update, or field correct mode, the system ensures that at least one nonblank character is present in the field. The checking is done as the advance to the next field is being processed.
- In verify mode, DR is ignored except during a field correct.
- FE**        Field exit required. In enter, update, verify, or field correct mode, a nondata key (for example Field Exit or Skip) must be pressed to leave the field. When a data character has been keyed into the last position of the field, the cursor remains beneath that character and blinks to signal the operator that a field exit key is needed. If the field is signed numeric (S), the cursor remains to the left of the sign position. A data key or Dup key entered at this time results in an error. The counter for positions remaining in the field, which is maintained on the status line, is at 01. If the operator wishes to make a correction in the field, pressing the ← key will turn off the blinking cursor and allow corrections to be made.
- FE must not be specified if RB or RZ is specified.

**Keyword    Meaning**

**ME**        **Mandatory enter.** In enter, update, or field correct mode, at least one data character must be entered into the field before the field is exited. Blanks are acceptable in a mandatory enter (ME) field; use the blank check (BC) specification if you do not want blanks to be entered into the field. The check is done as the advance to the next field is being processed.

The ME is ignored in verify mode, except during a field correct.

A field exit or Skip key pressed in the first position of an ME field does not satisfy the mandatory enter requirement; it results in an error. If a field exit or Skip key is pressed in any position other than the first position and no valid character has yet been entered, it results in an error.

**MF**        **Mandatory fill.** In enter, update, or field correct mode, the system ensures that if one character is entered into the field, all positions in the field must be filled. Blanks are acceptable.

MF cannot be specified with a right-adjust (RA) specification.

In a left-to-right field, any data entry must begin in the leftmost position of the field. After one data character is entered, the operator can proceed through the field only by keying each position. Any attempt to move forward in the field with a key other than a data key or space key results in an error. Any attempt to key the first character into a position other than the leftmost position results in an error. The Ins (Insert) and Del (Delete) keys are invalid.

In a right-to-right field (RL), any data entry must begin in the rightmost position of the field. After one data character is entered, the operator can proceed through the field only by keying each position. Any attempt to move backward in the field with a key other than a data key or space key results in an error. An attempt to key the first character into a position other than the rightmost position results in an error.

If the field is not filled with blanks when the cursor moves into the field, no mandatory fill (MF) checking is performed.

**Note:** Because in verify mode the Field Correct key initially fills the field with blanks, the mandatory fill function becomes enabled (if specified) during the field correct.

The Skip, Field+, or Field- key is valid in the first position of a mandatory fill (MF) field. The entire field is filled with blanks, except in the following two conditions:

1. When the Field- key is pressed in the first position of a field that is not specified as signed numeric (S), the rightmost position is set to minus zero.
2. When the Field- key is pressed in the first position of a signed numeric field, the rightmost field position on the screen is set to zero, and the sign position is set to minus. In the record buffer, the rightmost position is set to minus zero.

**Keyword    Meaning**

**RB**        Right adjust, blank fill. A right adjust field must be two or more bytes in length. When the Field Exit, Field+, or Field- key is pressed in enter, update, or field correct mode, the system automatically right adjusts the data within the field, and the nondata positions on the left are filled with the alphabetic fill character, which is normally blank. When a right adjust is performed in a signed numeric field, the data is justified to the next rightmost position of the field on the screen before the appropriate sign is inserted. If the field is processed as an auto field, or if any other key is used to exit the field, the field is processed but no right adjust is performed.

RB is not allowed for one-byte fields or if subfields are defined with the PIC parameter. Do not specify FE; field exit required is implied for a right adjust field.

**RZ**        Right adjust, zero fill. RZ is as for RB, except the numeric fill character, which is normally a zero, is used.

**RL**        Right-to-left. The first position of the field is the rightmost position; the last position of the field is the leftmost position. In enter, update, field correct, or verify mode, the cursor is initially positioned in the rightmost position of the field, which is the first manual position of the field. The system accepts or verifies data in the field moving from the rightmost position to the leftmost position. (For example, the keys A, B, C entered into an RL field would appear as:

   \_\_CBA

with the cursor positioned to the left of the C.)

RL is not allowed if subfields are defined with the PIC parameter, or if the field is a signed numeric, numeric, numeric-only, or digits-only field.



| Parameter Name | Description   |
|----------------|---|
| INIT           | Initialization; a character string of 1 to 32 characters enclosed in single quotes. |

*Example:*

```
.TITLE INIT='ASSEMBLER DRIVER';
```

#### **.EJECT Control Statement**

```
.EJECT ;
```

The **.EJECT** control statement stops the printing of the assembly listing on the current page and continues the printing on the next page. Use this statement whenever you wish to skip to a new page.

*Example:*

```
.EJECT;
```

#### **.SPACE Control Statement**

```
.SPACE [NUMB=] ;
```

When the assembler encounters a **.SPACE** control statement it inserts one or more blank lines in the assembly listing. This control statement should be included in the source program in each position where you want the assembly listing to leave blank lines.

| Parameter Name | Description  |
|----------------|--|
| NUMB           | Number; the number of blank lines to insert. If the parameter is omitted, it defaults to one line. The maximum is 255. |

*Example:*

```
.SPACE NUMB=4;
```

### **.PRINTON Control Statement**

**.PRINTON ,**

The **.PRINTON** control statement allows you to print an assembly listing after the listing has been disabled by a **.PRINTOFF**.

*Example:*

**.PRINTON;**

### **.PRINTOFF Control Statement**

**.PRINTOFF ;**

The **.PRINTOFF** control statement disables the printing of an assembly listing. If an assembler listing line is flagged as having an error in it, that line will be printed even if **.PRINTOFF** is in effect.

*Example:*

**.PRINTOFF;**

### **MISCELLANEOUS CONTROL**

The miscellaneous control statements specify a data set to insert into your object program, specify the labels of the common function subroutines your program uses, and indicate the end of your source program.

## **.INCLUDE Control Statement**

```
.INCLUDE NAME= ;
```

The `.INCLUDE` control statement allows you to insert source code from another data set into your current work file. When the assembler encounters an `.INCLUDE` statement, it stops reading your source data set, goes to the specified data set and reads it from beginning to end, then returns to your source data set. The source data set that contains the code to insert is not changed. The source file that contains your source program is not changed.

### **Parameter**

| <b>Name</b> | <b>Description</b>   |
|-------------|--|
| NAME        | Data set name; the name of data set you wish to insert into the current data set. Use the complete data set name, enclosed in single quotes. |

*Example:*

```
.INCLUDE NAME='PROG7' ;
```

## **.SELFCHK Control Statement**

```
.SELFCHK MOD= [LABEL= FLDLEN= WGTS= CNTL=  
DISP= ALTREG= INTAB= OUTTAB= PROD=] ;
```

The `.SELFCHK` control statement defines the self-check field, the self-check register, the modulus, and the algorithm for the self-check function. The assembler uses the parameters of the `.SELFCHK` control statement to set up the self-check control area. During program execution, the IBM 5280 uses this control area to generate the self-check number during a GSCK or IF . . . CK operation.

If you use Standard Modulus 10 or 11, see *Choosing Your Algorithm* under *Self-Check* in Chapter 2 for a description of the algorithm. If you design your own algorithm, see Appendix C for a description of how the `.SELFCHK` parameters define your algorithm.

### **Parameter**

| <b>Name</b> | <b>Description</b>  |
|-------------|---|
| MOD         | Modulus; specifies the modulus for the self-check algorithm. If you design your own algorithm, you may specify a number from 2 to 127 and use the other parameters to define the algorithm. Or you may specify S10 or S11 to use Standard Modulus 10 or 11 and omit the remaining parameters. |
| LABEL       | Label; specifies the alphameric name you wish to assign to the self-check control area.   |
| FLDLEN      | Field length; the number of bytes in the self-check field.  |

| Parameter Name | Description   |
|----------------|---|
| WGTS           | <p>Weighting factors; specifies up to 32 bytes of hex digits that act as the weights for the algorithm. The value of each byte must be less than the value of the modulus. Enter hex 00 in the positions of the self-check number, and in all other positions that are bypassed. When you use a product table, enter hex 01 in all positions except the positions of the self-check number and the positions to be bypassed.</p> <p>Begin the WGTS parameter in column 1 of a new line, and code the complete parameter on the single line. If you use a decimal register pair for the self-check register, all 72 positions of the line are used, as in the example following the parameter descriptions.</p>  |
| CNTL           | <p>Control; six positional fields specify how to find the products and how to determine the NR (rightmost number) and NL (leftmost number). The positional fields are separated by commas. Any of the six fields may be omitted, but the comma for the omitted field must be retained.</p> <p>Field 1; if the field 1 entry is omitted and PROD is not specified for a product table, each position of the foundation is multiplied by the corresponding weight, and the whole number products are summed. The sum becomes the NR, and NL is 0.</p> <p>If the field 1 entry is omitted and PROD is specified for a product table, the product table repeats every three characters. The NR is found by translating each position of the foundation through the upper half of the product table to find the products, and adding each digit of the products. The NL is found by translating each position of the foundation through the lower half of the product table to find the products, and adding each digit of the products.</p> <p>D Each position of the foundation is multiplied by the corresponding weight, and the unit digits of the products are summed. The modulus is usually 10 if this option is used.</p> <p>U Each position of the foundation is multiplied by the corresponding weight, and the unit digits of the products are summed.</p> <p>F Each position of the foundation is translated through the upper half of the product table to a product, and the digits of the products are summed to find the NR. To find the NL, each position of the foundation is translated through the lower half of the product table to a product, and each digit of the products are summed. The product table repeats every fourth digit.</p> |

| Parameter Name      | Description  |
|---------------------|--|
| CNTL<br>(continued) | <p>Field 2; If the field 2 entry is omitted, the NL (leftmost self-check number) is forced to zero, and NR (rightmost self-check number) is divided by the modulus.</p>  |
|                     | <p>D The NL is forced to zero, the digits of the NR are summed, and the sum is divided by the modulus.</p>   |
|                     | <p>K The NL and NR are added, and the digits of the sum are cross added. The hundreds digit of this sum is added to the units digit to equal the NR, and the tens digit is added to the carry from NR to equal the NL.</p>                                 |
|                     | <p>E For special modulus 8 and 3; the units position of the self-check number is converted to modulus 8, and the tens position is converted to modulus 3. Field 1 must not be omitted, and C cannot be specified in field 3 for a modulus less than 8.</p> |
|                     | <p>Field 3; If the field 3 entry is omitted, the NL and NR remain unchanged.</p>   |
|                     | <p>C The NL and NR are complemented to the modulus.</p>  |
|                     | <p>Field 4; If the Field 4 entry is omitted, it defaults to 1.</p>   |
|                     | <p>1 One digit is generated or checked. If K is entered in Field 2 and an output translate table is used, the NL and NR are summed and the sum is translated through an output translate table.</p>  |
|                     | <p>2 Two digits are generated or checked. If E is entered in Field 2, the NL is multiplied by 8, the product is added to the NR, and the sum is translated through an output translate table.</p>  |
|                     | <p>Field 5; If the Field 5 entry is omitted, the zone portion of NL and NR is forced to X'F' to produce the DL (displayable leftmost self-check digit) and DR (displayable rightmost self-check digit).</p>  |
|                     | <p>D The NR is used to produce a 2-digit decimal number. The units digit is converted to the DR, and the tens digit is converted to the DL.</p>  |
|                     | <p>If the result of this operation exceeds 99, the units digit output is correct, and the second digit has a zone portion of hex F and a digit portion of hex A-C.</p>   |
|                     | <p>Field 6; If the Field 6 entry is omitted, all eight bits of each byte in the input translate field are used for the input translate number.</p>   |

| <b>Parameter Name</b> | <b>Description</b>  |
|-----------------------|---|
| CNTL<br>(continued)   | F Each byte in the input translate field is interpreted as two hex digits. The rightmost hex digit becomes the input translate character, and the leftmost hex digit becomes a shift left count. The positions being translated and all higher positions in the register are shifted left, with zero fill, the number of positions in the shift count when the shifted register contains 16 bytes. All unused high-order bytes of the original register are bypassed.   |
| DISP                  | Displacement; specifies the displacement (0-32) of the rightmost self-check digit from the leftmost position of the register. It defaults to zero.<br><br>If the displacement is zero, the result of a GSCK operation must also be zero to pass the IF . . . CK operation. If the displacement is 1, and two self-check digits are specified, the leftmost digit of the self-check computation must be zero to pass the IF . . . CK operation. (The leftmost result of the GSCK operation is not stored.)                   |
| ALTREG                | Alternate register; specify a decimal register or register pair that contains the weighting factors for the self-check algorithm. If ALTREG is specified, the weights specified by the WGTS parameter are ignored.  |
| INTAB                 | Input translate table label; specify the LABEL entry from the .DC control statement that defined the table. The input translate table can translate all foundation characters to specific hex characters. If this parameter is omitted, the lower 4 bits of the EBCDIC of each foundation position translates to the numerals 0-9, with A-F translating to 0.   |
| OUTTAB                | Output translate table label; specify the LABEL from the .DC control statement that defined the table. The value of the self-check digit determines the buffer position of the character to be inserted into the self-check register. If omitted, the output self-check register is not translated.<br><br>If one digit is to be generated and Field 2 specifies K, the NL and NR are added and the sum is translated.<br><br>If Field 2 specifies E, the NL is multiplied by 8 and added to NR, and the sum is translated. |

| Parameter Name | Description   |
|----------------|---|
| PROD           | Product table label; specify the LABEL from the .DC control statement that defined the product table. The product table translates the rightmost four digits of each byte in the self-check register to the product at the corresponding position in the product table. If two self-check digits are to be generated, the second product is displaced 64 positions from the first product. If this parameter is omitted, the products are found by using the weights.<br><br>If the Field 2 entry is omitted, or if the Field 1 entry is U, the NL is forced to zero. |

*Examples:*

.SELFCHK MOD=S11; no more parameters needed for Standard Modules

.SFLFCHK MOD=7 LABEL=SKAREA INTAB=SKINTAB OUTTAB=SKOUTAB  
WGTS=X'0605040503020105060103010401020304050604010106060403020101020300'  
FLDLLEN=16 CNTL=D, , ,1,D;

**.XTRN Control Statement**

.XTRN LABEL= ;

The .XTRN control statement specifies the labels of the routines or global tables in the common function area that your program uses. This control statement reserves all specified labels as common function labels. This is a required control statement if any calls to the common function area are initiated. See *Common Function Routines* in Chapter 6 for the labels to specify.

| Parameter Name | Description  |
|----------------|--|
| LABEL          | Label; label of one or more common function routines or data areas. You may specify up to 30 labels on each .XTRN statement. Labels are separated by a comma. This parameter is mandatory. |

*Example:*

The following statement specifies that the program is using the standard load processor, program check error handler, and general I/O error handler.

.XTRN LABEL= CFLOAD01,CFPGMCHK,CFGIOERR;

## **.END Control Statement**

**.END ;**

The **.END** control statement is mandatory and must be the last control statement of every program to specify the end of the source code. There are no parameters for the **.END** statement. All system tables are built when the **.END** statement is processed. If a **.SYSTAB** control has not caused the system tables to be built in another location, the system tables are built at the address in the current location counter when the **.END** control statement is encountered. If a **.SYSTAB** statement has caused the system tables to be located at other addresses, they are built at those addresses when the **.END** statement is encountered.

### *Example:*

**.END; End of PAYROLS**



## Chapter 4. IBM 5280 Assembler Language Instructions

Source instructions specify program operations. The IBM 5280 assembler generates 4 bytes of object code from each source instruction.

Instructions may be interspersed with certain control statements, according to the conventions detailed in Chapter 3.

### INSTRUCTIONS FORMAT

Each source instruction contains one or more fields; each field is identified by the order of its position within the instruction. Blanks, commas, and parentheses separate the fields. Each instruction must be written on a single line, between column one and column 72. The format of a source instruction consists of an optional symbolic label, the instruction, and an optional comment.

```
[Label:] Instruction [;Comment]
```

#### Blanks

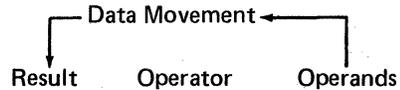
Optional blanks may be placed before and after an equal sign, parenthesis, or arithmetic operator. Blanks are not allowed within a field or within a binary arithmetic/logical operator. One or more blanks must be used to separate fields if no other delimiter is used.

#### Symbolic Labels

A symbolic label is a character string of one to eight characters. The first character must be an uppercase alphabetic character (A-Z, \$, @, #). The other characters may include any uppercase or lowercase character available to your keyboard. The label must begin in column 1 and must be followed by a colon (:).

## The Instruction Fields

Blanks, commas, and parentheses are used to separate fields, depending upon the particular type of instruction. If commas are used to separate optional fields, the commas must be retained when fields are omitted that are to the left of fields that are specified. The general format of an instruction is:



where:

*Result* specifies the address that contains the resulting data at the completion of the operation.

*Operator* may be an arithmetic symbol or a mnemonic to specify the operation. Mnemonics may be to the left or to the right of the result, depending upon the particular operation.

*Operand* may include one or more storage, register, or constant specifications, depending upon the particular operation.

## Comments

A comment may be included on any line following an instruction statement. The comment is preceded by a semicolon. An entire line may be specified as a comment line by placing an asterisk (\*) in the first column of the line.

*Examples:*

LABEL: INSTRUCTION; this is a comment on an instruction line.

\*This whole line is a comment line.

## STORAGE SPECIFICATIONS

An instruction may refer to a location in storage by specifying one of the following addressing methods:

- Labeled addressing.
- Base displacement addressing.

The length of the data area may optionally be specified with either method. These addressing methods are described in detail in Chapter 1 under *Storage*.

Whenever an instruction description includes the word *storage* either addressing method may be used. If the instruction allows only one of the methods, the format of that addressing method is indicated in the instruction description.

## Labeled Addressing

A storage area may be declared and labeled by a .DC control statement. An instruction may refer to that labeled storage area by specifying the following format:

label [(length)]

The label is the name assigned to the area by the .DC control statement. If the optional length is omitted, it defaults to the length declared in the .DC statement. If you want to access a number of bytes that is different from the declared length, include the number in parentheses to the right of the label.

## Base Displacement Addressing

An instruction may refer to a location in storage by using a base displacement specification. The format of a base displacement specification is as follows:

[displacement] [(length),BRn)

The binary register holds the base address. The optional displacement, which may be from 0 to 255 and defaults to 0, is added to the base address. The result is the address of the first byte of the storage area. The optional length of the area defaults to one byte. Except for the IFB instruction and store-a-constant instruction (under *Store at Base Displacement Address* in this chapter), the comma is retained if no length is specified.

## CONSTANT SPECIFICATIONS

Certain instruction operands may be specified as a constant. Constants may be specified in the forms described under *Blanks, Constants, and Coding Symbols* in Chapter 1. A 1-byte constant may be specified in any of the following forms:

- A decimal number, 0-255
- Two hexadecimal digits, X'II', where I = 0-9, A-F.
- Eight binary digits, B'IIIIIIII', where I = 0 or 1.
- One EBCDIC character, C'I', where I = any printable graphic.

The following example demonstrates the way the four forms of constants represent the same value:

| Form        | Representation |
|-------------|----------------|
| Decimal     | 193            |
| Hexadecimal | X'C1'          |
| Binary      | B'11000001'    |
| EBCDIC      | C'A' or 'A'    |

Whenever *constant* is specified in an instruction description, any form of constant may be specified in the source instruction.

## REGISTER AND INDICATOR SPECIFICATIONS

An instruction may refer to an indicator or a reserved register by specifying the indicator or register number (In, BRn, or Rn). Registers may be reserved with the RGLT parameter of the .START statement.

Labels can be assigned to indicators and registers. The control statements that assign these labels are discussed in Chapter 1 under *Indicators and Registers*. If an indicator or register has been assigned a label, an instruction may refer to it by specifying the label.

In the instruction descriptions in this chapter, indicators and registers are indicated by number specifications. Any time such a number specification is indicated either the number or label may be used.

In certain instructions an optional length may be indicated in a register specification. By indicating the length, in parentheses, to the right of the register specification, you can access a binary half register (BRn(1)), a binary double register (BRn(4)), or a decimal double register (Rn(32)).

## OPERATION TYPES

Source instructions are listed in this chapter by the type of operation the instruction performs. The instructions are divided into the following operation types:

|   |                    |   |                         |
|---|--------------------|---|-------------------------|
| A | Arithmetic/Logical | K | Keyboard/Display        |
| B | Branch and Skip    | M | Data Movement           |
| C | Communications     | P | Partition Load and Exit |
| D | Diskette           | T | Table                   |
| E | Printer            | X | Miscellaneous           |

In Figure 4-1, all mnemonics are listed in alphabetic order. The operation type and a brief description of the operation performed are included for each instruction mnemonic.

Figure 4-2 shows the operation types and lists the mnemonics for each type.

| Mnemonic | Type  | Operation                             |
|----------|-------|---------------------------------------|
| +        | A1    | Decimal add                           |
| -        | A2    | Decimal subtract                      |
| *        | A2    | Decimal multiply                      |
| /        | A2    | Decimal divide                        |
| +=       | A1    | Binary add                            |
| -=       | A1    | Binary subtract                       |
| *=       | A1    | Binary multiply                       |
| /=       | A1    | Binary divide                         |
| &=       | A1    | Binary AND                            |
| V=       | A1    | Binary OR                             |
| X=       | A1    | Binary exclusive-OR                   |
| <=>      | M1    | Exchange register data                |
| ALLOC    | D1    | Allocate diskette                     |
| BINDEC   | M2    | Convert binary storage to EBCDIC      |
| BINHEX   | M2    | Convert binary to EBCDIC storage      |
| BRa=     | M1    | Load binary register                  |
| BRn=     | M2    | Convert decimal register to binary    |
| (,BRn)=  | M1    | Store at base displacement address    |
| BUZZ     | K2    | Sound buzzer                          |
| CALL     | B2    | Call subroutine                       |
| CALLTB   | B2    | Call subroutine through table         |
| CLC      | X     | Compare logical characters            |
| CLICK    | K2    | Click keyboard                        |
| CLOZ     | D1, E | Close data set                        |
| CNENTR   | K2    | Cancel current ENTR                   |
| CRTMM    | M3    | Move bytes to screen                  |
| DECBIN   | M2    | Convert EBCDIC to binary storage      |
| DECR     | B6    | Decrement, branch if 0                |
| DISPEX   | K2    | Display extra line                    |
| DISPST   | K2    | Display status line                   |
| DUP      | X     | Duplicate a byte                      |
| ENABLE   | K1    | Enable external status                |
| ENTR     | K1    | Enter data via keyboard               |
| EXIT     | P     | Exit partition                        |
| GOTAB    | B1    | Branch through table                  |
| GOTO     | B1    | Unconditional branch                  |
| GSKC     | X     | Generate self-check digit             |
| HEXBIN   | M2    | Convert EBCDIC storage to binary      |
| IF BRn   | B4    | Branch on relational compare, binary  |
| IF BRn 0 | B3    | Branch if 0, binary                   |
| IF Rn    | B4    | Branch on relational compare, decimal |
| IF Rn -  | B3    | Branch if negative                    |
| IF Rn 0  | B3    | Branch if 0, decimal                  |
| IF Rn AN | B3    | Branch if positive number             |
| IF Rn CK | B3    | Branch if self check                  |
| IF Rn SN | B3    | Branch if signed numeric              |
| IF fmt   | B3    | Branch on format test                 |
| IFB      | B5    | Skip on bit test                      |
| IFC      | B5    | Skip on character test                |
| IFD Rn   | B4    | Branch on decimal compare             |
| IFDSI    | B3    | Branch on data set indicator test     |

Figure 4-1 (Part 1 of 3). Instruction Mnemonics

| Mnemonic   | Type  | Operation                            |
|------------|-------|--------------------------------------|
| IFHI       | B5    | Skip on mask                         |
| IFLO       | B5    | Skip on mask                         |
| IFH BRn    | B4    | Branch on immediate data compare     |
| IFI        | B3    | Branch on indicator test             |
| IFIR       | B3    | Branch on indicator test, and reset  |
| INIT       | D1    | Initialize diskette labels           |
| INSBLK     | D2    | Insert block into data set           |
| INXEQ      | X     | Indirect statement execution         |
| KACCPT     | K2    | Accept unformatted keystrokes        |
| KATTCH     | K2    | Attach keyboard to partition         |
| KDETCH     | K2    | Detach keyboard from partition       |
| KERRCL     | K2    | Reset error mode                     |
| KERRST     | K2    | Request error mode                   |
| KEYOP      | K2    | Keyboard operation                   |
| label=     | M1    | Store at labeled address             |
| LOAD       | P     | Load partition                       |
| MMCRT      | M3    | Move from screen to storage          |
| MOFF       | M3    | Move to decimal register offset      |
| MVC        | M3    | Move storage to storage              |
| MVCR       | M3    | Move storage right to left           |
| MVCV       | M3    | Move storage reverse fill            |
| MVER       | M3    | Move to corresponding decimal byte   |
| NOP        | B1    | Null operation                       |
| OPEN       | D1, E | Open a data set                      |
| POSN       | D1    | Position diskette pointer            |
| Ra=        | M1    | Load decimal register                |
| Rn=        | M2    | Convert/move to decimal register     |
| READ       | D2    | Read a data set record               |
| READMG     | K2    | Read magnetic stripe buffer          |
| REBF       | M3    | Move data to registers               |
| REPFLD     | K2    | Redisplay field on screen            |
| RESCAL     | K1    | Resume entry, call subroutine        |
| RESMXT     | K1    | Resume entry, enable external status |
| RESUME     | K1    | Resume entry                         |
| RETEXT     | K1    | Return and enable external status    |
| RETURN     | B2    | Return from subroutine               |
| RL         | A1    | Binary rotate left                   |
| RR         | A1    | Binary rotate right                  |
| RSTMG      | K2    | Reset magnetic stripe reader         |
| RTIMER     | K2    | Read interval timer                  |
| RXORW      | X     | Exclusive-OR, write                  |
| SEARCH     | D3    | Search a data set                    |
| SETOFF     | X     | Set off bit                          |
| SETON      | X     | Set on bit                           |
| SKIP WHILE | B6    | Loop control                         |
| SL         | A     | Shift left, binary or decimal        |
| SLS        | A2    | Shift left signed, decimal           |
| SR         | A     | Shift right, binary or decimal       |
| SRAT       | X     | Search resource allocation table     |
| SRR        | A2    | Shift right and round, decimal       |
| SRS        | A2    | Shift right signed, decimal          |

Figure 4-1 (Part 2 of 3). Instruction Mnemonics

| <b>Mnemonic</b> | <b>Type</b> | <b>Operation</b>                 |
|-----------------|-------------|----------------------------------|
| SOFF            | X           | Set indicator off                |
| SON             | X           | Set indicator on                 |
| SYSLCK          | X           | Lock system                      |
| SYSUNL          | X           | Unlock system                    |
| TBBS            | T3          | Search table, binary             |
| TBDL            | T2          | Delete table entry               |
| TBFH            | T3          | Search table for equal or higher |
| TBFL            | T3          | Search table for lower entry     |
| TBFX            | T3          | Search table for equal entry     |
| TBIN            | T2          | Insert table entry               |
| TBRD            | T2          | Read table entry                 |
| TBRL            | T2          | Read last table entry            |
| TBWE            | T2          | Extend table and write entry     |
| TBWT            | T2          | Write table entry                |
| TCLOZ           | C           | Close communications data set    |
| TCTL            | C           | Communications device control    |
| TINIT           | C           | Initiate communications session  |
| TLCK            | T1          | Lock shared table                |
| TOPEN           | C           | Open communications data set     |
| TRANS           | X           | Translate                        |
| TREAD           | C           | Receive communications record    |
| TRT             | X           | Translate and test               |
| TTERM           | C           | Terminate communications         |
| TUNLCK          | T1          | Unlock shared table              |
| TWAIT           | C           | Wait for communications I/O      |
| TWRT            | C           | Transmit communications record   |
| WAIT            | D1, E       | Wait for I/O completion          |
| WFMCR           | M3          | Move data to screen              |
| WRBF            | M3          | Store register data              |
| WRT             | D2, E       | Write a data set record          |
| WRTI            | D2          | Insert data set record           |
| WRTS            | D2          | Delete data set record           |
| ZONE            | A2          | Modify decimal register zone     |

**Figure 4-1 (Part 3 of 3). Instruction Mnemonics**

|           |                                   |          |         |
|-----------|-----------------------------------|----------|---------|
| <b>A1</b> | <b>Arithmetic/Logical Binary</b>  |          |         |
|           | +=                                | &=       | RL      |
|           | -=                                | V=       | RR      |
|           | *=                                | X=       | SL      |
|           | /=                                |          | SR      |
| <b>A2</b> | <b>Arithmetic/Logical Decimal</b> |          |         |
|           | +                                 | SL       | SR      |
|           | -                                 | SLS      | SRS     |
|           | *                                 |          | SRR     |
|           | /                                 |          | ZONE    |
| <b>B1</b> | <b>Branch Unconditional</b>       |          |         |
|           | NOP                               | GOTO     | GOTAB   |
| <b>B2</b> | <b>Subroutine</b>                 |          |         |
|           | CALL                              | RETURN   | CALLTB  |
| <b>B3</b> | <b>Branch Conditional, Full</b>   |          |         |
|           | IF BRn 0                          | IF Rn AN | IF fmt  |
|           | IF Rn 0                           | IF Rn CK | IFDSI   |
|           | IF Rn -                           | IF Rn SN | IFI     |
|           |                                   |          | IFIR    |
| <b>B4</b> | <b>Branch Conditional, Short</b>  |          |         |
|           | IF Rn                             |          | IF BRn  |
|           | IFD Rn                            |          | IFH BRn |
| <b>B5</b> | <b>Skip</b>                       |          |         |
|           | IFC                               | IFHI     |         |
|           | IFB                               | IFLO     |         |
| <b>B6</b> | <b>Loop Control</b>               |          |         |
|           | SKIP WHILE                        |          | DECR    |
| <b>C</b>  | <b>Communications</b>             |          |         |
|           | TINIT                             | TREAD    | TWAIT   |
|           | TOPEN                             | TWRT     | TCLOZ   |
|           | TCTL                              |          | TTERM   |
| <b>D1</b> | <b>Diskette Control</b>           |          |         |
|           | ALLOC                             | CLOZ     | INIT    |
|           | OPEN                              | WAIT     | POSN    |

Figure 4-2 (Part 1 of 3). Instruction Types

|           |   |         |        |
|-----------|---|---------|--------|
| <b>D2</b> | <b>Diskette I/O</b>                         |         |        |
|           | READ  | WRTS    | WRTI   |
|           | WRT   |         | INSBLK |
| <b>D3</b> | <b>Diskette Search</b>                      |         |        |
|           | SEARCH                                      |         |        |
| <b>E</b>  | <b>Printer Instructions</b>                 |         |        |
|           | OPEN  |         | WAIT   |
|           | WRT   |         | CLOZ   |
| <b>K1</b> | <b>Keyboard/Display Key Entry Commands</b>  |         |        |
|           | ENTR  | RESUME  | RESCAL |
|           | ENABLE                                      | RETEXT  | RESMXT |
| <b>K2</b> | <b>Keyboard/Display Keyboard Operations</b> |         |        |
|           | BUZZ  | KACCPT  | KEYOP  |
|           | CLICK                                       | KATTCH  | READMG |
|           | CNENTR                                      | KDETC   | RSTMG  |
|           | DISPEX                                      | KERRCL  | REPFLD |
|           | DISPST                                      | KERRST  | RTIMER |
| <b>M1</b> | <b>Load, Store, and Exchange</b>            |         |        |
|           | BRa=  | label=  | BRa<=> |
|           | Ra=   | (,BRn)= | Ra<=>  |
| <b>M2</b> | <b>Convert and Move</b>                     |         |        |
|           | Rn=   | BINHEX  | BINDEC |
|           | BRn=  | HEXBIN  | DECBIN |
| <b>M3</b> | <b>Move Bytes</b>                           |         |        |
|           | MOFF  | MVCR    | REBF   |
|           | MVER  | MVCV    | WRBF   |
|           | MVC   | CRTMM   | WFMCRT |
|           |   | MMCRT   |        |
| <b>P</b>  | <b>Partition Load and Exit</b>              |         |        |
|           | LOAD  |         | EXIT   |
| <b>T1</b> | <b>Table Control for Shared Tables</b>      |         |        |
|           | TLCK  |         | TUNLCK |

Figure 4-2 (Part 2 of 3). Instruction Types

|           |                             |        |        |
|-----------|-----------------------------|--------|--------|
| <b>T2</b> | <b>Table Read and Write</b> |        |        |
|           | TBRD                        | TBDL   | TBWE   |
|           | TBRL                        | TBIN   | TBWT   |
| <b>T3</b> | <b>Table Search</b>         |        |        |
|           | TBBS                        | TBFH   | TBFX   |
|           |                             | TBFL   |        |
| <b>X</b>  | <b>Miscellaneous</b>        |        |        |
|           | CLC                         | SETON  | SYSLCK |
|           | DUP                         | SETOFF | SYSUNL |
|           | GSKK                        | SON    | TRANS  |
|           | INXEQ                       | SOFF   | TRT    |
|           | RXORW                       | SRAT   |        |

Figure 4-2 (Part 3 of 3). Instruction Types

## ASSEMBLY TIME ARITHMETIC

For instruction operands that require a constant or a label, you can use an arithmetic expression, the ADDR function, or the LENG function in your source program. You can also change the declared length of an area, or change the data type of an area. The assembler makes the specified calculations or changes and places the result in the object code instruction.

### Arithmetic Expressions

An arithmetic expression may be specified for an instruction operand that allows a constant or a label of a data area. An expression consisting of *label ± constant* may replace any storage label. An expression consisting of *label-label* may replace any constant. This applies only to the executable instruction, and not to the control statements. The only control statement that may specify an expression is the .EQUATE statement.

An arithmetic expression specified in a source instruction is composed of two or more terms separated by arithmetic operators. A term may be a constant or a data label. Do not use labels that are defined for decimal registers. The label does not have to be defined before you use it in the expression.

Arithmetic operators for add (+) and subtract (-) may be used. Multiplication and division are not allowed. The arithmetic operations are performed from left to right, upon the constant and the address generated by the label. Do not add a label to a label. The result must be dimensionally correct for both a valid label and a constant.

If you use an expression for an operand that allows a length specification, do not put a constant between the label and the length.

*Right:* R15=LABL(10)+6;

*Wrong:* R15=LABL+6 (10);

The number of terms must not total over eight for the entire instruction. This total includes any terms to the left of the equal sign or within parentheses.

5 Terms:            1 2    3 4 5  
BR26(4)+=LABL+2-A

8 Terms:            1 2 3 4    5 6    7 8  
MVC(BR1(4),BR2,ADDR(LBL1-LBL2)+10)

An example of using assembly time arithmetic is to find the contents of a particular byte within a buffer. The following code declares a 50-byte buffer labeled INPUT and uses assembly time arithmetic to find the contents of the 24th byte.

.DC LABEL=SCAN TYPE=DEC; set up a decimal register.

.DC LABEL=INPUT LEN=50; set up the 50 byte storage buffer.

SCAN = INPUT(1)+23; copy contents of 24th byte into SCAN.

### The ADDR Function

The ADDR function can be used as an instruction operand in place of any constant to specify the address of a register, storage area, instruction, or control block. This is a convenient way to load a base address register. The format of the ADDR function is:

$$\text{ADDR} \left( \begin{array}{l} \text{BRn} \\ \text{Rn} \\ \text{label} \end{array} \right) [\pm\text{offset}]$$

If an offset is included for an ADDR specification in an expression, the ADDR specification must be the first term in the expression. If the label of a data table is used within the ADDR function, the value returned is the index into the system table where the address of the data table is stored. The address or index returned by the ADDR function is not checked for validity.

*Examples:*    The following instruction places the declared address of BUFR into BR65.

BR65=ADDR(BUFR);

The following instruction places the address of the fifth byte of BUFR into BR65.

BR65=ADDR(BUFR+4);

## The LENG Function

The LENG function can be used as an instruction operand to specify the declared length of an area. The format of the LENG function is:

```
LENG(label)
```

An offset cannot be included within the parentheses; however, an expression can follow the right parentheses to change the length:

```
LENG(label)+4
```

*Example:* The following code moves 'GHIJKL' into the area labeled LABL1.

```
.DC LABEL=LABL1 INIT='ABCDEF' ;  
.DC LABEL=LABL2 INIT='GHIJKLMNO' ;  
.  
.  
.  
BR30 = ADDR(LABL1) ;  
BR31 = ADDR(LABL2) ;  
MVC(BR30, BR31, LENG(LABL1)) ;
```

## Changing a Declared Length

In an instruction, you can follow any declared label with a constant enclosed in parentheses. The value of the constant supercedes the declared length of the area for that instruction. The declared length is not changed in storage.

*Example:* The following code loads 'AB' into BR55.

```
.DC LABEL= LABL1 INIT='ABCDEF' ;  
.  
.  
.  
BR55 = LABL1(2) ;
```

## Changing a Data Type

Any time you use assembly time arithmetic, you can use any of the types of data described in Chapter 2 under *Data Types*. However, the result of assembly time arithmetic is always a constant or a label of a data storage area. This is important to remember when using a binary register with assembly time arithmetic. Except for a binary register, when you specify a length of 1 in an instruction:

```
label(1)
```

the first byte of the labeled area is accessed. For a binary register, when you specify a length of 1 in an instruction, the second (rightmost) byte of the binary register is accessed. However, if you use the label of a binary register with a length of 1 with an expression:

```
label(1)+0
```

the first (leftmost) byte of the binary register is accessed because the result of an expression is a data storage area rather than a binary register.

*Examples:* The following code shows what is loaded into the binary register labeled A for the different specifications.

```
.DC LABEL=A TYPE=BIN INIT='FFFF' ;  
.DC LABEL=B TYPE=BIN INIT='1234' ;  
.DC LABEL=C LEVL=2
```

```
.  
.  
.
```

```
A = C-2      ; A contains '00FF'  
A(1)+0 = B   ; A contains '34FF'  
A = B        ; A contains '1234'  
A = B(2)     ; A contains '1234'  
A = B(1)     ; A contains '0034'  
A = B(1)+0   ; A contains '0012'  
A = C        ; A contains '0012'  
A = C(2)     ; A contains '1234'  
A = C+1      ; A contains '0034'
```

## ARITHMETIC/LOGICAL INSTRUCTIONS

All arithmetic/logical operations are performed upon binary or decimal registers. The 2-byte binary registers contain unsigned binary notation. The 16-byte decimal registers contain character or signed numeric data represented in EBCDIC notation. Decimal register data is negative if the zone portion of the rightmost byte contains a hex D.

### Binary Register Arithmetic/Logical

Full 2-byte binary registers (BR0-BR127) may be specified in any binary arithmetic/logical operation. One byte of a binary register (BRn(1)) may be used as a storage reference for operands that allow a storage reference. When one byte of a binary register is specified, only the rightmost byte is used, and the leftmost byte is padded with zeros. A double register (BRn(4)) may be specified as the result register in double precision add and subtract operations. When a double register is specified, the register referred to, and the next sequential register, are used in the operation. See *Binary Registers* under *Indicators and Registers* in Chapter 1 for more information about binary registers.

When the length of the operand is less than the length of the result register or result register pair, the operand is expanded to the left with zeros to the length of the result.

The format for binary arithmetic/logical instructions is as follows. There must be at least one blank between the operator and the operand.

| Result    | Operator   | Operand   |
|-----------|--|-----------|
| BRn [(4)] | $\left. \begin{array}{l} *= \\ /= \\ += \\ -= \\ \&= \\ V= \\ X= \end{array} \right\}$ | BRn [(1)] |
| BRn       |  | constant  |
|           |  | storage   |

where:

*Result* must specify a binary register (BRn). Except for AND, OR, and Exclusive-OR, it may be a binary double register (BRn [4]). When the instruction is executed, the contents of the result register are operated upon, and the resulting data replaces the original contents of the result register. The register specified for result can be specified for the operand.

*Operand* may specify a binary register, a binary half register, a constant, or a storage location:

*+= (Add)*

| Result   | Operator | Operand   |
|----------|----------|---|
| BRa[(4)] | +=       | $\left\{ \begin{array}{l} BRb[(1)] \\ \text{constant} \\ \text{storage} \end{array} \right\}$ |

The data specified by the operand is logically added to the data in the result register, and resulting data replaces the original contents of the result register.

When a binary double register is used as the result register, the operand is added to the contents of the specified double register, and the result is right-adjusted into the double register.

*Example:*

|                |      |                          |              |                          |
|----------------|------|--------------------------|--------------|--------------------------|
| <b>Before:</b> | BR27 | <u>00000000 00000011</u> | BR28         | <u>11111111 11111110</u> |
|                |      |                          | BR27(4) += 2 |                          |
| <b>After:</b>  | BR27 | <u>00000000 00000100</u> | BR28         | <u>00000000 00000000</u> |

*-= (Subtract)*

| Result   | Operator | Operand                             |
|----------|----------|-------------------------------------|
| BRa[(4)] | -=       | { BRb[(1)]<br>constant<br>storage } |

The data specified by the operand is logically subtracted from the contents of the result register, and the resulting data replaces the original contents of the result register.

When a binary double register is used for the result register, the operand is subtracted from the contents of the specified register and the next sequential register.

*Example:*

|         |     |                          |     |                          |
|---------|-----|--------------------------|-----|--------------------------|
| Before: | BRX | <u>00000000 00001000</u> | BRY | <u>00000000 00001000</u> |
|         |     | BRX -= BRY               |     |                          |
| After:  | BRX | <u>00000000 00000100</u> | BRY | <u>00000000 00001000</u> |

*\*= (Multiply)*

| Result   | Operator | Operand          |
|----------|----------|------------------|
| BRa[(4)] | *=       | { BRb<br>label } |

The contents of the 2-byte labeled data area specified by the operand are logically multiplied with the contents of the result register, and the resulting data replaces the original contents of the result register.

When a binary double register is used as the result register, the operand is multiplied by the contents of the leftmost register. The resulting data is right-adjusted into the double register.

*/= (Divide)*

| Result   | Operator | Operand          |
|----------|----------|------------------|
| BRa[(4)] | /=       | { BRb<br>label } |

The data specified by the operand is logically divided into the contents of the result register, and the resulting data replaces the original contents of the result register.

When a binary double register is used for the result register, the operand is logically divided into the contents of the leftmost register, and the resulting data replaces the contents of the leftmost register. The remainder replaces the original contents of the rightmost register.

**& = (And)**

| Result | Operator | Operand  |
|--------|----------|--|
| BRa    | &=       | $\left\{ \begin{array}{l} \text{BRb}[(1)] \\ \text{constant} \\ \text{storage} \end{array} \right\}$ |

The data specified by the operand is logically ANDed with the contents of the result register, and the resulting data replaces the original contents of the result register.

*Example:*

|                |     |                   |     |                   |
|----------------|-----|-------------------|-----|-------------------|
| <b>Before:</b> | BRX | 11111111 00000000 | BRY | 11110000 11110000 |
|                |     | BRX &= BRY        |     |                   |
| <b>After:</b>  | BRX | 11110000 00000000 | BRY | 11110000 11110000 |

**V = (Or)**

| Result | Operator | Operand  |
|--------|----------|--|
| BRa    | V=       | $\left\{ \begin{array}{l} \text{BRb}[(1)] \\ \text{constant} \\ \text{storage} \end{array} \right\}$ |

The data specified by the operand is logically ORed with the contents of the result register, and the resulting data replaces the original contents of the result register.

*Example:*

|                |     |                   |     |                   |
|----------------|-----|-------------------|-----|-------------------|
| <b>Before:</b> | BRX | 11111111 00000000 | BRY | 11110000 11110000 |
|                |     | BRX V= BRY        |     |                   |
| <b>After:</b>  | BRX | 11111111 11110000 | BRY | 11110000 11110000 |

*X = (Exclusive-OR)*

| Result | Operator | Operand  |
|--------|----------|--|
| BRa    | X=       | $\left\{ \begin{array}{l} \text{BRb}[(1)] \\ \text{constant} \\ \text{storage} \end{array} \right\}$ |

The data specified by the operand is exclusively ORed with the contents of the result register, and the resulting data replaces the original contents of the result register.

*Example:*

|                |     |                          |            |                          |
|----------------|-----|--------------------------|------------|--------------------------|
| <b>Before:</b> | BRX | <u>11111111 00000000</u> | BRY        | <u>11110000 11110000</u> |
|                |     |                          | BRX X= BRY |                          |
| <b>After:</b>  | BRX | <u>00001111 11110000</u> | BRY        | <u>11110000 11110000</u> |

### Binary Register Shift/Rotate

The contents of a binary register, a labeled storage area, or a binary double register (BRn (4)), may be shifted or rotated. Shift operations move the contents of the register out of one end of the register and set the bits from which data was shifted to zero. Rotate operations move the contents of the register out of one end and into the other end of the register.

The first operand of a shift instruction cannot be an arithmetic expression; however, it may be followed by a length in parentheses:

*Right:* label(1) SR 3  
*Wrong:* label+1 SR 3

The format of all shift and rotate operations is as follows.

| Result      | Operator   | Operand |
|-------------|--|---------|
| label (len) | $\left\{ \begin{array}{l} \text{SL} \\ \text{SR} \end{array} \right\}$ | 1-16    |
| BRn         | $\left\{ \begin{array}{l} \text{RL} \\ \text{RR} \end{array} \right\}$ | 1-8     |
| BRn(1)      |  | 1-32    |
| BRn(4)      |  |         |

where:

*Result* is the register (BRn), labeled storage area, or double register (BRn(4)) that contains data to be shifted. Upon completion of the operation, the resulting shifted data is in the result. The length, whether explicit or implied, must be 1, 2, or 4.

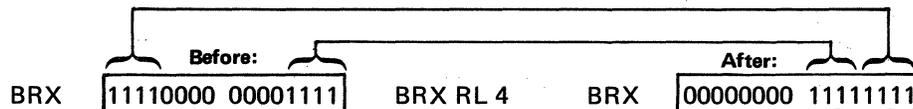
*Operand* specifies the number of bits to shift or rotate. The number must be less than or equal to number of bits contained in the result register, labeled area, or double register.

### RL (Rotate left)

| Result     | Operator | Operand |
|------------|----------|---------|
| label(len) |          |         |
| BRa        | RL       | 1-16    |
| BRa(1)     | RL       | 1-8     |
| BRa(4)     | RL       | 1-32    |

The data in the result register is rotated left the number of bits indicated by the operand. Data rotated off the high-order end is moved into the low-order end of the register.

*Example:*

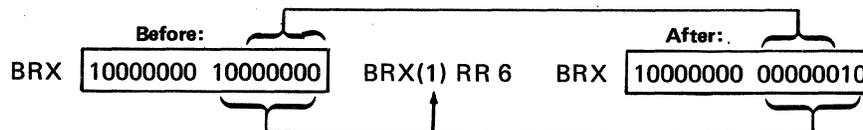


### RR (Rotate right)

| Result     | Operator | Operand |
|------------|----------|---------|
| label(len) |          |         |
| BRa        | RR       | 1-16    |
| BRa(1)     | RR       | 1-8     |
| BRa(4)     | RR       | 1-32    |

The data in the result register is rotated right the number of bits indicated by the operand. Data rotated off the low-order end of the register is moved into the high-order end.

*Example:*

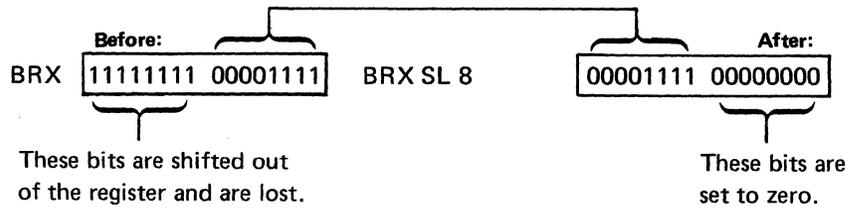


### SL (Shift left)

| Result     | Operator | Operand |
|------------|----------|---------|
| label(len) |          |         |
| BRa        | SL       | 1-16    |
| BRa(1)     | SL       | 1-8     |
| BRa(4)     | SL       | 1-32    |

The data in the result is shifted left the number of bits indicated by the operand.  
The bits from which data is shifted are set to zero.

*Example:*

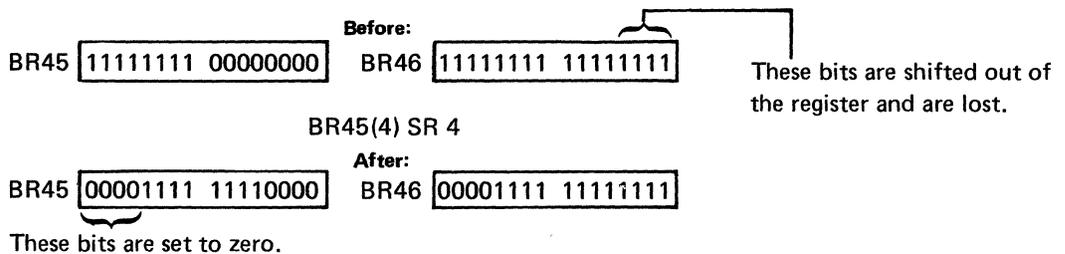


*SR (Shift right)*

| Result     | Operator | Operand |
|------------|----------|---------|
| label(len) |          |         |
| BRa        | SR       | 1-16    |
| BRa(1)     | SR       | 1-8     |
| BRa(4)     | SR       | 1-32    |

The data in the result is shifted right the number of bits indicated by the operand.  
The bits from which data was shifted are set to zero.

*Example:*



### Decimal Register Arithmetic

A decimal register (R0-R239) may be specified for any arithmetic operation. A decimal double register (Rn (32)) may be specified for the double precision multiply and divide operations. When a decimal double register is specified, the contents of the specified register and the next sequential decimal register are used in the operation. See *Decimal Registers* under *Indicators and Registers* in Chapter 1 for more information about decimal registers.

Decimal data can be positive or negative. The data is negative if the rightmost byte of the register or register pair has a hex D in the zone portion of the rightmost byte. Zones of the other register bytes are not checked to determine if the digit portions contain valid numeric data. If a register contains nonnumeric data, the operation proceeds with unpredictable results.

When a decimal register is altered, any leading blanks (hex 40) are changed to zeros (F0). When a zero results from a calculation, the zero is stored as positive (hex F0) in a decimal register.

The format of decimal arithmetic instructions is as follows. All data that is allowed for the operands is indicated.

| Result   | Operand1 | Operator | Operand2 |
|----------|----------|----------|----------|
| Rx       | =        | +        | Rx       |
|          |          | +        | Rx       |
|          |          | +        | 0-9      |
|          |          |          |          |
| Rx       | =        | -        | Rx       |
|          |          | -        | Rx       |
|          |          | -        | 0-9      |
|          |          |          |          |
| Ra[(32)] | =        | *        | Rx       |
|          |          | *        | Rx       |
|          |          | *        | 0-9      |
|          |          |          |          |
| Ra       | =        | /        | Rc       |
|          |          | /        | 1-9      |
|          |          |          |          |

where:

- Rx can represent any decimal register excluding Ra, Rb, and Rc. Each Rx can be the same as or different from any other Rx.
- Ra can represent any decimal register different from those specified by Rx, Rb, or Rc.
- Rb can represent any decimal register different from those specified by Rx, Ra, or Rc.
- Rc can represent any decimal register different from those specified by Rx, Ra, or Rb.

*Result* indicates a decimal register or, for double precision multiply, a decimal double register, into which the result of the operation is placed. If the result is positive, all zones contain a hex F; if the result is negative, the zone of the rightmost byte contains a hex D, and all other zones contain hex F.

**Operand1** specifies the data that is operated upon. A decimal register that holds operand1 may be specified. Except for multiply and divide, it may be the same decimal register that is specified as the result register. Except for divide, the contents of the register are unchanged by the operation unless it is the same register as the result register. Except for divide, operand1 may be a single-digit constant (0-9). If operand1 is a constant, operand2 must be a register.

For divide operations, operand1 must specify a decimal register or decimal double register that holds the operand1 data; it must not be the same register as the one specified for the result register. Upon completion of the operation, the remainder of the divide operation is placed into the register or double register specified for operand1. The original contents of the register are lost.

**Operand2** specifies data to operate upon the operand1 data. A decimal register that holds operand2 may be specified. If operand1 is not a constant, operand2 may be a single-digit constant. For divide, the constant may be one to nine; for all other operations the constant may be zero to nine. For add and subtract, operand2 may specify the same decimal register that is also specified as operand1. The operand2 data remains unchanged by the operation unless it is the decimal register that is also specified as the result register.

+ (Add)

| Result | Operand1 | Operator | Operand2 |
|--------|----------|----------|----------|
| Rx     | =        |          |          |
|        |          |          |          |
|        |          |          |          |
|        |          |          |          |
|        |          |          |          |

$$\left. \begin{array}{l} \text{Rx} \\ \text{0-9} \\ \text{Rx} \end{array} \right\} \begin{array}{c} + \\ + \\ + \end{array} \left. \begin{array}{l} \text{Rx} \\ \text{Rx} \\ \text{0-9} \end{array} \right\}$$

Operand2 is algebraically added to operand1, and the sum is placed in the decimal register specified as result.

If a carry results out of the high-order position of the result register, the decimal arithmetic overflow indicator (I124) is set on.

- (Subtract)

| Result | Operand1 | Operator | Operand2 |
|--------|----------|----------|----------|
| Rx     | =        |          |          |
|        |          |          |          |
|        |          |          |          |
|        |          |          |          |
|        |          |          |          |

$$\left. \begin{array}{l} \text{Rx} \\ \text{0-9} \\ \text{Rx} \end{array} \right\} \begin{array}{c} - \\ - \\ - \end{array} \left. \begin{array}{l} \text{Rx} \\ \text{Rx} \\ \text{0-9} \end{array} \right\}$$

Operand2 is algebraically subtracted from operand1 and the remainder is placed into the decimal register specified as result.

/ (Divide)

| Result | Operand1                     | Operator | Operand2      |
|--------|------------------------------|----------|---------------|
| Ra     | = { Rb [(32)]<br>Rb [(32)] } | /        | { Rc<br>1-9 } |

The contents of the register or register pair specified by operand1 are divided by operand2. The quotient is placed into the decimal register specified as result, and the remainder is placed into the decimal register or register pair specified for operand1. The result register *must not* be the same register as specified for operand1 or operand2.

Both operand1 and operand2 are signed quantities; if they have the same sign the result is positive, and if they have opposite signs the result is negative. The remainder retains the sign of the original contents of the operand1 register.

Division by zero is invalid and causes the decimal arithmetic overflow (I124) and the divide error (I120) indicators to be set on. If an error occurs during a double precision divide, the result register is unchanged.

Before:

|       |    |    |     |    |    |    |    |
|-------|----|----|-----|----|----|----|----|
|       | 00 | 01 | ... | 12 | 13 | 14 | 15 |
| Zone  |    |    | ... | F  | F  | F  | F  |
| Digit |    |    | ... | 0  | 0  | 0  | 0  |

AVERAGE = 0

|       |  |  |     |   |   |   |   |
|-------|--|--|-----|---|---|---|---|
|       |  |  | ... | F | F | F | F |
| Zone  |  |  | ... | 0 | 0 | 3 | 2 |
| Digit |  |  | ... |   |   |   |   |

TOTAL = 32

|       |  |  |     |   |   |   |   |
|-------|--|--|-----|---|---|---|---|
|       |  |  | ... | F | F | F | D |
| Zone  |  |  | ... | 0 | 0 | 0 | 5 |
| Digit |  |  | ... |   |   |   |   |

LAST = -5

AVERAGE = TOTAL / LAST

The contents of TOTAL are divided by the contents of LAST. Because the two signs are opposite, the result register will contain a minus sign (D-zone in the units position). The register TOTAL will retain the original sign and will hold the remainder to the division. The register LAST remains unchanged.

After:

|       |    |    |     |    |    |    |    |
|-------|----|----|-----|----|----|----|----|
|       | 00 | 01 | ... | 12 | 13 | 14 | 15 |
| Zone  |    |    | ... | F  | F  | F  | D  |
| Digit |    |    | ... | 0  | 0  | 0  | 6  |

AVERAGE = -6

|       |  |  |     |   |   |   |   |
|-------|--|--|-----|---|---|---|---|
|       |  |  | ... | F | F | F | F |
| Zone  |  |  | ... | 0 | 0 | 0 | 2 |
| Digit |  |  | ... |   |   |   |   |

TOTAL = 2

|       |  |  |     |   |   |   |   |
|-------|--|--|-----|---|---|---|---|
|       |  |  | ... | F | F | F | D |
| Zone  |  |  | ... | 0 | 0 | 0 | 5 |
| Digit |  |  | ... |   |   |   |   |

LAST = -5

\* (Multiply)

| Result    | Operand1              | Operator | Operand2            |
|-----------|-----------------------|----------|---------------------|
| Ra [(32)] | = { Rx<br>0-9<br>Rx } | *        | { Rx<br>Rx<br>0-9 } |

Operand1 is algebraically multiplied by operand2 and the product is stored in the decimal register(s) specified as result. Neither operand1 nor operand2 may be specified as the result register.

If a carry results out of the high order position of result, both the decimal arithmetic overflow (I124) and the multiply overflow (I123) indicators are set on. The low-order product is preserved.

*Examples:*

WAGES = HRS \* RATE

The contents of the decimal registers addressed by the labels HRS and RATE are multiplied and the product is placed into the decimal register WAGES. The contents of HRS and RATE remain unchanged by the operation.

TOTAL(32) = QUANTY \* ITEM\$

The contents of the decimal registers addressed by the labels QUANTY and ITEM\$ are multiplied and the product is placed in the decimal register pair addressed by the label TOTAL. The contents of QUANTY and ITEM\$ remain unchanged by the operation.

**Decimal Register Shift**

The contents of a decimal register can be shifted to the left or right. Any data shifted out of one end of a decimal register is lost. Bytes from which data is shifted are filled with zeros or blanks, depending upon the particular shift instruction.

The format of decimal register shift instructions is as follows.

| Result | Operand1   | Operator  | Operand2   |
|--------|--|---|--|
| Ra =   | $\left\{ \begin{array}{c} Ra \\ Rb \end{array} \right\}$ | $\left\{ \begin{array}{c} SL \\ SLS \\ SR \\ SRS \\ SRR \end{array} \right\}$ | $\left\{ \begin{array}{c} Rc \\ 1-15 \end{array} \right\}$ |

where:

*Result* indicates a decimal register into which the shifted results of the operation is placed. If the zone of the rightmost byte contains a hex D, the contents of the register are negative.

*Operand1* indicates a decimal register which holds the data to be shifted. The contents of this register are not changed unless it is the same register as the one specified for result.

*Operand2* indicates the number of bytes to shift the operand1 data. Operand2 may be a constant (1-15) or a decimal register that contains the shift count.

If operand2 specifies a register, a single-digit shift count (decimal 1-9) is determined from the digit portion of the low-order position of the register. A shift count of hex 1-F may be accomplished by setting the digit portion of the low-order position of the decimal register to a binary number. This is done by overlaying the rightmost 2 bytes of the decimal register with a binary register, then loading the binary register with the hex shift count. This can be done only in decimal registers that are located within the area of overlapped storage that can also contain binary registers (R0-R14). See the .DC control statement description in Chapter 3 for information about overlaying registers.

**SL Shift Left, Blank Fill**

| Result | Operand1   | Operator | Operand2   |
|--------|--|----------|--|
| Ra =   | $\left\{ \begin{array}{c} \text{Ra} \\ \text{Rb} \end{array} \right\}$ | SL       | $\left\{ \begin{array}{c} \text{Rc} \\ \text{1-15} \end{array} \right\}$ |

The bytes of the operand1 register are shifted left the number of bytes indicated by operand2, and the shifted result is placed into the result register.

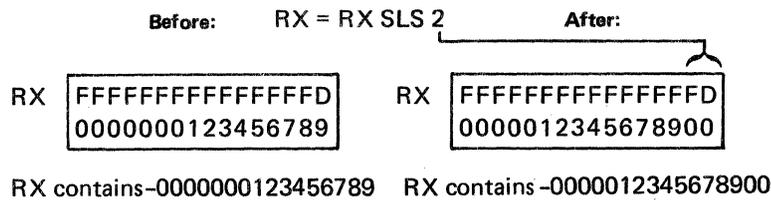
The low-order positions of the shift result contain the blank character (hex 40) for the number of positions shifted. If a negative number is shifted left, the D-zone is shifted left out of the units position. Therefore, the register no longer contains a negative number.

**SLS Shift Left, Zero Fill and Retain Sign**

| Result | Operand1   | Operator | Operand2   |
|--------|--|----------|--|
| Ra =   | $\left\{ \begin{array}{c} \text{Ra} \\ \text{Rb} \end{array} \right\}$ | SLS      | $\left\{ \begin{array}{c} \text{Rc} \\ \text{1-15} \end{array} \right\}$ |

The bytes of the operand1 register are shifted left the number of bytes indicated by operand2, and the shifted result is placed into the result register. The low-order bytes of the shifted result contain zeros (hex F0) for the number of positions shifted. If a negative number is shifted left, the units position of the result register retains the D-zone.

*Example:*



**SR (Shift 1 to Blank Decimal Register)**

| Result | Operand1 | Operator | Operand2 |
|--------|----------|----------|----------|
| Rn =   | 1        | SR       | 1        |

The bytes are shifted as though operand1 were a decimal register with decimal 1 in the rightmost byte, and bytes 0 to 14 were blanks. The rightmost byte is shifted out of the register so the register contains only blanks. These blanks replace the contents of the result register.

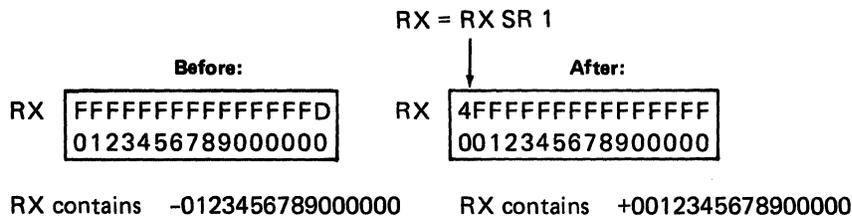
This is the quickest way to blank a decimal register.

**SR (Shift Right, Pad Blank)**

| Result | Operand1     | Operator | Operand2       |
|--------|--------------|----------|----------------|
| Ra =   | { Ra<br>RB } | SR       | { Rc<br>1-15 } |

The bytes of the operand1 register are shifted right the number of bytes indicated by operand2, and the shifted result is placed into the result register. The high-order bytes of the shifted result contain blanks (hex 40) for the number of characters shifted. If a negative number is shifted right, the D-zone is shifted out of the register and the register contents are no longer negative.

**Example:**

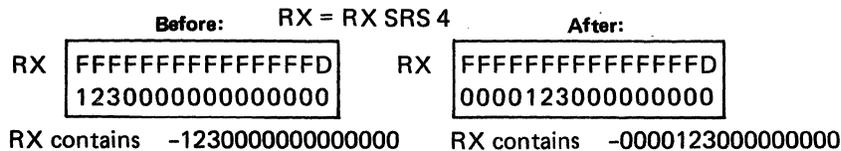


**SRS (Shift Right and Retain Sign)**

| Result | Operand1     | Operator | Operand2       |
|--------|--------------|----------|----------------|
| Ra     | { Ra<br>Rb } | SRS      | { Rc<br>1-15 } |

The bytes of the operand1 register are shifted right the number of bytes indicated by operand2, and the shifted result is placed into the result register. The high-order bytes of the result register contain zero (hex F0) for the number of bytes shifted. Any blanks present are shifted without change. If the unshifted contents of the operand1 register contained a negative value, the result register contains a hex D in the zone portion of the rightmost byte. All other zones remain unchanged.

**Example:**

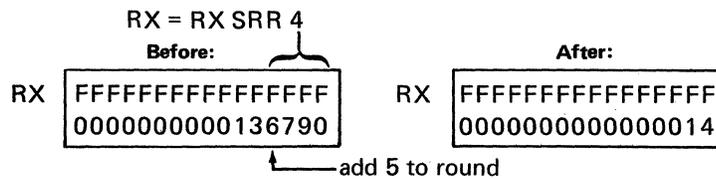


### SRR Shift Right, Retain Sign, and Round

|               |  |                 |   |
|---------------|--|-----------------|---|
| <b>Result</b> | <b>Operand1</b>  | <b>Operator</b> | <b>Operand2</b>   |
| Ra =          | $\left\{ \begin{array}{c} \text{Ra} \\ \text{Rb} \end{array} \right\}$ | SRR             | $\left\{ \begin{array}{c} \text{Rc} \\ 1-15 \end{array} \right\}$ |

The bytes of the operand1 register are shifted right the number bytes indicated by operand2, and the shifted result is placed into the result register. The high-order bytes of the shifted result contains zeros (hex F0) for the number of positions shifted, and the units position of the shifted number retains the zone of the original contents of the operand1 register. The result is rounded by adding 5 of like sign to the last byte shifted out of the right end of the result register (the shift-count-minus-1 position).

*Example:*



### Decimal Register Zone Modification

The zone portions of decimal register bytes can be modified. One or more zones can be changed to a specified hex character.

The format for the zone modification instruction is as follows:

| Operator | Result | Operand   | Offset   | Length |
|----------|--------|---|----------|--------|
| ZONE     | (Ra,   | $\left\{ \begin{array}{c} \text{Rb} \\ \text{X'I} \end{array} \right\}$ | [X'0-F', | 1-16]) |

where:

*Result* indicates the decimal register (Ra) that has zones to modify.

*Operand* specifies the hex character, or the decimal register that contains the hex character, that replaces the zones of the specified bytes in the result register. If a decimal register is specified, the character in the zone portion of the rightmost byte is used.

*Offset* specifies the number of bytes (0-15), from the leftmost byte of the register, to be skipped before zone modification begins. Offset defaults to zero.

*Length* indicates the number of bytes (1-16) to modify. Length defaults to 1 byte.



where:

*Operand* specifies the address of the instruction to which the branch is made.

A binary register (BRn) may be included in the operand. This register may hold the address of the instruction to which to branch, or it may act as an index register for an indexed branch or a branch through a label table. BR0 must not be used.

### *NOP (Null Operation)*

**Mnemonic**

NOP

The NOP instruction is used as a space filler and performs a branch to the next sequential instruction. There are no operands for this instruction.

### *GOTO (Unconditional Branch)*

| Mnemonic | Operand   |
|----------|---|
| GOTO     | $\left\{ \begin{array}{l} \text{label} \\ \text{BRn} \\ \text{BRn,label} \\ \text{BRn,disp} \end{array} \right\}$ |

The labeled statement at the operand label is the next statement executed, unless the optional binary register is included with the label. In that case, the contents of BRn are added to the address of the instruction with the indicated label, and the result is taken as the address of the next instruction to execute.

If a binary register is coded with no label, a branch is made to the address in the binary register. This address should be on a 4-byte boundary.

No registers, indicators, or storage areas are changed.

**Note:** The resulting address is not checked for validity unless indexing is used.

### *GOTAB (Unconditional Branch Through Table)*

| Mnemonic | Operand   |
|----------|-----------|
| GOTAB    | Brn,label |

The operand *label* specifies the label of a label table defined in a .LABTAB control statement. The contents of BRn are taken as the index into the label table specified. The address stored at that entry position is taken as the address of the next instruction to be executed. The first table index is 0.

No registers or storage areas are changed.

## Subroutine Call and Return

A subroutine call consists of a mnemonic and an operand. You can make an indexed call or a call through a label table. You can call a subroutine that is stored within the current partition, or you can call a common function routine that is stored in the common functions area. If you call any routines that are stored in the common functions area, you must specify the routine labels in an .XTRN control statement in your source program.

A subroutine return consists of a mnemonic. Optionally, you can include an operand to make an indexed return. There are special return operations for keyboard/display external status subroutines. These external status returns are discussed under *Key Entry Instructions* in this chapter.

The format for subroutine calls and returns is as follows:

| Mnemonic | Operand   |
|----------|---|
| CALL     | $\left\{ \begin{array}{l} \text{label} \\ \text{BRn} \\ \text{BRn,label} \\ \text{BRn,disp} \end{array} \right\}$ |
| RETURN   | [(BRn)]   |
| CALLTB   | BRn,label   |

where:

*Operand* specifies the subroutine to call or the return address, as follows:

| Entry | Description  |
|-------|--|
| label | The label of a subroutine.   |
| BRn   | A binary register used to hold an index or address.                                |
| disp  | May be specified with a binary register for a call to a base displacement address. |

### *CALL (Subroutine Call)*

| Mnemonic | Operand   |
|----------|---|
| CALL     | $\left\{ \begin{array}{l} \text{label} \\ \text{BRn,label} \\ \text{BRn,disp} \\ \text{BRn} \end{array} \right\}$ |

When a call instruction is encountered, the address of the next sequential instruction is stored in the subroutine stack at the address specified by BR18. BR18 is then incremented by 2.

A call is made to the instruction specified by the operand. If a label is specified without the optional binary register, the call is made to the instruction at the label specified.

If a label is specified with a binary register, the contents of the register are added to the address of the specified label, and the call is made to the resulting address.

If a displacement is specified with a binary register, the displacement is added to the address contained in the binary register, and the call is made to the resulting address.

If a binary register is specified with no label and no displacement, the call is taken to the address in the binary register. This address should be on a 4-byte boundary.

After the call, the instructions of the called subroutine are executed sequentially until the subroutine is terminated.

#### *RETURN (Subroutine Return)*

| <b>Mnemonic</b> | <b>Operand</b> |
|-----------------|----------------|
| RETURN          | [(BRn)]        |

When a return statement is encountered in a subroutine, the subroutine is terminated and BR18 is decremented by two. If no operand is included in the instruction, the next statement to be executed is the statement at the address in the partition stack location pointed to by BR18.

If a binary register is included in the instruction, the contents of the binary register, which must be a multiple of four, are added to the address in the partition stack location pointed to by BR18. The result is the address of the next statement to be executed. If the resulting address is not on a 4-byte boundary, a program check error (hex 03) results. If the resulting address is outside the partition, or if the addition results in a carry, a program check (hex 01) occurs.

#### *Example, Normal Call:*

```
COMP1:  TALY = TALY + 3 ; Subroutine start
      .
      .
      .
COMPX:  RETURN          ; Subroutine end
      .
      .
      .
SUB1:  CALL COMP1      ; Call subroutine COMP1
ADDC:  TALY = XY + X
```

When the instruction SUB1 executes, the absolute address of ADDC is placed in the partition stack at the address specified by BR18, and a branch is made to the subroutine at the label COMP1. The statements following COMP1 are executed until the RETURN at statement COMPX is encountered. At this time the address of ADDC is taken from the partition stack at the address specified by BR18, and the next statement to be executed is ADDC.

*Example, Indexed Call:*

```
BR28 = X'08'
CALL BR28,RTN6
RTN6: X = X + Y; assume start RTN6 at address X'0670'
      .
      .
      .
RTN7: X = Y; assume start RTN7 at address X'0678'
```

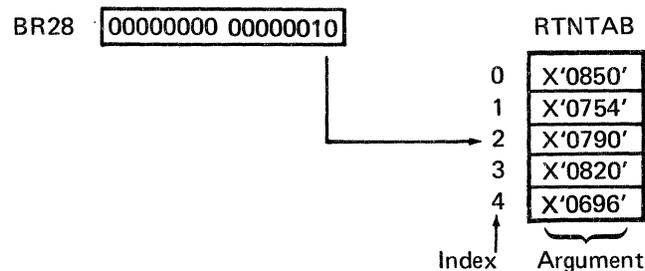
When the instruction CALL BR28, RTN6 is executed, the branch is made to RTN7.

*CALLTB (Subroutine Call Through Table)*

| Mnemonic | Operand    |
|----------|------------|
| CALLTB   | BRn, label |

The label specified is the label of a label table defined in a .LABTAB control statement. The contents of the binary register (Brn) are used as an index into the table. The index is the entry number, not the byte displacement. The address at the table entry indicated by the binary register is the address of the next statement to execute. If BRn contains 0, the first entry position is used.

*Example:*



The instruction CALLTB BR28,RTNTAB would call the subroutine whose first instruction begins at address hex 0790.

## Full Conditional Branch on Test

A full conditional branch tests an operand against a specified condition. If the operand meets the condition, a branch is made to a specified branch label. The branch label can indicate an instruction at any location in the current partition.

The format of a full conditional branch instruction is as follows. All types of data allowed for the operand and condition is indicated for each mnemonic.

| Mnemonic | Operand  | Condition   | Branch     |
|----------|----------|---|------------|
| IF       | BRn      | $\left\{ \begin{array}{c} \text{IS} \\ \text{NOT} \end{array} \right\}$ 0   | GOTO label |
| IF       | Rn       | $\left\{ \begin{array}{c} \text{IS} \\ \text{NOT} \end{array} \right\}$ $\left\{ \begin{array}{c} 0 \\ - \\ \text{AN} \\ \text{CK} \\ \text{SN} \end{array} \right\}$ | GOTO label |
| IF       | fmt      | $\left\{ \begin{array}{c} \text{IS} \\ \text{NOT} \end{array} \right\}$ FMT   | GOTO label |
| IFDSI    | dsin,dsn | $\left\{ \begin{array}{c} \text{IS} \\ \text{NOT} \end{array} \right\}$ ON  | GOTO label |
| IFI      | In       | $\left\{ \begin{array}{c} \text{IS} \\ \text{NOT} \end{array} \right\}$ ON  | GOTO label |
| IFIR     | In       | $\left\{ \begin{array}{c} \text{IS} \\ \text{NOT} \end{array} \right\}$ ON  | GOTO label |

where:

*Operand* specifies the data to be tested.

*Condition* includes the keyword IS or NOT (but not both), and the condition keyword. The condition keywords are explained in the individual operation descriptions.

*Branch* includes the mnemonic GOTO, and the label of the statement to which a branch is made if the operand meets the specified condition.

### IF BRn 0 (Test Binary Register for Zero)

| Mnemonic | Operand | Condition   | Branch     |
|----------|---------|---|------------|
| IF       | BRn     | $\left\{ \begin{array}{c} \text{IS} \\ \text{NOT} \end{array} \right\}$ 0 | GOTO label |

If the condition specifies IS, and the register specified for operand contains all zeros, a branch is made to the label. Otherwise, the next sequential statement is executed.

If the condition specifies NOT, and the register specified for operand contains some value other than zero, a branch is made to the label. If the register contains zero, the next sequential statement is executed.

A 1-byte half register is not allowed in this operation.

No registers, storage areas, or indicators are changed.

*IF Rn (Test Decimal Register)*

| Mnemonic | Operand | Condition   | Branch  |
|----------|---------|---|---|
| IF       | Rn      | $\left\{ \begin{array}{l} \text{IS} \\ \text{NOT} \end{array} \right\}$ | $\left\{ \begin{array}{l} 0 \\ - \\ \text{AN} \\ \text{CK} \\ \text{SN} \end{array} \right\}$ |
|          |         |   | GOTO label  |

The contents of the decimal register specified as the operand are tested against the specified condition. Decimal register bytes are tested from left to right.

When the condition is met, a branch is made to the instruction specified by the label. When the condition is not met, the next sequential instruction is executed. (Note that the keyword IS or NOT is part of the condition. Therefore, for the keyword NOT, when the remainder of the condition is met, the next sequential instruction is executed; and when the remainder of the condition is not met, a branch is made to the instruction specified by the label.)

The decimal register test conditions include:

| Entry | Description   |
|-------|---|
| 0     | The condition is met if the decimal register bytes contain only blanks or zeros.  |
| -     | The condition is met if the units position (rightmost byte) of the decimal register contains a hex D in the zone portion.   |
| AN    | As for SN, except the zone of the rightmost byte must contain hex F unless the register contains all blanks.  |
| CK    | The condition is met if the decimal register bytes contain data that includes a self-check digit, and the self-check digit is verified as correct by the self-check algorithm.  |
| SN    | The condition is met if the decimal register bytes contain all blanks. It is also met if the decimal register bytes contain leading blanks followed by one or more valid numbers (hex F0-F9) and the rightmost byte contains either a hex F or a hex D in the zone portion. It is also met if the decimal register bytes contain all valid numbers (hex F0-F9) and the rightmost byte contains either a hex F or a hex D in the zone portion. |

*Example:*

RX 

|                  |
|------------------|
| FFFFFFFFFFFFFFFF |
| 0000000000000170 |

      RY 

|                   |
|-------------------|
| FFFFFFFFFFFFFFFFD |
| 0000000000000355  |

      RZ 

|                  |
|------------------|
| 44444444444444FF |
| 0000000000000078 |

- The instruction      IF RX IS 0 GOTO LOOPX      results in no branch.
- The instruction      IF RY NOT - GOTO LOOPX      results in no branch.
- The instruction      IF RY NOT AN GOTO LOOPX      results in a branch to LOOPX.
- The instruction      IF RZ IS SN GOTO LOOPX      results in a branch to LOOPX.

The instruction IF RZ NOT CK GOTO ERR5 results in a branch to ERR5 if the self-check digit is incorrect when verified against the self-check algorithm. If the self-check digit is correct, the next instruction statement is executed.

*IF fmt (Test Format Number)*

| Mnemonic | Operand | Condition   | Branch     |
|----------|---------|---|------------|
| IF       | fmt     | $\left\{ \begin{array}{l} \text{IS} \\ \text{NOT} \end{array} \right\}$ FMT | GOTO label |

The format specified by the operand is tested against the format used in the last I/O instruction. It may be used to determine what format was used in a data directed READ.

For the operand, specify the LABEL parameter from the .FMTST control statement that set up the desired format.

If the condition specifies IS, and the format specified for operand is the same as the format used in the last I/O instruction, a branch is made to the instruction at label. Otherwise, the next sequential instruction is executed.

If the condition specifies NOT, and the format specified for operand is not the same as the format used in the last I/O instruction, a branch is made to the instruction at label.

*Example:*

IF FORMAT6 IS FMT GOTO LOOPX

This instruction results in a branch to LOOPX if the last format used for an I/O operation was FORMAT6.

### *IFDSI (Test Data Set Indicator)*

| Mnemonic | Operand  | Condition  | Branch     |
|----------|----------|--|------------|
| IFDSI    | dsin,dsn | $\left\{ \begin{array}{l} \text{IS} \\ \text{NOT} \end{array} \right\}$ ON | GOTO label |

The data set status indicator specified by the operand is tested against the specified condition. The operand entry includes the data set status indicator number (dsin), a comma, and the data set number (dsn). The entry for the data set number is the DSN parameter of the .DATASET control statement that defined the data set you wish to use. The entry for data set status indicator is a number from 0-15. If you specify a number from 0 to 7, it represents bits 0-7 of the status byte (byte hex 00) of the data set IOB. If you specify a number from 8 to 15, it represents bits 0-7 of the data set flag byte (byte hex 13) of the data set IOB. See the *Functions Reference Manual* for a description of these bits.

#### *Example:*

IFDSI 8,3 IS ON GOTO LOOP

This instruction results in a branch if bit 0 of the IOB data set flag byte (byte hex 13) of data set 3 contains a B'1'; this bit is 1 if the data set is open.

### *IFI (Test Indicator)*

| Mnemonic | Operand | Condition  | Branch     |
|----------|---------|--|------------|
| IFI      | In      | $\left\{ \begin{array}{l} \text{IS} \\ \text{NOT} \end{array} \right\}$ ON | GOTO label |

The indicator specified by the operand is tested against the condition IS ON or NOT ON. If the indicator meets the condition, a branch is made to label. Otherwise, the next sequential statement is executed.

#### *Examples:*

IFI I96 NOT ON GOTO LOOP

This instruction results in a branch if I96 contains B'0'; it results in no branch if I96 contains B'1'.

IFI I77 IS ON GOTO LOOP

This instruction results in a branch if I77 contains B'1'; it results in no branch if I77 contains B'0'.

**IFIR (Test Indicator and Reset)**

| Mnemonic | Operand | Condition        | Branch     |
|----------|---------|------------------|------------|
| IFIR     | In      | { IS<br>NOT } ON | GOTO label |

The indicator specified by the operand is tested against the condition IS ON or NOT ON. If the indicator meets the condition, a branch is made to label. The indicator is set to off (B'0') after the test is completed, regardless of the result of the test.

**Examples:**

I97 = B'1' I98 = B'0'

IFIR I197 IS ON GOTO LOOP

This instruction results in a branch, and I97 is reset to B'0'.

IFIR I98 NOT ON GOTO LOOP

This instruction results in a branch, and I98 remains B'0'.

**Short Conditional Branch on Relational Compare**

A short conditional branch operation compares two operands for a specified relation condition. Except for the IFD instruction, both the zone and digit portions of the bytes are compared. Therefore, a hex 40 (blank) will not equal a hex F0 (zero). If the result of the relational compare is true, a branch is made to the instruction specified by the label. If the result is not true, the next sequential statement is executed. The label must specify an instruction within +128 or -127 instructions from the short branch instruction.

The format of a short branch instruction is as follows. All types of data that are allowed for the mnemonics and operands are indicated.

| Mnemonic      | Operand1 | Operator                             | Operand2 | Branch     |
|---------------|----------|--------------------------------------|----------|------------|
| { IF<br>IFD } | BRa      | { EQ<br>GE<br>GT<br>LE<br>LT<br>NE } | BRb      | GOTO label |
|               | Ra       |                                      | { Rb }   | GOTO label |
|               | BRn      |                                      | constant | GOTO label |
|               | IFH      |                                      |          |            |

where:

*Operand1* specifies the register which contains the data to be compared.

*Operator* must be one of the following:

| Operator | Meaning   |
|----------|---|
| EQ       | Branch<br>if operand1 is equal to operand2.                 |
| GE       | Branch<br>if operand1 is equal to or greater than operand2. |
| GT       | Branch<br>if operand1 is greater than operand2.             |
| LE       | Branch<br>if operand1 is less than or equal to operand2.    |
| LT       | Branch<br>if operand1 is less than operand2.                |
| NE       | Branch<br>if operand1 is not equal to operand2.             |

*Operand2* specifies the register or constant which will be compared against operand1.

*Branch* is always the mnemonic GOTO and a label. The label specifies an instruction, which must be within +128 or -127 instructions from the branch instruction. A branch is made to this instruction if the specified relational condition is true.

*IF BRa (Binary Register Relational Compare)*

| Mnemonic | Operand1 | Operator  | Operand2 | Branch     |
|----------|----------|---|----------|------------|
| IF       | BRa      | $\left. \begin{array}{c} \text{EQ} \\ \text{GE} \\ \text{GT} \\ \text{LE} \\ \text{LT} \\ \text{NE} \end{array} \right\}$ | Bb       | GOTO label |

The contents of the operand1 register are compared to the contents of the operand2 register. A branch is made to the label if the relational compare is true. Otherwise, the next sequential statement is executed.

No registers, data areas, or indicators are changed.

*Examples:*

BREG 11110011 11110100      CNTR 11110000 11110000  
 BREG contains X'F3F4'      CNTR contains X'F0F0'

The instruction IF BREG EQ CNTR GOTO LOOP4 results in no branch.

The instruction IF BREG GE CNTR GOTO LOOP4 results in a branch to LOOP4.

*IF Ra (Decimal Register Relational Compare)*

| Mnemonic | Operand1 | Operator   | Operand2  | Branch     |
|----------|----------|--|---|------------|
| IF       | Ra       | <div style="display: inline-block; vertical-align: middle;"> <div style="border-left: 1px solid black; border-right: 1px solid black; padding: 0 5px;">                     EQ<br/>                     GE<br/>                     GT<br/>                     LE<br/>                     LT<br/>                     NE                 </div> </div> | <div style="display: inline-block; vertical-align: middle;"> <div style="border-left: 1px solid black; border-right: 1px solid black; padding: 0 5px;">                     { Rb }<br/>                     { 0-9 }                 </div> </div> | GOTO label |

The contents of the operand1 register are compared against operand2 for the condition specified by the relational operator. If the result of the relational compare is true, a branch is made to the instruction specified by the label. Otherwise, the next sequential statement is executed.

If operand2 is a register (Rb), the corresponding positions of the operand1 and operand2 registers are compared byte for byte, starting at the high order position of each register. The standard EBCDIC collating sequence is used.

If operand2 is a constant (0-9), the contents of the operand1 register are compared against fifteen leading blanks with the specified constant in the rightmost byte of the register. The leading blanks cannot be successfully compared with zeros.

No registers, data areas, or indicators are changed.

**Note:** You should use the IFD instruction when comparing decimal data. IFD does not compare the zone portions of the bytes. This avoids problems resulting from comparing blanks (hex 40) to zeros (hex F0).

*Examples:*

REG1 ... FFFF  
1111      REG2 ... FFFF  
2222      REG3 ... FFFF  
0007

The instruction IF REG1 NE REG2 GOTO LOOPX results in a branch to LOOPX.

The instruction IF REG3 EQ7 GOTO LOOPX results in no branch. This is because the leading zeros of REG3 are compared against leading blanks associated with the constant 7, and hex F0 does not equal hex 40.

*IFD Rn (Decimal Register Relational Compare to Decimal)*

| Mnemonic | Operand1 | Operator   | Operand2  | Branch     |
|----------|----------|--|---|------------|
| IFD      | Ra       | $\left\{ \begin{array}{c} \text{EQ} \\ \text{GE} \\ \text{GT} \\ \text{LE} \\ \text{LT} \\ \text{NE} \end{array} \right\}$ | $\left\{ \begin{array}{c} \text{0-9} \\ \text{Rb} \end{array} \right\}$ | GOTO label |

The contents of the operand1 register are algebraically compared to operand2 for the condition specified by the relational operator. If the result of the comparison is true, a branch is made to the instruction specified by the label. Otherwise the next sequential statement is executed.

If the zone portion of the rightmost byte of a decimal register contains hex D, the contents of the register are negative. If it is not hex D, the contents of the register are positive.

If operand2 is another decimal register (Rb), the digit portion of each corresponding byte of the two registers and the zone portion of the rightmost byte of each register are compared.

If operand2 is a constant, the contents of the operand1 decimal register are compared against 15 leading blanks and the specified single-digit constant. The constant is assumed to be a positive value.

*IFH BRn (Binary Register Relational Compare to Immediate Data)*

| Mnemonic | Operand1 | Operator   | Operand2         | Branch     |
|----------|----------|--|------------------|------------|
| IFH      | BRa      | $\left\{ \begin{array}{c} \text{EQ} \\ \text{GE} \\ \text{GT} \\ \text{LE} \\ \text{LT} \\ \text{NE} \end{array} \right\}$ | 0-255<br>X'0-FF' | GOTO label |

The rightmost byte of the binary register specified as operand1 is compared against the binary representation of the constant specified by operand2. A branch is made to the instruction specified by the label if the result of the comparison is true. Otherwise, the next sequential instruction is executed.

No registers, indicators, or data areas are changed.

*Examples:*

Only this byte is compared.

BREG[00000000 11110011]

The instruction IFH BREG EQ X'F3' GOTO LOOPX results in a branch to LOOPX.

The instruction IFH BREG GE X'F4' GOTO LOOPX results in no branch.

### Skip on Constant Compare

The contents of a specified byte are compared with a 1-byte constant.

The format for a constant compare skip instruction is as follows:

| Mnemonic | Test                            | Condition | Operand  | Result |
|----------|---------------------------------|-----------|----------|--------|
| IFC      | [offset,] label<br>[offset,] Rn | IS<br>NOT | constant | SKIP   |
| IFB      | [disp] (BRn)                    | IS        | constant | SKIP   |

where:

*Test* specifies the byte that holds data to compare with the constant, as follows.

*[offset,] label* specifies the label of a storage area. If the storage area is longer than one byte, you may specify an optional offset. The offset specifies the offset from the first byte of the storage area where the byte is located. Offset defaults to 0, which specifies the first (leftmost) byte of the data area.

*[offset,] Rn* specifies a decimal register (Rn) and the offset (0-15) from the leftmost byte of the register where the byte is located. For example, 7,RX indicates byte 7 of the decimal register labeled RX. The offset is optional and defaults to zero, which specifies the leftmost byte of the register. If the test register is a decimal register, the operand must be a decimal register.

*[disp] (BRn)* specifies the base displacement address of the storage byte. The binary register holds the base address. The displacement (0-255) is optional and defaults to zero. A base displacement address is valid only for the IFB mnemonic.

**Note:** Do not specify length for this base displacement address. Do not precede the binary register with a comma, as is required for other base displacement addresses.

*Condition* specifies how the test byte is compared, as follows.

| Entry | Description  |
|-------|--|
| IS    | Specifies that a skip takes place if the contents of the test byte equal the specified constant.   |
| NOT   | Specifies that a skip takes place if the contents of the test byte do not equal the specified constant. This is valid only for the IFC mnemonic. |

*Operand* specifies a 1-byte constant, as described near the beginning of this chapter under *Constant Specifications*.

*IFC (Skip on Constant Compare)*

| Mnemonic | Test                                    | Condition         | Operand  | Result |
|----------|---|-------------------|----------|--------|
| IFC      | { [offset,] label }<br>{ [offset,] Rn } | { IS }<br>{ NOT } | constant | SKIP   |

The contents of the byte specified by test are compared against the specified constant. If IS is coded, and the test byte equals the constant, the next sequential statement is skipped. If NOT is coded, and the contents of the test byte do not equal the constant, the next sequential instruction is skipped. Otherwise the next sequential instruction is executed.

*Example:* The following code initializes a decimal register to contain fifteen bytes of zeros and one byte of hex FF. Hex data is right-justified into a decimal register, so byte 15 of the decimal register contains the hex constant.

|     | 00 | 01 | 02 | 03 | 04 | 05 | 06 | 07 | 08 | 09 | 10 | 11 | 12 | 13 | 14 | 15 |
|-----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| REG | F  | F  | F  | F  | F  | F  | F  | F  | F  | F  | F  | F  | F  | F  | F  | F  |
|     | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | F  |

```

.DC LABEL = REG TYPE = DEC INIT = X'FF';
STMT1: IFC 15,REG NOT X'FF' SKIP ;STMT2 is not skipped
STMT2: IFC 14,REG IS X'F0' SKIP ;STMT3 is skipped
STMT3: IFC 15,REG IS X'FF' SKIP ;not executed
STMT4: IFC 1,REG NOT X'FF' SKIP ;STMT5 is skipped
STMT5: REG = 0
STMT6: IFC 15,REG NOT X'F0' SKIP ;STMT7 is skipped

```

### *IFB (Base Displacement Skip on Constant Compare)*

| Mnemonic | Test         | Condition | Operand  | Result |
|----------|--------------|-----------|----------|--------|
| IFB      | [disp] (BRn) | IS        | constant | SKIP   |

The displacement is added to the base address contained in the binary register. The contents of the byte at the resulting address are compared to the 1-byte constant specified by operand. If the contents of the byte are equal to the constant, the next sequential instruction is skipped. If the contents of the byte are not equal to the constant, the next sequential instruction is executed. It is invalid to specify NOT for the condition.

### **Skip on Bit Mask**

The contents of a 1-byte base displacement address are masked with a 1-byte constant. The format for a bit mask instruction is as follows:

| Mnemonic | Test         | Condition     | Operand | Result |
|----------|--------------|---------------|---------|--------|
| IFB      | [disp] (BRn) | { ON<br>OFF } | X'11'   | SKIP   |

where:

*Test* specifies a base displacement address of a storage byte. The binary register holds the base address. The displacement (0-255) is optional and defaults to zero.

**Note:** Do not specify length for this base displacement address. Do not precede the binary register with a comma as is required for other base displacement addresses.

*Condition* indicates how the bits in the test byte that are specified by the mask are tested.

| Entry | Description   |
|-------|---|
| ON    | Specifies that a skip takes place if any of the masked bits are on (B'1').  |
| OFF   | Specifies that a skip takes place if any of the masked bits are off (B'0'). |

*Operand* specifies the 1-byte mask that specifies which bits of the test byte are tested.

*Result* is always the mnemonic SKIP.

The contents of the byte at the base displacement address are masked with the 1-byte operand. The bits that correspond with the bits in the mask that are on (B'1') are tested against the specified condition. If one or more of these bits meets the specified condition (ON or OFF), the skip results. Otherwise, the next sequential instruction is executed.

## Skip on AND/Exclusive-OR Mask

This mask test can perform logical operations upon one byte of a binary register.

The format for an AND, Exclusive-OR mask test instruction is as follows:

| Test         | Mnemonic | Mask1 | Operand | Mask2 | Result |      |
|--------------|----------|-------|---------|-------|--------|------|
| IFHI<br>IFLO | BRn      | AND   | X'II'   | IS    | X'II'  | SKIP |

where:

*Test* specifies the byte of the binary register to operate upon. It includes a binary register (BRn) and either IFHI to specify the high-order byte of the binary register, or IFLO to specify the low-order byte.

*Mask 1, Mask 2* are 1-byte constants to mask with the test byte. See *Constant Specifications* near the beginning of this chapter for the forms of the constants that may be used.

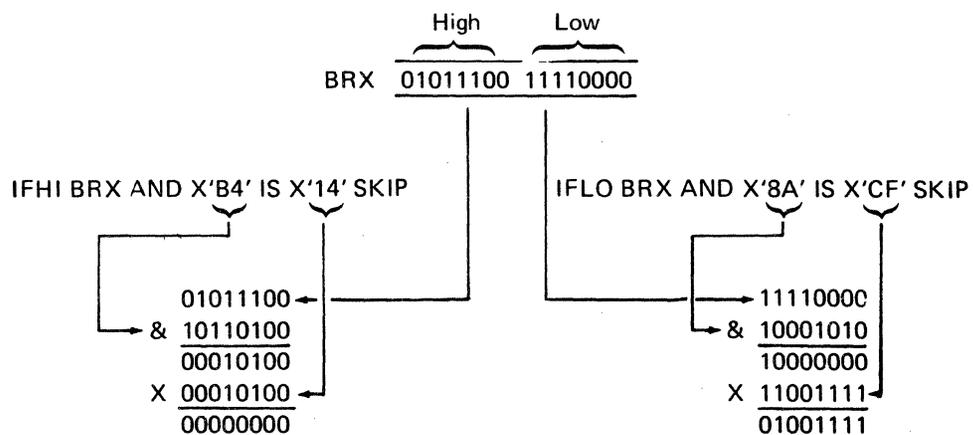
*Operand* is always IS.

*Result* is always the mnemonic SKIP.

The byte that is accessed depends on the test entry. Coding IFHI BRn indicates the high order byte of the specified binary register (BRn). Coding IFLO BRn indicates the low-order byte of the specified binary register (BRn). The original contents of the binary register remain unchanged by the operation.

The contents of the indicated byte are logically ANDed with mask 1, and the result is exclusively-ORed with mask 2. If the result of the operation is zero, the next sequential instruction is skipped. Otherwise the next sequential instruction is executed.

*Examples:*



This instruction results in a skip.

This instruction does not result in a skip.

## Loop Control

Loop control can be performed with the following two instructions, which decrement or increment a binary register, then test the register contents against zero or a specified limit.

The format of the loop control instructions is as follows:

| Mnemonic   | Test | Condition | Limit | Operand      |
|------------|------|-----------|-------|--------------|
| SKIP WHILE | BRa  | LE        | BRb   | [STEP 0-255] |
| DECR       | BRn  |           |       | GOTO label   |

where:

*Test* indicates the binary register that is decremented or incremented.

*Condition* is specified only for SKIP WHILE, and is always LE.

*Limit* is specified only for SKIP WHILE, and indicates the limit that is compared with the test register.

*Operand* is always specified for DECR, to indicate a branch label. It is optional for SKIP WHILE, to indicate the value of the increment to add to the test register. If it is not specified, the register is incremented by one.

**SKIP WHILE (Increment Binary Register and Skip if Not Limit)**

| Mnemonic   | Test | Condition | Limit | Operand      |
|------------|------|-----------|-------|--------------|
| SKIP WHILE | BRa  | LE        | BRb   | [STEP 0–255] |

When this instruction is executed, an increment is added to the contents of the test binary register (BRa), and the sum replaces the contents of the test binary register (BRa). The new contents of the test register (BRa) are then compared with the contents of the limit register (BRb). If the contents of the test register (BRa) are lower than or equal to the limit register (BRb), the next sequential instruction is skipped.

If the optional increment operand (STEP n) is coded, the contents of the test register are incremented by n, where n is 0-255. If STEP n is not coded, the test register is incremented by one.

*Example:* The following code sums the integers from 1 to 10 using SKIP WHILE as the loop control.

```
.DC LABEL=SUM TYPE=BIN INIT=0; set up register for sum
*
.DC LABEL=NUM TYPE=BIN INIT=0; set up register for integers
*
.DC LABEL=LMT TYPE=BIN INIT=10; set loop limit to 10
*
LOOP1: SKIP WHILE NUM LE LMT STEP 1;
* Add 1 to NUM and test for 10.
* When NUM is greater than LMT, EXIT1 EXECUTES.
EXIT1: GOTO OUT:
SUM += NUM ; add next integer.
GOTO LOOP1 ; continue loop.
OUT: NOP ; NUM = 11 and SUM = 55.
```

**DECR (Decrement Binary Register and Branch If Not 0)**

| <b>Mnemonic</b> | <b>Operand</b> | <b>Branch</b> |
|-----------------|----------------|---------------|
| DECR            | BRn            | GOTO label    |

The contents of the binary register are first decremented by 1 every time this instruction is executed, and then the decremented result is tested against 0. If the register contents are not 0, a branch is made to the statement at label.

*Example:*

```
X#1:  BREG = B'11'; initialize BRn to loop 3 times  
LOOP:
```

```
ENDL:  DECR BREG TO LOOP; decrement BRn by one,  
* if BRn is not zero after decrement, branch to LOOP.  
X#2:  REGX = REGX + 1
```

The statement labeled X#1 sets BREG to 3, then the statements from LOOP to ENDL are executed 3 times. The third time ENDL is executed, BREG will contain 0 and the statement labeled X#2 will be the next statement executed.

**COMMUNICATIONS INSTRUCTIONS**

The IBM 5280 communications facilities include a communications access method that provides an interface between your IBM 5280 assembler application program and the host system. The communications access method can support either BSC (binary synchronous communication) or SDLC (synchronous data link control). SDLC is provided through SNA (Systems Network Architecture) and is referred to in this manual as SNA. The following communications instructions and the .COMM control statement described in Chapter 3 are used with the IBM 5280 communications access method. See the *IBM 5280 Communications Reference Manual*, SC34-0247, for more information about the communications access method.

Most of the communications instructions are used for both the BSC and the SNA versions of the communications access method. Any instruction or instruction parameter that is unique to either the BSC or SNA versions is so indicated. The format for the communications instructions is as follows:

| Mnemonic | Data Set | Format             | Operand1  | Operand2      | Operand3 |
|----------|----------|--------------------|-----------|---------------|----------|
| TINIT    | (1-15)   |                    |           |               |          |
| TCTL     | (1-15)   |                    | [,X'IIII' | , {<br>N<br>O | ,D)      |
| TOPEN    | (1-15)   |                    |           |               |          |
| TWAIT    | (1-15)   |                    |           |               |          |
| TCLOZ    | (1-15)   |                    |           |               |          |
| TTERM    | (1-15)   |                    |           |               |          |
| TREAD    | (1-15)   | [, {<br>*<br>label | , -       | , {<br>N<br>O | , -)     |
| TWRT     | (1-15)   | [,label            | ,F        | , {<br>N<br>O | ,B)      |

where:

*Data set* specifies the data set to access. Enter the DSN parameter of the .COMM control statement that defined the data set.

*Format* indicates edit formatting you want during an input or output operation. Format may be omitted (retain the comma) if no edit formatting is desired. If you want formatting, specify one of the following:

| Entry | Description |
|-------|-------------|
|-------|-------------|

|       |  |
|-------|--|
| label | The label of the edit format you wish to use. Enter the LABEL parameter of the .FMTST control statement that started the edit format definition. |
|-------|--|

|   |   |
|---|---|
| * | May be specified for a TREAD instruction to indicate data directed formatting. <i>Data Directed Formatting</i> , in Chapter 2, and the .FMTST control statement writeup describe this method of format selection. |
|---|---|

*Operand1* depends upon the particular instruction and is explained with the operand description for each instruction.

*Operand2* specifies overlapped (O) or nonoverlapped (N) I/O. It defaults to nonoverlapped. See *Overlapped I/O* in Chapter 1 for more information.

*Operand3* depends upon the particular instruction and is explained with the operand description for each instruction.

**TINIT (Initialize Communications Session)**

**Mnemonic      Data Set**

TINIT          (1-15)

For BSC and for SNA, this must be the first communications instruction in your application program. The TINIT operation must be completed before an I/O instruction can be executed. For BSC, this instruction must be followed by a TOPEN instruction.

For BSC, the TINIT instruction establishes the linkage between the COMM IOB and your program, which is executing in a partition, and the communications access method, which is concurrently executing in another partition. When this instruction is executed, it prepares to establish the line connection between the IBM 5280 and the host system.

For SNA, this instruction initializes the network by establishing the data link and attaching a session. If log-on is required by the host system, the communications IOB must point to a logical buffer that contains the log-on data when the TINIT instruction is issued. See the *IBM 5280 Communications Reference Manual* for information about communications sessions. Multiple sessions may be established.

**TOPEN (Open Communications Data Set)**

**Mnemonic      Data Set**

TOPEN          (1-15)

This instruction opens the specified communications data set IOB for transmitting or receiving records, or for inquiry. This instruction must be issued before any TREAD, TWRT, or TCTL instructions are issued. The TOPEN instruction must always be followed, at any later point in your program, with a TCLOZ instruction for the same data set.

**TCTL (Communications Control)**

**Mnemonic      Data Set      Operand1      Operand2      Operand3**

TCTL          (1-15          [,X'IIII'          , { N }          ,D])

Operand1 specifies a hex code for one of the following control operations:

| Hex Code | Operations Valid for BSC   |
|----------|--|
| 0100     | Write status   |
| 0300     | Transmit EOT (The EOT function is used only to initiate the premature termination of a transmit or receive operation.) |
| 0400     | Transmit RVI (The RVI function turns the line around.)   |
| 0500     | Transmit header (SOH-heading-STX)  |
| 0600     | Transmit header (SOH-heading-ETB)  |
| 0700     | Transmit header (SOH-heading-ITB)  |
| 0800     | Transmit header (SOH-heading-STX-ETX)  |
| 0001     | Set compression (Expand blank-compressed data that is received.)   |
| 0002     | Reset compression (Do not expand compressed data.)   |
| 0003     | Set transparent mode (Transmit in transparent mode.)   |
| 0004     | Reset transparent mode (Do not transmit in transparent mode.)  |
| 0005     | Set trace (on)   |
| 0006     | Reset trace (off)  |

#### Operations Valid for SNA

|      |  |
|------|--|
| 0001 | Cancel                                       |
| 0002 | Chase  |
| 0003 | LU status                                    |
| 0004 | Request shutdown                             |
| 0005 | Positive response                            |
| 0006 | Negative response                            |
| 0007 | Transmit signal command (SIGNAL) to the host |
| 0008 | Shutdown complete                            |

If D is specified for operand3, the diagnostic flag is set in the IOB, and the operation is performed in diagnostic mode.

#### *TWAIT (Wait for I/O Completion)*

| Mnemonic | Data Set |
|----------|----------|
|----------|----------|

|       |        |
|-------|--------|
| TWAIT | (1-15) |
|-------|--------|

For BSC and for SNA, this instruction ensures that any transmit or receive operation is completed before the next sequential instruction is executed. This is generally used in conjunction with overlapped I/O to prevent loss of data. See *Overlapped I/O* in Chapter 1 for more information.

*TCLOZ (Close Communications Data Set)*

**Mnemonic      Data Set**

TCLOZ          (1-15)

This instruction closes the specified communications data set IOB, and signifies the end of the communications operations. The IOB that is opened (using TOPEN) must be closed (using TCLOZ) before the same IOB is opened for another operation.

For BSC, if the preceding operation was TWRT (FINAL), a null record and EOT will be sent. If the preceding operation was TWRT, only a null record will be sent.

*TTERM (Terminate Communications Session)*

**Mnemonic      Data Set**

TTERM          (1-15)

For BSC and SNA, this instruction terminates the logical connection between your application program and the communications access method. If a switched line is being used, the line connection with the host is also terminated.

The communications access method remains in the background partition, available for use by other application programs.

Any IOB for which a TINIT has been issued must be terminated before another TINIT is issued for any IOB.

*TREAD (Receive Communications Record)*

| Mnemonic | Data Set | Format              | Operand1 | Operand2   | Operand3 |
|----------|----------|---------------------|----------|------------|----------|
| TREAD    | (1-15    | [, { *<br>label } ] | .-       | { N<br>O } | .-])     |

When the TREAD instruction is executed, data is received from the host system into the I/O buffer of the data set specified by the data set number. If an edit format is indicated with a format label or the asterisk (for data directed formatting), the data is edited and moved from the I/O buffer to the registers as specified in the appropriate edit format. If no edit format is specified in the instruction, the data remains in the I/O buffer at the end of the operation. If 0 is specified for operand2, successive instructions are executed concurrently with the TREAD instruction. If operand2 specifies N or is omitted, the TREAD operation must be completed before any successive instructions are executed.

For SNA and for BSC, if operand1 is a minus sign, it indicates that the status is to be received. If operand1 is omitted, it indicates that data is to be received.

For SNA and for BSC, if operand3 is a minus sign, it indicates that an entire block of data is to be received. If operand3 is omitted, the next logical record is received.

### TWRT (Transmit Communications Record)

| Mnemonic | Data Set | Format  | Operand1 | Operand2  | Operand3 |
|----------|----------|---------|----------|---|----------|
| TWRT     | (1-15    | [,label | ,F       | $\left. \begin{array}{c} N \\ O \end{array} \right\}$ | ,B))     |

When the TWRT instruction is executed, a record is transmitted to the host system from the data set specified by the data set number. If the instruction specifies an edit format, data is moved from the registers specified by the format into the I/O buffer, and editing is performed as specified in the format. If no format is specified, it is assumed that the record is already in the I/O buffer. The record is transmitted from the I/O buffer. If operand2 is O, successive instructions are executed concurrently with the TWRT instruction. If operand2 is N or is omitted, the TWRT instruction must be completed before successive instructions are executed.

For SNA, if operand1 is F, it indicates that this is the final record to be transmitted in an interactive application. Otherwise operand1 is omitted. Retain the comma.

For BSC, if operand1 is F, it indicates that this is the last write (that is, the last record to be transmitted) in a series of write operations. This causes CAM to wait for a positive acknowledgement from the host that the entire transmission has been successfully received by the host. An EOT is then transmitted. There is no data associated with write final or BSC, and TCLOZ should be the next instruction.

For SNA and for BSC, if operand3 is B, it indicates that the I/O buffer is to be filled with blanks at the start of the operation. If operand3 is omitted, the I/O buffer is not changed at the start of the operation. This operand may be specified only for a formatted TWRT operation.

*Example:* The following code sets up a BSC data set with the .COMM statement. It then initializes a BSC communications session, transmits one data record, receives one data record, and terminates the session.

```
.START ENTRY=INIT;
.DC LABEL=LBUFR LEN=80; logical buffer for transmit data set
.DC LABEL=DATAR LEN=132; logical buffer for receive data set
.COMM CAM=BSC DSN=1 RECL=80 LBUF=LBUFR ELAB=ERR
TYPE=SW; set up a write sequential transmit data set IOB
.COMM CAM=BSC DSN=2 RECL=132 LBUF=DATAR ELAB=ERR
TYPE=SR; set up a read sequential receive data set IOB
.INIT TINIT(1); initialize session with host
TOPEN(1); open transmit data set IOB
* Code to move data
* to the logical buffer LBUFR
* must be included here.
TWRT(1); transmit a record from LBUFR
TWRT(1,,F); send EOT
TCLOZ(1); close transmit data set IOB
TTERM(1);
TINIT(2);
TOPEN(2); open receive data set IOB
TREAD(2); receive a record into DATR
```

- \* Code to move data from
- \* the logical buffer DATAR
- \* must be included here.

```

TREAD(2);received end of data condition
TCLOZ(2);close receive data set IOB
TTERM(1);terminate session with host
ERR:      ;error routine.
.END;
```

## DISKETTE INSTRUCTIONS

The diskette instructions include operations to control, read/write, and search a data set. The operations are performed upon data set records or, for label update data sets, upon data set labels. Data set labels on diskette index cylinders describe each data set that is stored on the diskette. Information contained on the data set label includes the data set name, the exchange type, and the diskette addresses of the BOE (beginning of extent), EOD (end of data), and EOE (end of extent) for that data set. The data set label will be described in detail in the *Functions Reference Manual*.

Records may be organized into sequential or key indexed data sets. The records are moved between the diskette and a physical buffer in main storage. Logical records may optionally be moved between the physical buffer and a logical record buffer for blocking and deblocking records, or pointer I/O can be used to block and deblock records. See *Diskette Data Management* in Chapter 2 for a description of the data set organizations and buffers.

During the execution of a diskette operation, the IBM 5280 maintains a record counter to keep track of the record number, relative to the first data set record, of the logical record currently being processed.

### Control Operations

In order to share the diskette devices with the different data sets, the IBM 5280 maintains an IOB chain for each drive. An IOB chain begins with a pointer, in the system control area, which holds the address of the first IOB that uses the device. This IOB in turn holds the address of the next IOB that uses the device. When the device finishes the work described by one IOB, it goes to the next IOB on the IOB chain. An IOB address is placed on the IOB chain whenever a data set is opened with the OPEN operation. When the data set has finished using the device, it may be closed with the CLOZ operation, which removes the IOB address from the IOB chain.

Besides the open and close operations, data set control includes instructions to allocate a data set and to initialize diskettes.

The format for the data set control instructions is as follows:

| Mnemonic | Data Set | Operands   |
|----------|----------|--|
| ALLOC    | (1-15    | „BRn)  |
| OPEN     | (1-15    | [„BRn ])   |
| CLOZ     | (1-15    | [, { R<br>E<br>D<br>N }, { W<br>P }, { V<br>C<br>* }, { C<br>L<br>* }, BRn]) |
| WAIT     | {(1-15)} | ,BRn)  |
| INIT     | (1-15    |  |
| POSN     | (1-15    | { BOE<br>EOD<br>CURR<br>LAST }, [, { O<br>N } ])                             |

where:

*Data Set* specifies the data set number. This is the DSN parameter of the .DATASET control statement that set up the IOB for this data set. If data set is specified as hex 0 in the WAIT instruction, it specifies the keyboard rather than a data set. Do not specify 0 for any other control operation.

*Operands* are explained for each instruction in the operation description.

#### ALLOC (Allocate a Data Set)

| Mnemonic | Data Set | Operand |
|----------|----------|---------|
| ALLOC    | (1-15    | „BRn)   |

This instruction is always executed nonoverlapped. When the ALLOC operation is executed, the data set is allocated in the physical space following the last valid data set existing on the diskette, provided sufficient extent and label space exists. A data set cannot be allocated between existing data sets and always originates on a physical track/sector boundary.

The data set HDR1 label is placed in the first deleted HDR1 label space. If there are no deleted HDR1 label spaces, the allocation cannot take place, and an external status (3229) is presented. The HDR1 information is taken from the data set IOB and from a parameter string you prepare and place into storage. The binary register (BRn) in the ALLOC instruction contains the address of the *fifth* byte of the parameter string. Two commas must precede the binary register. The format of the parameter string is as follows.

| Byte | Meaning   |
|------|---|
| 1    | Data set exchange type. Enter the number that corresponds to the appropriate exchange type:<br><br>X'00' = basic exchange<br>X'01' = H exchange<br>X'02' = I exchange (This is the type normally used)<br>X'03' = E exchange, unblocked and unspanned<br>X'04' = E exchange, blocked and unspanned<br>X'05' = E exchange, blocked and spanned |
| 2-4  | The number of logical records to allocate. Enter 0 to allocate the maximum number of records that can be placed on the remaining diskette space.  |
| 5    | The first of up to 14 characters of an optional owner identification, required for allocating on a secure diskette. The address stored in the binary register always points to the first byte of this owner ID. If the owner ID is omitted, the address points to the end blank.  |
| end  | The last byte in the parameter string must be a blank (hex 40) if the owner ID is omitted or if it is less than 14 characters in length. If the owner ID is 14 characters in length, the blank is omitted.  |

**Note:** This parameter string can also be used to open a data set on a secure diskette; the OPEN instruction does not use the bytes before the fifth byte.

The information that is taken from the data set IOB is as follows:

| Parameter                    | Explanation  |
|------------------------------|--|
| Data set name (NAME)         | The data set name is mandatory for allocating a diskette data set.   |
| Logical record length (RECL) | If this optional .DATASET parameter is omitted, the record length is set to equal the block size.  |
| Block size (BSIZ)            | Except for blocked and spanned data sets, the block size must equal, or be a multiple of, the logical record length. For blocked and spanned data sets, BSIZ is an optional parameter; if specified it must equal sector size, and if omitted the IBM 5280 sets it to sector size. |
| Delete character (DFLG)      | F or E and I exchange data sets, this character is checked, during allocation, for a valid printable character. Valid delete characters are A-Z, 0-9, or one of the following symbols:<br><br>.,-/,%#@\$&  |

During the allocation operation, the data set organization byte of the HDR1 label is set to blank (hex 40) for basic and H exchange data sets. It is set to D for E and I exchange data sets. It is invalid to allocate a label update data set with the ALLOC instruction. A device opened for label update may not have any open data sets. Similarly, a data set may not be opened on a device already open for label update.

Upon completion of the ALLOC operation, the allocated data set is also opened. The HDR1 label is placed into the first 128 bytes of the physical buffer. The opcode byte in the data set IOB is replaced with hex 00. Upon completion of the ALLOC; or if there is an external status for (1) insufficient physical buffer size (3430), (2) two physical buffers specified with unequal sizes (3435); or if a 3730 warning message is presented the minimum number of 128-byte blocks required for sufficient buffer size space is placed into hex 78 of the data set IOB. If any other external status occurs, this number is not placed into the IOB.

*OPEN (Open a Data Set)*

| <b>Mnemonic</b> | <b>Data Set</b> | <b>Operand</b> |
|-----------------|-----------------|----------------|
| OPEN            | (1-15           | [,,BRn])       |

The OPEN operation is always executed in nonoverlapped mode. When the OPEN is executed, the IBM 5280 searches the diskette index for the HDR1 label of the specified data set. The search begins at the position indicated by the data set IOB at hex 30 and 31. The assembler initializes this value to zero, which causes the search to begin with the first HDR1 label. If you know where your data set HDR1 label is stored, you can place the position number (the first HDR1 label is at number 8) in the data set IOB to save time when the data set is opened. If the position number in the IOB is greater than 8, the search begins with that position and continues until the desired data set HDR1 label is found or until the last label is searched. No labels preceding the position number in the IOB are searched. If the desired HDR1 label is not found, or if the position number is greater than the number of HDR1 labels existing on the diskette, an external status (3215) is presented. When the IBM 5280 finds the label, it places the label number in bytes hex 30 and 31 of the data set IOB. On subsequent OPENS, the search begins with this number unless you reset these bytes to zero.

If the data set is stored on a secure diskette, you must include the binary register (BRn) in the OPEN instruction. The binary register contains the address of an owner identification, as described for the ALLOC instruction. If you include the binary register and the diskette is not a secure diskette, the binary register is ignored. Two commas must precede the binary register.

When the desired HDR1 label is found, the expiration date is checked. If the date is all blanks, the data set is expired. If it is all 9s, the data set is not expired. If the HDR1 expiration data contains a year value less than 50, the year is assumed to be in the 21st century. If a system date is available and exceeds the HDR1 label date, the data set is expired. An unexpired data set cannot be opened as an erase data set.

The physical buffer is checked to ensure adequate buffer size. External status (3430) is presented if there is not enough space to read the HDR1 labels. For SW or SU and erase data sets using pointer I/O, and for KU and KR data sets, external status (3430) is presented if the buffer is large enough to read HDR1 labels but too small to process the data set. For all other data sets, external status (3730 or 3799) is presented if there is insufficient buffer space to process the data set. If no external status occurs, or if external status 3730, 3799, or 3705 is presented, the OPEN completes and byte hex 78 of the data set IOB contains the minimum number of 128-byte blocks required to process the data set. If external status 3430 or 3435 is presented, the OPEN does not complete but the minimum number of 128-byte blocks required to process the data set is placed into byte hex 78 of the data set IOB. If any other external status is presented, the OPEN does not complete and the IOB does not contain any information about the required buffer size.

If the block size and logical record length are not specified in the data set IOB, the HDR1 values are used. If they are specified in the IOB, they are compared with the HDR1 values and an error results if they are not the same.

For an erase data set, the block size and record length on the HDR1 label are updated to the values in the IOB if the IOB values are not zero. An empty data set is created by setting the end of data value equal to the beginning of extent. If no logical record length is specified, it is assumed to be the same as the block size.

When the open completes, the address of the data set IOB is placed into the IOB chain for the device. The open-flag in the data set IOB is set to indicate that the data set is open. The HDR1 label is placed into the first 128 bytes of the physical buffer (in PB1 if two physical buffers are used). For a label update data set, however, the VOL1 label is placed into the buffer in place of the HDR1 label.

The OPEN instruction must be used to open a data set during program execution, or to reopen a data set. If you reopen a data set, the data set pointers are reset to the beginning of the data set and the parameters specified in the .DATASET control statement are used to supply the open information.

#### *CLOZ (Close a Data Set)*

| Mnemonic | Data Set | Operands   |
|----------|----------|--|
| CLOZ     | (1-15    | [, $\begin{Bmatrix} R \\ E \\ D \\ N \end{Bmatrix}$ , $\begin{Bmatrix} P \\ W \end{Bmatrix}$ , $\begin{Bmatrix} V \\ C \\ * \end{Bmatrix}$ , $\begin{Bmatrix} C \\ L \\ * \end{Bmatrix}$ , BRn]) |

A CLOZ operation removes the data set IOB address from the IOB chain and resets the open-flag in the data set IOB. The CLOZ operation is always executed in nonoverlapped mode. If any records have been added to the data set, the HDR1 label is updated as appropriate. Any functions specified in the operand fields are performed. When the CLOZ operation is completed, even if external status is presented, the opcode in the data set IOB is reset to zero.

The operand fields indicate close options. You may specify one option in each of five fields, or you may leave any of the fields blank. If you omit a field that is to the left of a specified field, retain the comma for the omitted field. If you omit all the fields, a normal close as described above is performed, with no additional functions. The operand fields are as follows.

*Field 1:* Close functions option. If omitted, a normal close is performed. You may not specify R, E, or D for a shared or unexpired data set.

| <b>Entry</b> | <b>Description</b>  |
|--------------|---|
| R            | Release; the EOE is replaced with the EOD-1 value to free unused extent space.  |
| E            | Erase; the EOD is replaced with the BOE value to create a new output-only data set.   |
| D            | Delete; the data set label is marked as deleted.  |
| N            | No label update; the IOB address is removed from the IOB chain and the IOB open-flag is reset, but the HDR1 label is not updated. |

*Field 2:* Write-protect option. If omitted, the contents of the write-protect position in the HDR1 label remain unchanged.

| <b>Entry</b> | <b>Description</b>   |
|--------------|--|
| P            | Protect; a P is placed into the write-protect position so the data set does not accept any write operations. |
| W            | Write; clears the P from the HDR1 label so the data set can accept writes.                                   |

*Field 3:* Verify/copy option. If omitted, the contents of the verify/copy position on the HDR1 label remains unchanged.

| <b>Entry</b> | <b>Description</b>   |
|--------------|--|
| V            | Verified; a V is placed into the verify/copy position of the HDR1 label to indicate that the data set has been verified.   |
| C            | Copy; a C is placed into the verify/copy position to indicate that the contents of the data set have been successfully transferred. Do not enter the C for a partial data set copy or for a null data set. |
| *            | Clear; the contents of the verify/copy position are replaced with a blank.   |

*Field 4:* Multivolume option. If omitted, the contents of the multivolume fields are left unchanged. Field 5 must also be omitted.

| <b>Entry</b> | <b>Description</b>   |
|--------------|--|
| C            | Continued; a C is placed into the multivolume position to indicate that the data set is continued on another diskette.                 |
| L            | Last; an L is placed into the multivolume position to indicate that this is the last diskette on which a continued data set is stored. |
| *            | Clear; the contents of the fields are replaced with blanks.  |

*Field 5:* Multivolume option sequence number.

BRn must be specified if field 4 has the C or L options specified. Otherwise BRn is ignored. The binary register contains the sequence number for this volume of a multivolume data set. The contents of this register are placed into the HDR1 label in the volume sequence position. You may specify a sequence number from 01 to 99. A value of 00 blanks the field.

If an error occurs during a CLOZ, the IOB is not removed from the chain. Overwriting the IOB by loading another program would break the chain. However, if two consecutive CLOZ commands are issued (even if they contain close options) that cause errors, the IOB address is removed from the IOB chain but the HDR1 label is not updated and no optional functions are performed.

Another use for the two consecutive CLOZ instructions is to close a data set that has been interrupted by accidental opening of the diskette door. If a data set is being processed when the diskette door is opened, the diskette drive makes a buzzing sound for five seconds and a 3151 error is reported. If no I/O operation was taking place when the door was opened, you may simply close the door and reset the error. The IBM 5280 assumes that the data set is the same one being processed before the door opened. However, if an I/O operation was in the process of moving data between the physical buffer and the diskette, the data movement is terminated but the data set is not closed; a 3251 error is reported. Two consecutive CLOZ instructions remove the IOB address from the IOB chain. You may close an unopened or undefined data set with no effect.

#### *WAIT (Wait for I/O Completion)*

| <b>Mnemonic</b> | <b>Data Set</b> |
|-----------------|-----------------|
|-----------------|-----------------|

|      |          |
|------|----------|
| WAIT | [(1-15)] |
|------|----------|

The WAIT instruction is used when you are processing with overlapped I/O. When the IBM 5280 encounters an overlapped I/O instruction, it issues the I/O operation to the appropriate device and immediately proceeds to execute subsequent instructions. If a subsequent instruction accesses data that was involved in an overlapped I/O operation, you should place a WAIT instruction immediately before that instruction. When the IBM 5280 encounters the WAIT instruction, it waits until all outstanding I/O operations for the specified data set are completed before it executes the instructions following the WAIT. It also detects any errors that occurred during the I/O.

If the WAIT instruction specifies hex 0 for the data set, the IBM 5280 waits until all outstanding keyboard I/O operations are completed. If the WAIT instruction omits the data set number, all I/O operations for all data sets must be completed before the IBM 5280 proceeds with the instructions following the WAIT.

*INIT (Initialize a Diskette)*

| Mnemonic | Data Set | Operand |
|----------|----------|---------|
| INIT     | (1-15    | ,BRn)   |

The INIT operation initializes the diskette with information from the IOB of the specified data set. The data set must have been previously opened for initialization (TYPE=INI on the .DATASET control statement). The diskette may not be shared during an initialization operation. All data on the diskette prior to the INIT is lost.

The binary register (BRn) contains the address of a parameter string you have prepared and placed into storage. The format of the parameter string is as follows.

| Byte | Bit  | Meaning  |
|------|------|--|
| 1    | 0    | Head number  |
|      | 1-7  | Track number   |
| 2    | 0-1  | 00 for type 1<br>01 for type 2<br>11 for type 2D   |
|      | 2-7  | Number minus one of 128-byte blocks that make up the sector size.  |
|      | 3-28 | Sequence of sector numbers. If byte 3 = hex FF, the track specified by byte 1 is flagged as a defective track. |

Each track is accessed as a separate data set. The physical sector IDs and records are written to the track. The data set is accessed sequentially, starting at cylinder 0, head 0, sector 1. Each track must be initialized with the INIT instruction before it can accept a write instruction. You must use instructions to write the HDR1 labels for the index area.

*POSN (Position Record Pointer)*

| Mnemonic | Data Set | Operand1                         | Operand2         |
|----------|----------|----------------------------------|------------------|
| POSN     | (1-15    | , { BOE<br>CURR<br>LAST<br>EOD } | [, { O<br>N } ]) |

When the POSN operation is executed, the current record pointer number is modified. The diskette is repositioned to the specified record, specified in the instruction as follows:

| <b>Operand1<br/>Mnemonic</b> | <b>Purpose</b>   |
|------------------------------|--|
| BOE                          | Sets the record number to 0. The first record in the data set is record 1, so a subsequent READ instruction that specifies next-record (+) reads the first record of the data set. Or a subsequent search-forward instruction begins the search with record 1. |
| CURR                         | Rereads the current record (as specified by the current record pointer) from the diskette into the physical buffer and into the logical record buffer. The data set type must allow reads.   |
| LAST                         | Sets the record number to the number of the last logical record of the data set, and reads the last record from the diskette into the physical buffer and into the logical buffer. The data set type must allow reads.   |
| EOD                          | Sets the record number to the EOD number, which is the number of the next available record space on the diskette past the last record (last record plus one). Subsequent write instructions that specify current-record extend the data set.                   |

#### *Read and Write Operations*

Records can be read according to a key, or read and written by sequential record number or relative record number. For sequential records, the system keeps track of the record number in an internal register. For operations at the relative record number, you must assign a binary register or register pair to hold the relative record number. After every read or write instruction, the system updates the internal register. You must update the assigned binary register if you are using relative record access.

The buffer management for the following read and write operations is described as for automatic logical buffering. If you use pointer I/O, the logical buffer address in the data set IOB points to the logical record within the physical record buffer. See *Diskette Data Management* in Chapter 2 for more information about pointer I/O.

The format for read and write instructions is as follows:

| Mnemonic     | Data Set | Format              | Record  | Operand1     | Operand2       |
|--------------|----------|---------------------|---|--------------|----------------|
| READ         | (1-15)   | [, { *<br>label } , | $\left\{ \begin{array}{l} Rn \\ BRn[(4)] \\ - \\ + \\ 0 \end{array} \right\}$ | , { N<br>O } | )              |
| WRT          | (1-15)   | [ ,label ,          | $\left\{ \begin{array}{l} BRn[(4)] \\ + \\ - \\ 0 \end{array} \right\}$       | , { N<br>O } | ,B)            |
| WRTS<br>WRTI | (1-15)   | [ ,label            | ,0  | , { N<br>O } | ,B)            |
| INSBLK       | (1-15)   |                     |   | „BRn         | , { O<br>N } ) |

where:

*Data Set* specifies the number of the data set to read or write. The label is the DSN parameter from the .DATASET control statement. Do not specify 0 for an I/O instruction.

*Format* specifies the label of the edit format to use. The label is the LABEL parameter of the .FMTST control statement that defined the format. If the format is omitted, you must move the data to or from the logical buffer with other instructions. No format may be specified with pointer I/O. An asterisk (\*) may be specified for READ, indicating that formatting is data directed. See *Data Directed Formatting* in Chapter 2 for more information.

**Note:** When formatting is used, operand 1 defaults to nonoverlapped I/O regardless of what is specified in the instruction.

*Record* specifies the logical record to read or write. The following specifications are allowed.

| Entry                       | Description   |
|-----------------------------|---|
| (Key) <i>Rn</i>             | A decimal register may be specified (for READ only) when using a key data set. The decimal register contains the key of the record to read.   |
| (Relative) <i>BRn [(4)]</i> | <p>When a binary register is included for a write instruction the IBM 5280 checks the .DATASET parameters to determine the data set type. If this is an SCS conversion data set, the binary register is used as a pointer to a storage area that you have declared and stored SCS control characters. See <i>Printer Instructions</i> and also <i>Appendix B</i> for a description of the SCS control characters for an SCS conversion data set.</p> <p>If this is not a write instruction to an SCS conversion data set, the binary register contains the relative record number of a record in a sequential data set. This number replaces the current record number. If necessary for the size of the number, a binary register pair may be used. The number is relative to the BOE (beginning of extent) for the data set, which is record 1. The contents of this register are not changed after the operation. If it must be incremented, you must code instructions for incrementation.</p> <p>A minus sign indicates the previous sequential non-deleted record. A previous-record specification defaults to 0 for the WRTS or WRTI instruction. A previous-record specification does not permanently alter the current record number for a write operation. Consecutive previous-record write operations process the same record. If a WRT instruction that specifies previous-record is issued at EOD, the last record of the data set is overwritten and the record pointer still points to EOD. A read-previous operation decrements the current record pointer.</p> <p>A plus sign indicates the next sequential non-deleted record. The next-record specification is the default for the READ instruction. When a read-next operation is executed, the current record number is incremented.</p> <p>The next-record specification defaults to 0 for the WRTS or WRTI instruction. If a WRT instruction specifies next-record, the current record number is not incremented. Consecutive next-record write operations write the same record. If an instruction that specifies next-record is issued at EOD, an external status is presented.</p> |

| Entry | Description  |
|-------|--|
| 0     | <p>A zero indicates the current record number. The current-record specification is the default for the write operations. If a write operation is issued at EOD, the current record number is incremented and the data set is extended. The record is written into the EOD space, and the EOD is incremented. The current record number and the EOD point to the same record space on the diskette.</p> <p>If a READ instruction specifies current-record, the current record is not incremented and the current record is re-read.</p> |

*Operand1* specifies whether the operation is executed as overlapped (O) or nonoverlapped (N) I/O. If not specified, or if formatting is to be done, it defaults to nonoverlapped. See *Overlapped I/O* in Chapter 1 for more information.

*Operand2* B may be specified if a format is included for the write instructions, but B is not allowed for READ. If B is coded, the I/O buffer is blanked at the start of the operation.

#### Access Methods Valid for Each File Type

The access methods for the file types are illustrated in Figure 4-3. The keywords, which indicate both file type and access method, are described under the TYPE parameter of the .DATASET control statement in Chapter 3.

| Type:   | SR |   |   |     | SW | SU |   |   |     | KR/N |   |   |    | KU/N |   |   |    | I |   |   |     |
|---------|----|---|---|-----|----|----|---|---|-----|------|---|---|----|------|---|---|----|---|---|---|-----|
| Record: | +  | - | O | Brn | O  | +  | - | O | Brn | +    | - | O | Rn | +    | - | O | Rn | + | - | O | Brn |
| READ    | C  | C | V | C   |    | C  | C | V | C   | C    | C | V | C  | C    | C | V | C  | C | C | V | C   |
| WRT     |    |   |   |     | U  | V  | V | E | C   |      |   |   |    | V    | V | E |    | V | V | V | C   |
| WRTS    |    |   |   |     | U  |    |   | E |     |      |   |   |    |      |   | E |    |   |   |   | V   |
| WRTI    |    |   |   |     |    |    |   | U |     |      |   |   |    |      |   | U |    |   |   |   |     |
| INSBLK  |    |   |   |     |    |    |   | U |     |      |   |   |    |      |   | U |    |   |   |   |     |

#### Key

V = Valid.

C = Valid; current record pointer updated.

E = Valid; same as C if not writing at EOD. If writing at EOD, the current record pointer and EOD are updated.

U = Valid; current record pointer and EOD updated.

Figure 4-3. Access Methods for Data Set Types

**READ (Read a Data Set Record)**

| Mnemonic | Data Set | Format            | Record  | Operand1   |
|----------|----------|-------------------|---|--|
| READ     | (1-15    | [, { *<br>label } | $\left\{ \begin{array}{l} Rn \\ BRn[(4)] \\ - \\ + \\ 0 \end{array} \right\}$ | $\left\{ \begin{array}{l} N \\ O \end{array} \right\} )$ |

When the READ instruction is executed, the specified record is copied from the physical buffer into logical I/O buffer. If the logical record is not within the physical buffer, logical records that include the specified record are read from the diskette until the physical buffer is filled. Then the logical record is copied into the logical buffer. If formatting is to be done, the data is moved and formatted from the logical buffer into the locations specified in the indicated format.

**WRT (Write a Record)**

| Mnemonic | Data Set | Format  | Record  | Operand1   | Operand2 |
|----------|----------|---------|---|--|----------|
| WRT      | (1-15    | [,label | $\left\{ \begin{array}{l} BRn[(4)] \\ - \\ + \\ 0 \end{array} \right\}$ | $\left\{ \begin{array}{l} N \\ O \end{array} \right\}$ | ,B)      |

When the WRT instruction is executed with a format, the logical I/O buffer is blanked if the operand is coded B. The data is formatted and moved from the locations indicated in the format into the logical I/O buffer. The data is written from the logical I/O buffer into the physical buffer, at the position indicated by the record parameter. If you are using pointer I/O, the logical record address in the IOB is not modified unless you are extending the data set at EOD.

For an SCS data set, you can use the binary register normally specified for the Record operand to specify SCS control characters. See *SCS Conversion* under *Printer Instructions* for more information about an SCS conversion data set. See Appendix B for a description of SCS control characters.

**WRTS (Delete a Record)**

| Mnemonic | Data Set | Format  | Record | Operand1   | Operand2 |
|----------|----------|---------|--------|--|----------|
| WRTS     | (1-15    | [,label | ,0     | $\left\{ \begin{array}{l} N \\ O \end{array} \right\}$ | ,B)      |

When the WRTS instruction executes, the current record is written as in the WRT instruction. In addition, the record is marked as deleted.

For a basic or H exchange data set, a special address mark is used to flag a deleted sector. For an I or E exchange data set, the delete character specified on the HDR1 label is placed in the last byte of a deleted logical record.

If you issue a WRTS to a HDR1 label, it is written as a physically deleted record (address mark) if it is the only label in the sector. On 2D diskettes there may be 2 labels in a sector; physical deletes are used only when both labels in the sector are deleted.

If you want to delete a record at a relative record number, read the record at that relative record number, then delete the record with the WRTS operation. You can use WRTS at EOD to create a deleted record. In order to read a deleted record you must use a relative record read instruction.

*WRTI (Insert a Record)*

| Mnemonic | Data Set | Format   | Record | Operand1   | Operand2 |
|----------|----------|----------|--------|--|----------|
| WRTI     | (1-15    | [,label, | 0,     | $\left\{ \begin{array}{c} N \\ O \end{array} \right\}$ | ,B])     |

This instruction writes the current logical record to the current record position of the physical buffer. The record that was in the current record position, and all records beyond the inserted record, move down one position until EOD or a deleted record is encountered. There must always be enough room for one record at the end of the data set, even if there are deleted records in the data set. If the record is inserted as the last record in the data set, this instruction acts as a write (WRT) instruction. You must specify two physical buffers and a logical buffer in the .DATASET control statement for the data set in order to use the WRTI instruction. WRTI is not allowed for a shared data set, an SCS data set, or a pointer I/O data set.

*INSBLK (Insert a Block of Records)*

| Mnemonic | Data Set | Operand   |
|----------|----------|---|
| INSBLK   | (1-15    | „BRn ,[ $\left\{ \begin{array}{c} O \\ N \end{array} \right\}$ ]) |

The INSBLK operation inserts a number of logical records into the specified data set. The binary register (BRn) specifies the number of records to insert. The inserted records are treated as deleted records and may be written with the WRTI instruction. The current record counter is modified to point to the first record inserted. You must specify two physical buffers and a logical buffer in the .DATASET control statement for the data set in order to use the INSBLK instruction. INSBLK is not allowed for a shared data set, an SCS data set, or a data set that uses pointer I/O.

To make room for the inserted records, the current record and all following records are moved down. If there is not enough room in the data set for the inserted records, external status is presented and the insert does not take place.

If the instruction specifies 0, the operation is executed in overlapped mode. Otherwise it is executed in nonoverlapped mode.

If the current record pointer is set to EOD when this instruction is executed, you must reposition the pointer to EOD to perform write-current operations at EOD.

The WRTI instruction performs a similar operation except only one logical record is inserted and written. The INSBLK instruction can save time when a large number of records must be added to a data set. Performance is improved with a greater buffer size for I and E exchange.

### **Search Operations**

The IBM 5280 can search a data set for a record that agrees with one or more mask specifications. The IBM 5280 reads each logical record, then searches it for the mask specifications. If a record is found that matches the mask specifications, it is placed into the logical record buffer and the search ends. If no matching record is found, an external status is presented; the contents of the logical buffer depend upon the kind of search performed. You can specify a binary search to search the data set for a specified mask. Or you can specify a sequential search to search the data set or the current contents of the logical record buffer for one or more relational mask specifications.

A binary search is performed to find the location within the data set of the logical record that matches the contents of the mask specification. The data field that is searched must be in the same record position of each record and must be in ascending order.

A sequential search is performed to find a record that matches the contents of one or more mask specifications. Multiple mask specifications include relational and logical operators. There is no limit to the number of mask specifications you can include for each sequential search. The logical operations are logical AND and logical OR. The logical operations are performed from left to right with AND having priority over OR. You cannot group the mask specifications to give OR priority, such as: (FLD1 or FLD2) and FLD3. You can accomplish this operation, however, by expanding the specifications sequence to: FLD1 AND FLD3 OR FLD2 AND FLD3. For each mask specification you may indicate only one logical operator and one relational specification.

If translation is performed on the data set being searched, it occurs in the physical buffer before the logical records are searched.

A record begins with position 1. The detailed format for the mask specifications is illustrated with each search operation.

The format of the search instruction is as follows:

| Mnemonic | Data Set | Operand1 | Operand2   |
|----------|----------|----------|--|
| SEARCH   | (1-15    | ,BRn,    | $\left. \begin{matrix} B \\ F \\ R \\ L \end{matrix} \right\}$ ) |

where:

*Data Set* indicates the data set to search.

*Operand1* indicates a binary register that contains the storage address of the mask specifications. The mask specifications must be loaded into a storage address before you issue the SEARCH instruction. The format of the mask specifications depends upon the kind of search performed. Each format is explained with the search type descriptions.

*Operand2* indicates the kind of search to perform. One of the search keywords must be specified, as follows:

#### Operand2

| Entry | Description   |
|-------|---|
| F     | Forward sequential: an unordered data set is searched, starting with the record following the current logical record and ending with the desired record or the last data set record. Each record is searched for the mask specifications. If a matching record is found, it is written into the logical record buffer. If no matching record is found, the last logical record in the data set is placed into the logical record buffer and an external status (3702) is presented. |

If the current record pointer is at zero, the search starts at the beginning of the data set.

**Operand2  
Entry  
(continued)**

**Description**

The format of a mask specification for a type F search is as follows:

| <b>Byte</b> | <b>Contents</b>  |            |                     |   |             |   |            |   |                |   |                   |   |                         |   |                            |   |            |   |                |
|-------------|--|------------|---------------------|---|-------------|---|------------|---|----------------|---|-------------------|---|-------------------------|---|----------------------------|---|------------|---|----------------|
| 0-1         | Length of the mask.  |            |                     |   |             |   |            |   |                |   |                   |   |                         |   |                            |   |            |   |                |
| 2           | Relative and logical operators. The IBM 5280 does not check bits 0 and 1 when it processes the first mask specification. However, every following mask specification must have either bit 0 or bit 1 turned on. Each mask specification can have one, and only one, of bits 2-7 turned on. If more than one is on, an external status (3417) is presented.   |            |                     |   |             |   |            |   |                |   |                   |   |                         |   |                            |   |            |   |                |
|             | <table><thead><tr><th><b>Bit</b></th><th><b>Meaning If 1</b></th></tr></thead><tbody><tr><td>0</td><td>logical AND</td></tr><tr><td>1</td><td>logical OR</td></tr><tr><td>2</td><td>LT (less than)</td></tr><tr><td>3</td><td>GT (greater than)</td></tr><tr><td>4</td><td>LE (less than or equal)</td></tr><tr><td>5</td><td>GE (greater than or equal)</td></tr><tr><td>6</td><td>EQ (equal)</td></tr><tr><td>7</td><td>NE (not equal)</td></tr></tbody></table> | <b>Bit</b> | <b>Meaning If 1</b> | 0 | logical AND | 1 | logical OR | 2 | LT (less than) | 3 | GT (greater than) | 4 | LE (less than or equal) | 5 | GE (greater than or equal) | 6 | EQ (equal) | 7 | NE (not equal) |
| <b>Bit</b>  | <b>Meaning If 1</b>  |            |                     |   |             |   |            |   |                |   |                   |   |                         |   |                            |   |            |   |                |
| 0           | logical AND  |            |                     |   |             |   |            |   |                |   |                   |   |                         |   |                            |   |            |   |                |
| 1           | logical OR   |            |                     |   |             |   |            |   |                |   |                   |   |                         |   |                            |   |            |   |                |
| 2           | LT (less than)   |            |                     |   |             |   |            |   |                |   |                   |   |                         |   |                            |   |            |   |                |
| 3           | GT (greater than)  |            |                     |   |             |   |            |   |                |   |                   |   |                         |   |                            |   |            |   |                |
| 4           | LE (less than or equal)  |            |                     |   |             |   |            |   |                |   |                   |   |                         |   |                            |   |            |   |                |
| 5           | GE (greater than or equal)   |            |                     |   |             |   |            |   |                |   |                   |   |                         |   |                            |   |            |   |                |
| 6           | EQ (equal)   |            |                     |   |             |   |            |   |                |   |                   |   |                         |   |                            |   |            |   |                |
| 7           | NE (not equal)   |            |                     |   |             |   |            |   |                |   |                   |   |                         |   |                            |   |            |   |                |
| 3-4         | Field position in which to begin search.   |            |                     |   |             |   |            |   |                |   |                   |   |                         |   |                            |   |            |   |                |
| 5-6         | Field position in which to end search.   |            |                     |   |             |   |            |   |                |   |                   |   |                         |   |                            |   |            |   |                |
| 7-n         | Mask.  |            |                     |   |             |   |            |   |                |   |                   |   |                         |   |                            |   |            |   |                |

The mask specification can be repeated from byte 0. Follow the mask in the last specification with hex 0000 to indicate the end.

**B** Binary; an ordered data set is searched for a record that agrees with a mask specification. The mask specification is compared to a field that must be located in the same position of each record and must contain data in ascending order. If a record matching the mask is found, it is written into the logical record buffer. If a matching record is not found, the logical record immediately following the relative record position where the record would have been located is placed into the logical record buffer, and an external status (3702) is presented. However, if the record position would have been beyond the EOD, the last record in the data set is placed into the logical record buffer and an external status (3703) is presented.

**Operand2  
Entry  
(continued)      Description**

The format of a mask specification for a type B search is as follows:

| Byte | Contents                                 |
|------|--|
| 0-1; | length of the mask.                      |
| 2-3; | field position in which to begin search. |
| 4-n; | mask.                                    |

Only one mask specification may be used.

**R**      Reverse; an unordered data set is searched as for a forward sequential (F) search, except the data set is searched in a backward direction. The search begins with the record preceding the current logical record. If the current record pointer is at zero, the search begins at the end of the data set (at the last record). When a matching record is found, it is placed in the logical record buffer. If no matching record is found, the first record in the data set is placed in the logical record buffer, and external status (3702) is presented.

The format of the mask specification for a type R search is as for a type F search.

**L**      Logical record; the contents of the current logical record buffer are searched for mask specifications. If the record matches the mask specifications, no external status is presented. If the record does not match, an external status (3702) is presented.

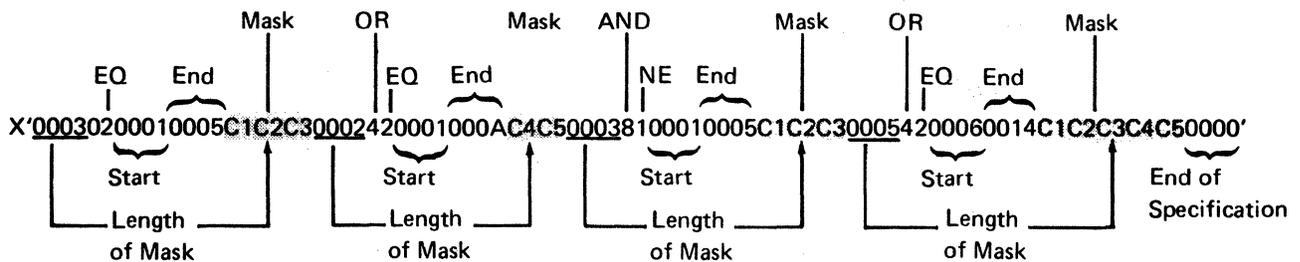
The format of the mask specifications for a type L search is as for a type (F) search.

*Examples:* The following mask specification uses a binary search to search a data set for a record containing 137, starting in position 15.

Length
Mask  
⏟
⏟  
X'0003000FF1F3F7'  
⏟
  
Position

The following mask specifications search a data set for a record that satisfies *one* of the following three conditions.

1. The record contains ABC in positions 1-5.
2. The record contains DE in positions 1-10 AND ABC is not in positions 1-5.
3. The record contains ABCDE in positions 6-20.



## PRINTER INSTRUCTIONS

Printer instructions include instructions to open, write, and close a data set. A printer data set must be described in a .DATASET control statement as follows:

- There must be one physical buffer; two physical buffers are allowed for double buffering unless the data set specifies share attributes.
- If the logical buffer address is omitted or specified to be the same as the physical buffer address, pointer I/O is selected by the assembler. Pointer I/O is not valid for SCS conversion data sets.
- The logical record length must be specified. It must be less than or equal to the length of the physical buffer, and less than or equal to the space allocated for the logical buffer.
- The data set type must be sequential write (SW).

You can write a diskette data set to the printer if the data set specifications are valid for the printer. If a diskette data set has an ALLOC (allocate) instruction, the printer processes it as though it were an OPEN instruction. The binary register specified with the ALLOC instruction is ignored.

The format of the printer instructions is as follows:

| Mnemonic | Data Set | Format  | Operands  |
|----------|----------|---------|---|
| OPEN     | (1-15)   |         |   |
| WRT      | (1-15    | [,label | ,0 , { $\begin{matrix} N \\ O \end{matrix}$ } ,B) |
| WAIT     | (0-15)   |         |   |
| CLOZ     | (1-15)   |         |   |

where:

*Data Set* specifies the data set number. This is the DSN parameter of the .DATASET control statement that set up the IOB for this data set. If data set is specified as 0 in the WAIT instruction, it specifies the keyboard rather than a data set. Do not specify 0 for any other instruction.

*Format* specifies the label of the edit format to use; specified only for the WRT instruction. Do not specify a format if you use pointer I/O.

*Operands* are explained for each instruction in the operation description.

### *OPEN (Open a Data Set)*

| <b>Mnemonic</b> | <b>Data Set</b> |
|-----------------|-----------------|
|-----------------|-----------------|

|      |        |
|------|--------|
| OPEN | (1-15) |
|------|--------|

When an OPEN instruction is executed, the printer specifications are initialized to the following:

Page size = 1; the forms control can page forward at every line.

SGEA = (-,1); print one dash and continue printing for unprintable characters.

Maximum line size = logical record length (from the .DATASET control statement).

These printer specifications can be altered by SCS control characters in the data stream. See Appendix B for a description of the SCS control characters.

The OPEN instruction is always processed in nonoverlapped mode. The data set must be a sequential write data set. The IOB is placed on the IOB chain, and the open flag in the data set IOB is set to indicate that the data set is open.

### *WRT (Write a Record)*

| <b>Mnemonic</b> | <b>Data Set</b> | <b>Format</b> | <b>Operands</b>  |
|-----------------|-----------------|---------------|--|
| WRT             | (1-15)          | [,label       | ,0 , $\left\{ \begin{matrix} N \\ O \end{matrix} \right\}$ ,B] |

When the WRT instruction is executed with a format, the logical I/O buffer is blanked if the operand B is coded. The data is formatted and moved from the locations indicated in the format into the logical I/O buffer. The data is written from the logical I/O buffer into the physical buffer unless you are using pointer I/O. Unless early write is specified, the logical records are not written from the physical buffer until the physical buffer is full. When the data has been sent from the physical buffer to the printer, it is placed into a printer device buffer. The physical buffer is free to accept more logical records while the data in the printer device buffer is being printed. Therefore, a print error can occur for data that is already sent from the physical buffer.

The zero, for write current, may be omitted, but the comma must be retained if an operand to the right is specified.

If O is specified, the write is executed as overlapped I/O. See *Overlapped I/O* in Chapter 2 for more information about overlapped I/O. If N is specified, the write is executed as nonoverlapped I/O.

### *WAIT (Wait for I/O Completion)*

| <b>Mnemonic</b> | <b>Data Set</b> |
|-----------------|-----------------|
|-----------------|-----------------|

|      |        |
|------|--------|
| WAIT | (0-15) |
|------|--------|

The WAIT instruction is used when you are processing with overlapped I/O. It is processed as described for the diskette instructions to prevent modification of the logical buffer prior to completion of the I/O instruction last issued to each I/O device.

## *CLOZ (Close a Data Set)*

| <b>Mnemonic</b> | <b>Data Set</b> |
|-----------------|-----------------|
| CLOZ            | (1-15)          |

When the CLOZ instruction is executed, the IBM 5280 checks to determine if the data that was sent to the printer for the data set has completed printing. If it has, the data set is closed; the IOB is removed from the IOB chain and the data set open flag in the IOB is turned off to indicate that the data set is not open. If it has not completed printing, and if it is the only IOB on the chain, the IBM 5280 will wait up to three minutes (approximately) for the printing to be completed. If the printing is completed within this time, the data set is closed. If the printing is not completed, a 2292 error for failure to close is reported. If more than one IOB is on the IOB chain, the time limit is extended. If a directed close is issued before the data has completed printing, the data still within the printer device buffer and the data that is still within the physical buffer in main storage is lost. See *Error Recovery Procedures*, in this section, for a description of a directed close.

## *SCS Conversion*

The format of the printed output can be altered with SCS (standard character string) control characters. You can insert the control characters into your data stream, or you can use an SCS conversion data set. See Appendix B for a description of the SCS control characters you can use if you are not using an SCS conversion data set. If you are using an SCS conversion data set, the IBM 5280 inserts the control characters into the data set. A data set that has SCS conversion specified in the .DATASET control statement is referred to as an SCS conversion data set. The IBM 5280 inserts the SCS control characters into the SCS conversion data set record during a write operation when the record is moved from the logical buffer to the physical buffer. All I/O operations for an SCS conversion data set are processed by the IBM 5280 rather than by the I/O device, and must therefore be specified as nonoverlapped I/O.

You can store a data set on a diskette and save diskette space by defining the data set as an SCS conversion data set. If you later write the data set from the diskette to the printer, do not again request SCS conversion on the data set. This would cause the IBM 5280 to again try to perform the SCS conversion processing on a data set that already contains SCS control characters. A data set that already contains SCS control characters is referred to as an SCS data set.

SCS control characters have EBCDIC values under hex 40. In order to prevent unpredictable printer results, do not place an EBCDIC below hex 40 into the logical buffer. If SCS control characters are in data that is to be translated, they must not be altered by the translation. See *Translation* under *Miscellaneous Instructions* in this chapter for more information.

When a data set is processed during SCS conversion, presentation control characters are substituted for four or more blanks that occur on a single line. Blank characters that occur between the last nonblank character and the end of the line are replaced by a new-line control character. When the IBM 5280 processes a line equal to or greater than the line number specified as the last line (LSTLN), an indicator (I115)

is set on. Your program may at this time cause SCS control characters to be placed into the data stream. This is done by including a binary register with the write instruction. The binary register holds the address of a string of control data. When using this binary register to point to control data to insert into the data stream, the line size (LINSIZ) and logical record length (LRECL) must be equal. The format of the data string is as follows:

| Byte | Bits | Meaning  |
|------|------|--|
| 0    | 0=1  | The IBM 5280 spaces the number of lines given in byte 1 of this control string. If the IBM 5280 is at or past the line specified for the LSTLN parameter of the .DATASET control statement, I115 is set on. A value of zero causes only a carriage return. Bit 0 and bit 1 are mutually exclusive.   |
|      | 1=1  | The IBM 5280 skips to the line number given in byte 1. If the line number is greater than the number specified for the PGSIZ parameter of the .DATASET control statement, an error (2229) is reported and processing stops. A value of zero causes only a carriage return. Bit 1 and bit 0 are mutually exclusive.   |
|      | 2=1  | The IBM 5280 inserts into the physical buffer the control characters specified in this data string. Byte 1 specifies the number of control characters to insert, Byte 2 is the first byte of control characters to insert. The control characters may be any of the control characters described in this appendix. The contents of the logical buffer are moved into the physical buffer, then the control characters are moved into the physical buffer.  |
|      | 3=1  | If you put in any control characters that change the presentation surface format (maximum number of lines per page or maximum number of characters per line), you must also update bytes 43 and 47 of the data set IOB to reflect these changes. If you put in control characters that change the line number or character position for the next character to be printed after the control string is completed, you must also update bytes 41 and 46 of the data set IOB to reflect these changes. |
|      | 4-7  | Reserved.  |
| 1    |      | Space control if byte 0, bit 0 = 1. Skip control if byte 0, bit 1 = 1. Number of bytes of control characters if byte 0, bit 2 = 1.   |
| 2    |      | First byte of control characters to move to physical buffer; used only if byte 0, bit 2 = 1.   |

The .DATASET control statement that requests SCS conversion must specify a logical buffer. You cannot use pointer I/O and specify SCS conversion concurrently. The .DATASET control statement must also specify the logical record length (RECL), the block size (BSIZ), and the line size, and they must be equal to or less than 132 bytes.

If you want to close a data set IOB that requested SCS conversion, then reopen the same data set IOB, you must reinitialize certain bytes of the IOB. (The bytes at displacement hex 40, 41, and 42 must be set to hex 00. The 2 bytes at displacement hex 44-45 must be reinitialized to the symbol you want to use to represent unprintable EBCDIC values [SGEA]). Reloading the program will accomplish this reinitialization.

### *Pointer I/O*

Pointer I/O may be used for a printer data set unless SCS conversion is specified. The logical records are blocked and deblocked directly to and from the physical buffer. The logical buffer address points to the address, within the physical buffer, where the next logical record is to be placed. See *Pointer I/O* in Chapter 2 for more information.

### *Early Write*

Early write may be specified for any printer data set to transfer one logical record to the printer for each write operation. If early write is not specified, the data is not sent to the printer until the physical buffer is full. Early write is always used for SCS conversion data sets whether or not it is specified in the .DATASET control statement.

### *Share Data Sets*

The conventional method for using printer data sets is to open a data set, issue write instructions, close the data set, then open the next data set. However, if you want to keep an IOB open but also let other IOBs use the same printer, you can use share data sets. A share data set has share attributes specified in the .DATASET control statement that sets up the data set IOB. The IOBs that share a printer may be in the same partition or in different partitions. Do not use double physical buffering for a share data set; use one physical buffer and one logical buffer. Pointer I/O is allowed for shared data sets. It is recommended that you always specify early write for a share data set so the requirement for a full physical buffer does not prevent data from being sent to the printer.

When more than one share data set IOB is open to the same printer, the printer uses the specifications (such as lines per page and line length) from the last IOB that was opened. For example, if the first share IOB specifies line length of 80 characters, then a second IOB is opened that specifies line length of 132 characters, the printer uses 132 characters for line length for printing data from either data set.

A share data set IOB retains control of the printer until the physical activity for that data set is complete (data has been moved from the physical buffer to the printer, and the printer has completed printing the data). If a second share data set IOB tries to use the printer (with an OPEN or a WRT instruction) before the physical activity for the first share IOB is complete, a 2755 warning message is reported to the second share IOB. Subsequent retries return the 2755 warning message to the second IOB until the physical activity for the first IOB is reported to be complete. The second share IOB then takes control of the printer until the physical activity for the second share IOB is reported to be complete.

It is possible for the printer to report physical activity complete for an IOB before all the data you expect from the IOB has been printed. This can happen, for example, when the IBM 5280 must process other instructions for the IOB between the WRT instructions. To avoid letting a second share IOB from gaining unexpected access to the printer, you may wish to use an indicator that prevents another IOB from using the printer and is turned off only when all the WRTs you expect from the first IOB have been executed. Any share IOB would check this indicator when it needs the printer, and would try to share the printer only when the indicator is off. Or, you can avoid using share data sets; open and close each data set IOB as it needs the printer.

### **Error Recovery Procedures**

In overlapped I/O mode, error recovery must be provided by the user programmer. In non-overlapped I/O mode, the IBM 5280 provides error recovery as follows.

Error recovery depends upon: (1) the instruction stored in the data set IOB when the error occurred, and (2) whether the operation that was being executed when the error occurred had completed processing data. If the operation had not completed processing data when the printer reported external status, the instruction may need to be reissued for that operation. For example, if a write operation is being executed and the printer reports external status before the logical record has been processed, the WRT instruction must be reissued to reload the logical buffer. Unless external status is reported when an OPEN is in the data set IOB, the IBM 5280 sets byte 13, bit 5 of the data set IOB when external status is reported before an operation has completed processing data. If the operator presses the Reset key the IBM 5280 goes to your printer external status subroutine. If your subroutine clears the condition and then issues a RETURN, the IBM 5280 returns to reissue the instruction if byte 13, bit 5 is on, and returns to the next sequential instruction if the bit is not on. (This bit can be tested with the IFDSI instruction, using dsin 13.)

If external status occurs when an OPEN is in the data set IOB, the IBM 5280 always returns to the instruction following the OPEN when the RETURN is issued.

The data set open flag (bit 0 of byte 13 of the IOB) indicates that the data set has been successfully opened and has not been closed. If an external status occurs during an OPEN, the OPEN must be reissued if processing is to continue for the printer. A CLOZ must be issued if processing is to not be continued for the printer. The data set open flag should not be used to determine if an OPEN must be reissued during error recovery. Once the data set open flag is turned on by a successful OPEN, it remains on until a successful CLOZ, even if an intervening OPEN is issued and results in an error.

### *A Directed Close*

If a CLOZ is issued after entering the external status subroutine, the data set is closed and any unprinted data is lost. This is referred to as a directed close. You may want to use a directed close when the external status condition cannot be cleared. The closed data set can then be opened for another printer.

### *External Status during Close*

If the printer reports external status when a CLOZ is in the data set IOB, the IBM 5280 checks byte 13, bit 5 of the data set IOB when the RETURN is issued. If the bit is on, the IBM 5280 reissues the CLOZ instruction. This causes a directed close, and can result in a loss of data. To avoid loss of the unprinted data, you can (1) have your subroutine keep checking byte 0, bit 2 (dsin 2) of the data set IOB until it is turned off (physical activity complete), then issue the RETURN, (2) direct the operator to clear the condition and allow the printer to finish printing before pressing the Reset key, or (3) if the printer is still printing (2292 error), direct the operator to let the printer finish printing before pressing the Reset key.

### *Operator Determined Error Recovery*

You can let the operator determine error recovery by using function keys or key sequences. For example, the Cmd, End of Job key sequence can be used to issue a CLOZ for a directed close. Another function key or key sequence could call a subroutine to check byte 0, bit 2 (dsin 2) of the data set IOB, and issue a close when this bit is off; this could be used instead of the Reset key after a printer error.

### *Discontinuing a Print Job from the Printer*

If you want to discontinue a print job from the printer before the printing is completed, use the Cancel switch rather than the Stop switch on the printer. Cancel causes a 2601 error; a CLOZ should be issued for this condition in your program. The printer will finish printing the data that has been sent from the physical buffer in main storage before it stops printing. If you use the Stop switch, the data that has been sent from the physical buffer in main storage is not cleared from the printer buffer when the printer stops printing; this data is retained until the Start switch is used, then it is printed.

## KEYBOARD AND DISPLAY INSTRUCTIONS

The keyboard and display instructions include the key entry commands and keyboard operations. The commands initiate formatted data entry through the keyboard. Keyboard operations allow unformatted key entry, cause limited data movement, and perform other tasks related to the keyboard and display screen.

A command or an operation can be requested by a program executing in either a foreground or background partition. While a keyboard operation is being processed for a partition, the keyboard attached to the partition is locked. If a keyboard operation is requested while a key entry command is in process, data entry under that command is suspended at least until the operation is completed.

### Key Entry Instructions

A formatted record can be entered only when an ENTR command is being processed. An ENTR command may be issued by a program loaded into any partition. However, the partition must be attached to its associated keyboard.

When the IBM 5280 encounters an ENTR while it is processing instructions within a partition, it unlocks the keyboard attached to that partition. The keyboard/display may then accept a record. This input record is formatted according to a screen format control string, which is generated from a series of .SFMT control statements and is specified in the ENTR command. The screen format control strings are described in Chapter 2 under *Screen Formats*.

During key entry under an ENTR command, an external status condition may occur. External status conditions are described in Chapter 6 under *Keyboard/Display External Status*. When an external status condition occurs, the keyboard/display locks the keyboard to temporarily suspend key entry. It sets an external status bit and specifies the condition number in the appropriate partition control area locations, then it notifies the IBM 5280. When the IBM 5280 checks the partition control area and finds an external status condition outstanding, it executes the appropriate external status subroutine. This subroutine handles the condition and determines when the keyboard is unlocked and when the external status bit is turned off. If the interrupted screen format is not completed, return from the external status subroutine is made to the place in the format control string where the interruption occurred. If the screen format is completed, return is made to the instruction following the ENTR instruction.

The format for the key entry command and external status subroutine instructions is as follows:

| Mnemonic | Operands   |
|----------|--|
| ENTR     | (label [,BRn , { $\begin{matrix} N \\ O \end{matrix}$ } ]) |
| RESUME   | [(B)]  |
| ENABLE   | (label [,POP])   |
| RESCAL   | (BRn ,label)   |
| RETEXT   | [(BRn)]  |
| RESMXT   | [(BRn)]  |

where:

*Operands* depend upon the particular instruction and are explained with the operand description for each instruction.

#### *ENTR (Accept key Entry)*

| Mnemonic | Operand1 | Operand2   | Operand3 |
|----------|----------|--|----------|
| ENTR     | (label   | [, BRn , { $\begin{matrix} O \\ N \end{matrix}$ } ]) |          |

where:

*Operand1* specifies the screen format control string with which to format the input record. Enter the LABEL parameter from the .SFMTST statement of the screen format you wish to use.

*Operand2* specifies that the functions of the current record buffer (CRBA) and the previous record buffer (PRBA) are alternated. The buffer used as the CRBA for the most recently completed ENTR becomes the PRBA for this ENTR. The binary register (BRn) is set to the address of this buffer. The buffer used as the PRBA for the most recently completed ENTR becomes the CRBA for this ENTR. If the binary register is omitted, the functions of the CRBA and the PRBA are not alternated; the CRBA for the most recently completed ENTR will be reused as the CRBA for this ENTR. If you wish to preserve the contents of the CRBA resulting from the most recently completed ENTR, you must use data movement instructions (such as MVC instructions) to move the contents of the CRBA to the PRBA. This must be done before each new record is entered into the CRBA.

*Operand3* indicates whether the ENTR is processed as overlapped (O) or nonoverlapped (N) I/O. This is an optional parameter that defaults to nonoverlapped I/O. See *Overlapped I/O* in Chapter 1 for more information.

When the IBM 5280 encounters an ENTR command, it sets a bit in the partition control area to indicate to the keyboard/display that an ENTR command is outstanding. The keyboard/display unlocks the keyboard and processes the input of the record, according to the screen format specified by operand 1. The prompts and display attributes specified in the screen format are moved to the screen in the order in which they were coded. The input data is checked against the screen format input field definitions. All valid data is placed into the input buffer designated to hold the current record.

If operand 2 specifies the binary register (BRn), the functions of the current record buffer and previous record buffer are alternated. If operand 3 specifies 0, overlapped I/O occurs.

*RESUME (Resume Key Entry)*

| <b>Mnemonic</b> | <b>Operand</b> |
|-----------------|----------------|
| RESUME          | [(B)]          |

where:

*Operand* is an optional entry to reposition the cursor after an external status hex 04 or 05 condition.

When data entry under an ENTR command is interrupted by an external status condition, the appropriate external status subroutine is called. If a RESUME is encountered in the external status subroutine, the keyboard is unlocked and input from the keyboard resumes under the interrupted ENTR command. RESUME does not return control from a subroutine, nor does it clear the external status bit in the partition control area.

Instruction processing will continue with the instruction following the RESUME instruction. At the same time, keyboard screen format processing will resume. If keyboard external status occurs in this condition, instruction processing of the original external status routine will be suspended in order to process the new external status. Also, binary registers BR19, BR20, BR21, and BR22 may be reset. When a RETURN is made, it will be to the point of interruption of the original external status routine.

If the operand (B) is included with the RESUME instruction, it is ignored except after an external status hex 04 or 05 condition. After an external status hex 04 or 05 condition, which indicates a forward or backward pass over an RG specification, the cursor is repositioned to the last manual position *preceding* the RG specification. There must be at least one manual data position defined preceding the RG specification when the RESUME(B) instruction is executed.

The (B) option should only be used to intentionally position the cursor in the manual position preceding the RG specification, regardless of what keystroke caused the pass over Rg specification.

If external status hex 04 has occurred, and the operand (B) is omitted, the format resumes processing in a forward direction with the format specification *following* the RG specification.

If external status hex 05 occurred and the operand (B) is omitted, the format resumes processing in a backward direction with the format specification preceding the RG specification.

The RESUME instruction should not be issued if byte 8 of the keyboard I/O control block is 0.

This operation is not meaningful for IBM 3270 mode.

#### *RESCAL (Resume and Call Subroutine)*

| <b>Mnemonic</b> | <b>Operand1</b> | <b>Operand2</b> |
|-----------------|-----------------|-----------------|
| RESCAL          | (BRn            | , label)        |

where:

*Operand1* specifies a binary register that contains either the index into the label table or a displacement.

*Operand2* specifies the label of a subroutine or a label table.

When a RESCAL command is encountered in an external status subroutine, the keyboard is unlocked and the processing of the interrupted ENTR command is resumed. At the same time, the subroutine specified by the operands is called and executed.

If the operands specify a label with no binary register, the call is made to the subroutine at that label. If a subroutine label is specified with a binary register, the contents of the binary register are added to the address of the subroutine and the call is made to the resulting address. If a binary register is specified with the label of a label table, the contents of the binary register are used as the index into the label table. The call is made to the address at that index.

This instruction can be used when the cursor enters a screen format field that has CNTL = RG specified in the control string specification for that field. Entering the field causes external status condition 4; the IBM 5280 calls the appropriate subroutine, which includes a RESCAL command. The processing of the ENTR continues, and the IBM 5280 calls the subroutine specified by the RESCAL instruction. This subroutine executes, probably processing the data entered into the field, while input for successive fields of the input record are being accepted by the keyboard. RESCAL does not turn off the external status bit in the partition control area or return control from the external status subroutine.

This operation is not meaningful for IBM 3270 mode.

#### *ENABLE (Reset External Status Bit)*

| <b>Mnemonic</b> | <b>Operand</b>   |
|-----------------|------------------|
| ENABLE          | [(label [,POP])] |

Whenever an external status condition occurs during the processing of an ENTR, the external status bit is set in the partition control area. If an ENABLE instruction is encountered in the external status subroutine that is called, the bit is cleared.

If no operand label is included in the instruction, the next sequential instruction is executed.

If a label is included in the instruction, a branch is made to the statement at that label.

If POP is coded, BR18 is decremented by two. It points no longer to the last entry into the partition subroutine stack, but to the entry prior to the last entry. The ENABLE instruction may be used when the subroutine is terminated with a GOTO rather than a return operation.

**Note:** If a RESUME or RESCAL is issued before an ENABLE, you can get a double external status condition, especially with a forward or backward pass over an RG.

*RETEXT (Subroutine Return and Enable External Status)*

**Mnemonic    Operand**

RETEXT    [(BRn)]

This instruction acts as the RETURN and ENABLE instructions combined. The subroutine is terminated and the external status bit is turned off. BR18 is decremented by two, and the return is made to the address in the partition subroutine stack pointed to by BR18. If the operand binary register is included, the contents of the binary register are added to the address in the partition stack, and the return is made to the resulting address.

*RESMXT (Retext and Resume)*

**Mnemonic    Operand**

RESMXT    [(BRn)]

This instruction acts as the RETURN, ENABLE, and RESUME instructions combined. The subroutine is terminated, the external status bit is turned off, and data entry processing is resumed. BR18 is decremented by two, and the return is made to the address in the partition subroutine stack pointed to by BR18. If the operand binary register is included, the contents of the binary register are added to the address in the partition stack, and the return is made to the resulting address.

This operation is not meaningful for IBM 3270 operation.

**Note:** Use of RESMXT avoids possible double external status that can result from issuing a RESUME or RESCAL before an ENABLE.

**Keyboard Operations**

Instructions for keyboard operations may be issued at any location in your program. If a keyboard operation is issued while an ENTR command is being processed, data entry under the command is suspended at least until the operation is completed.

The format of keyboard operation instructions is as follows:

| <b>Mnemonic</b> | <b>Operand1</b>     | <b>Operand2</b> |
|-----------------|---------------------|-----------------|
| BUZZ            |                     |                 |
| CLICK           |                     |                 |
| CNENTR          |                     |                 |
| DISPEX          |                     |                 |
| DISPST          |                     |                 |
| KACCPT          | (BRa, BRb)          |                 |
| KATTCH          |                     |                 |
| KDETCB          |                     |                 |
| KERRCL          | (BRa)               |                 |
| KERRST          | (BRa, BRb)          |                 |
| KEYOP           | (X'nn' [,BRa ,BRb]) |                 |
| READMG          | (BRa, BRb)          |                 |
| RSTMG           |                     |                 |
| REPFLD          |                     |                 |
| RTIMER          | (BRa)               |                 |

The operands depend upon the particular mnemonic. They are described in the operation description for each mnemonic.

**Note:** Repeated use of these instructions may result in a degradation of performance because the schedule for processing the partitions is altered.

#### *BUZZ (Sound Buzzer)*

##### **Mnemonic**

##### **BUZZ**

The BUZZ instruction sounds the alarm on the keyboard associated with the partition. The duration of the alarm is approximately 180 milliseconds.

This instruction can be used to signal the operator for various purposes. For example, during key entry the operator may not watch the screen at all times. The buzzer can bring the operator's attention back to the screen for a particular prompt, when a screen format control string has completed, after a record advance, or when a background program attaches to the keyboard.

#### *CLICK (Sound Click)*

##### **Mnemonic**

##### **CLICK**

The CLICK instruction clicks the keyboard associated with the partition. The CLICK instruction could be used after accepting keystrokes under the KACCPT operations, to give a response click.

### *CNENTR (Cancel Current ENTR)*

#### **Mnemonic**

#### **CNENTR**

The current enter command is terminated. The end of format control string functions are executed, and data entry is no longer accepted by the keyboard/display. On the status line, the counters, insert mode symbol, keyboard shift, and hex display positions are set to blanks. The command opcode is set to 0.

If there is no current ENTR outstanding, the CNENTR acts as a null operation and the next sequential instruction is executed.

This instruction may be used when an error occurs in a field and the operator indicates that no further processing of the record should be done. It can also be used to exit from a marked record following a record mark.

If this operation is executed in an external status subroutine during the processing of a nonoverlapped ENTR, the return issued in the subroutine is made to the interrupted ENTR. The ENTR is reissued and processing begins at the start of the format control string specified.

This operation is not meaningful for IBM 3270 mode.

### *DISPEX (Display Extra Line)*

#### **Mnemonic**

#### **DISPEX**

The top line on the screen can display either the status line or an extra line used by the screen format. The status line is referred to as row 0, and the extra line is referred to as row 1. Both row 1 and row 0 are always maintained, although only one may be displayed at any given time. See *Nondisplay of the Status Line* in Chapter 2.

The DISPEX instruction is used when the extra line (row 1) is being used in the screen format, and the status line has been displayed in its place to report an error. The DISPEX instruction returns the extra line to the screen, replacing the status line.

### *DISPST (Display Status Line)*

#### **Mnemonic**

#### **DISPST**

The DISPST instruction displays the status line (row 0) in the top line of the screen, replacing the extra line (row 1). The extra line is maintained and may be returned to the screen with the DISPEX instruction above.

### *KACCPT (Accept Keystrokes and Store)*

| <b>Mnemonic</b> | <b>Operand1</b> | <b>Operand2</b> |
|-----------------|-----------------|-----------------|
| KACCPT          | (BRa,           | Brb[(4)])       |

where:

*Operand1* indicates a binary register that contains the storage address to which the keystroke data is stored.

*Operand2* indicates a binary register or binary double register. If a 2-byte binary register is specified, it contains the information described for bytes 0 and 1 below; the keystrokes are not displayed as they are entered. If a 4-byte binary double register is specified, it contains the information described for bytes 0 to 3 below; the keystrokes are displayed as they are entered.

| <b>Byte</b> | <b>Bit</b> | <b>Meaning</b>   |
|-------------|------------|--|
| 0           | 0          | Option flags:<br>=1 if the keyboard is to sound a response click for each keystroke.<br>=0 if the keyboard is not to sound a response click. |
|             |            | 1-4<br>Not used.   |
|             | 5          | =1 if the monospace function is enabled.<br>=0 if the monospace function is not enabled.   |
|             | 6-7        | Keyboard shift flags:<br>=00 for Alpha shift<br>=01 for Num shift<br>=10 for special characters shift  |
| 1           | 0-7        | Number of keystrokes to accept, minus 1.   |
| 2           | 0-7        | Row number where keystroke display begins.   |
| 3           | 0-7        | Column number where keystroke display begins.  |

If the operand 2 register is not a binary double register, the number in the low-order byte specifies one less than the number of keystrokes to accept. If bit zero of the high-order byte of the operand2 register (BRb) is not zero, the keyboard sounds a response click for each keystroke. If bit 5 of the high-order byte is not zero, keystrokes are converted to their uppercase equivalent as they are entered. The keyboard is set to the shift indicated by bits 6-7. The keystroke data is stored in main storage, starting at the byte specified by the operand 1 binary register (BRa). The keystrokes are not displayed on the screen.

If operand 2 is a binary double register, the data is displayed as it is entered. The rightmost binary register contains the row and column position where the first accepted keystroke is displayed. The row number is specified in the high order byte of the rightmost binary register, and the column number is specified in the low-order byte. The leftmost register contains data as described above.

The keystroke data is stored in byte pairs. The first byte of each pair contains the keystroke scan code, and the second byte contains the EBCDIC code generated from the scan code. The data is not applied to an outstanding ENTER command. Shift keys are not stored into main storage; although the keyboard is shifted, they do not effect the keystroke count. Command key sequences are stored except when the Command key is followed by a console request. In this case, only the command is stored before the console function is performed. Hex key sequences and diacritic key sequences are stored as two keystrokes. The scan code and EBCDIC for control keys such as the Enter key are stored, but the control function is not performed.

If this instruction is issued from a foreground partition, there should be no ENTER command outstanding. If this instruction is issued from a background partition, the background partition must be attached to the associated keyboard.

This instruction could be used after an error status is displayed, to accept the Reset key.

This operation is not valid for IBM 3270 mode.

### *KATTCH (Attach Keyboard/Display to Partition)*

#### **Mnemonic**

#### **KATTCH**

When the KATTCH operation is executed, the partition that issued the KATTCH instruction is attached to the associated keyboard. All keystrokes from the attached keyboard are directed to the attached partition. A partition must be attached to a keyboard in order to perform an ENTR, a KACCPT, or a keyboard operation to pass scan code (KEYOP(OA)) or to pass EBCDIC (KEYOP(OB)), KERRST, KERRCL, perform keyboard function (KEYOP(11)), or open keyboard (KEYOP(15)).

The KATTCH operation can be initiated by the KATTCH instruction or by an operator initiated console request.

Each foreground partition is permanently associated with (assigned to) a keyboard/display. A background partition is associated with the keyboard/display that was attached to the partition that loaded the background partition. One foreground and several background partitions can be associated with a keyboard/display, but only one partition can be attached to a keyboard/display at any given time. If all background partitions associated with a keyboard/display are detached (see the KDETCH instruction), the associated foreground partition is assumed to be attached in that all function keys and command key sequences that normally cause an external status condition are directed to the foreground partition. Also, an automatic attach operation is performed each time a foreground partition issues a keyboard I/O instruction, so the foreground partition does not have to issue a KATTCH instruction.

If the attach operation is successfully completed, control returns to the second instruction following the KATTCH instruction. If the operation is not completed successfully, perhaps because another background partition is attached or the foreground partition is performing I/O, control returns to the first instruction following the KATTCH instruction.

When a partition is successfully attached to a keyboard/display, the status line is modified to display the partition number of the newly attached partition.

This operation is not valid for IBM 3270 mode.

#### *KDEATCH (Detach Keyboard)*

##### **Mnemonic**

**KDEATCH**

The KDEATCH instruction releases the keyboard. It is issued by the program in the partition that is currently in control of the keyboard, or by the common function operator detach (CFPERATT) routine in response to the Attn (Attention) key. Normally, it is better to let the operator control the detach with the Attention key rather than to issue a KDEATCH instruction from the program.

It is essential that a KDEATCH operation be executed when a partition no longer needs the keyboard. Failure to detach the keyboard inhibits any other partition from attaching to the keyboard. See CFPERATT and CFDETFGD under *Common Function Routines* in Chapter 6 for more information.

This operation is not valid for IBM 3270 mode.

#### *KERRST (Request Keyboard Error Mode)*

| <b>Mnemonic</b> | <b>Operand1</b> | <b>Operand2</b> |
|-----------------|-----------------|-----------------|
| KERRST          | (BRa,           | BRb)            |

where:

*Operand1* indicates a binary register that contains an attribute mask and control information for the display of the status line. The format of the information is as follows:

| <b>Byte</b> | <b>Bit</b> | <b>Meaning If 1</b>   |
|-------------|------------|---|
| 0           |            | Attribute Mask  |
|             | 0          | Buzz keyboard   |
|             | 1-2        | Reserved  |
|             | 3          | Column separators displayed   |
|             | 4          | Blink   |
|             | 5          | Underscore  |
|             | 6          | Highlight   |
| 1           | 7          | Reverse image   |
|             |            | Control Information   |
|             | 0          | Display status line if it is not currently being displayed                  |
|             | 1          | Start in column 1.<br>(if bit 1=0, start in column 3)                       |
|             | 2-7        | Message length minus 1, up to 63. If 63 is specified, it indicates 0 bytes. |

*Operand2* indicates a binary register that contains the storage address of the message to move to the status line.

The KERRST instruction places the keyboard/display in software error mode.

When the keyboard/display unit is in error mode, all data keys, function keys, and command key sequences are ignored. However, if the KEYOP instruction for keyboard operation hex 11 (perform keyboard function) is issued, the keyboard function is performed regardless of the error mode as long as the keyboard is in an appropriate state.

Bits 3-7 of the attribute mask are exclusively ORed with bits 3-7 of the row attribute byte (which determines the display of the row) for the top line of the screen. If the status line is not being displayed, the extra line will have the attributes specified.

Bytes are moved from the storage address specified by operand2 (BRb) to the status line. The bytes are translated through the display translate table, and attributes are translated and passed. The bytes moved from storage overwrite the original status line data, and the original status line data is destroyed.

If the status line is currently being displayed when this instruction is executed, the indicated message is displayed in column 1 or column 3, according to byte 1, bit 1 of operand1 (BRa). If the status line is not currently being displayed, the message is not displayed if byte 1, bit 0 contains 0; it is displayed if it contains 1.

This operation is invalid if the keyboard/display is already in error mode, or if issued from an unattached partition.

The KERRST can be used by the program to signal a software detected error, such as in a self-check field edit. An example of the KERRST instruction follows the KERRCL operation description.

This operation is not valid for IBM 3270 mode.

#### *KERRCL (Request Keyboard Error Mode Reset)*

| Mnemonic | Operand1 |
|----------|----------|
| KERRCL   | (BRn)    |

where:

*Operand1* specifies a binary register that contains an attribute mask and control information, as for KERRST.

The KERRCL instruction takes the keyboard/display unit out of software error mode. A change of screen attributes and display is allowed, according to the contents of the binary register (BRn). This register contains an attribute mask and control information as described for KERRST. If byte 1, bit 0 = 1, and the KERRST operation caused the status line to replace the extra line in the top row of the screen, the status line is removed from the screen and the extra line is redisplayed. Byte 1, bit 1 indicates the column in which the KERRST message started. Byte 1, bits 2-7 indicates the number bytes (of the KERRST message) to replace with blanks when the KERRCL executes. Byte 0, the attribute mask is as for KERRST and can restore the attributes changed by the KERRST operation.

If an ENTR is outstanding, and bits 2-7 of byte 1 do not equal 0, the field shift, current position counter, insert mode indicator, and positions remaining in current field counter are restored in the status line.

The KERRCL operation is invalid unless a KERRST operation has set the keyboard/display in error mode. It is invalid if issued from an unattached partition.

After a KERRST operation is executed, the KERRCL operation must be executed in order to enable the keyboard.

*Example:* The program can branch to ERRCK when an error in a self-check field edit is encountered during key entry.

```

ERRCK: KERRST (BRX, BRY) ; keyboard in software error mode,
*          blinking error message is displayed.
          KACCPT (BRXX, BRY) ; accept the Reset key.
*          Your program code to resolve the error.
          KERRCL (BRX) ; keyboard software error mode cleared,
*          blink attribute removed, message blanked.
          RESUME ; accept key entry under current ENTR.

```

The program can now request that the invalid field be rekeyed.

This operation is not valid for IBM 3270 mode.

#### *KEYOP (Keyboard Operation)*

| Mnemonic | Operand1 | Operand2 | Operand3 |
|----------|----------|----------|----------|
| KEYOP    | (X'nn')  | [,BRa]   | [,BRb])  |

where:

*Operand1* specifies the keyboard operation code that indicates the operation to perform.

*Operand2* indicates a binary register that contains the parameter1 information for the operation.

*Operand3* indicates a binary register that contains the parameter2 information for the operation.

The KEYOP instruction can perform operations that have no mnemonics. It can also perform several operations that do have their own instruction mnemonics. These mnemonics are listed below. For mnemonics that require operands, the entries for operand1 and operand2 are used for parameter1 and parameter2, respectively. Each of these mnemonics is described individually and is not repeated for the KEYOP instruction.

| Keyboard<br>Op Code | Mnemonic | Parameter1 | Parameter2 |
|---------------------|----------|------------|------------|
| 10                  | BUZZ     |            |            |
| 14                  | CLICK    |            |            |
| 05                  | CNENTR   |            |            |
| 0C                  | DISPEX   |            |            |
| 0D                  | DISPST   |            |            |
| 09                  | KACCPT   | (BRa,      | BRb)       |
| 0F                  | KERRCL   | (BRn)      |            |
| 0E                  | KERRST   | (BRa,      | BRb)       |
| 17                  | READMG   | (BRa,      | BRb)       |
| 16                  | RSTMG    |            |            |
| 03                  | RTIMER   | (BRn)      |            |

*Example:* To perform the KERRST operation with the KEYOP instruction, code the following KEYOP instruction:

KEYOP (X'0E', MASKBRN, ADDRBRN)

The KEYOP instructions for operations that have no mnemonics (and can be performed only by the KEYOP instruction) are as follows:

| Operation   | Keyboard<br>Op Code |
|---|---------------------|
| Change row attribute  | 07                  |
| Open keyboard/display   | 15                  |
| Pass scan code to keyboard  | 0A                  |
| Pass EBCDIC to keyboard   | 0B                  |
| Perform keyboard function   | 11                  |
| Position screen position pointer                                      | 08                  |
| Release character and field edits                                     | 06                  |
| Keyboard/display read buffer assist                                   | 80                  |
| Keyboard/display read modified assist                                 | 81                  |
| Keyboard/display write assist for the display                         | 82                  |
| Keyboard/display write assist for the printer                         | 83                  |
| Keyboard/display null non-print fields                                | 84                  |
| Keyboard/display erase all unprotected assist                         | 85                  |
| Keyboard/display read buffer assist for SNA SSCP-SLU<br>owned session | 86                  |
| Keyboard/display indicator and keyboard control                       | 87                  |
| Keyboard/display clear and initialize screen                          | 88                  |

| Mnemonic | Keyboard<br>Op Code | Operation                         |
|----------|---------------------|-----------------------------------|
| KEYOP    | (X'06')             | Release character and field edits |

The following character and field edit checks are discontinued for the current field:

- Character set check
- Data required

- Blank check
- Mandatory enter
- Mandatory fill

The checks are discontinued only until the field is exited in the forward or backward direction. If the same field is later advanced or backspaced into, the checks will be in effect.

This operation is not valid for IBM 3270 mode.

| Mnemonic | Keyboard |  | Parameter1 | Parameter2 | Operation            |
|----------|----------|--|------------|------------|----------------------|
|          | Op Code  |  |            |            |                      |
| KEYOP    | (X'07'   |  | ,BRa       | ,BRb)      | Change row attribute |

Parameter1 specifies a binary register that contains, in the low-order byte, the relative row number of the row attribute on the screen to be changed. The valid range for the row number is from 1 to the maximum number of lines on the screen. Parameter2 specifies a binary register that contains two 1-byte masks. The format of the row attribute byte is as follows:

| Bits | Meaning  |
|------|--|
| 0-1  | System Indicator<br>00= None<br>01= None<br>10= Dash (—)<br>11= Solid rectangle (■)  |
| 2    | 1= Valid row starting attribute. This bit must equal 1 for bits 3-7 to be valid.<br>0= Invalid row starting attribute. Bits 3-7 are ignored. |
| 3    | 1= Column separators are displayed.<br>0= Column separators are not displayed.   |
| 4    | 1= Blink the row<br>0= Do not blink the row  |
| 5    | 1= Underline the row <sup>1</sup><br>0= Do not underline the row   |
| 6    | 1= High intensity <sup>1</sup><br>0= Normal intensity  |
| 7    | 1= Reverse image <sup>1</sup><br>0= Do not reverse image   |

When the operation is executed, the row attribute byte (which determines the display of the row, maintained by the IBM 5280) for the row specified in parameter1 is logically ANDed with the mask in the high-order byte of parameter2. The result is then exclusively ORed with the mask in the low-order byte of parameter2.

Note that a valid row attribute remains in effect until the next valid row attribute or the next valid character attribute. Normally, the display contains a valid row attribute for row 1 only.

<sup>1</sup>If bits 5, 6, and 7 equal 111, the display of the row is inhibited.

| Mnemonic | Keyboard<br>Op Code | Parameter1 | Operation             |
|----------|---------------------|------------|-----------------------|
| KEYOP    | (X'08'              | ,BRn)      | Change cursor address |

The contents of the binary register (BRn) specify the row (high-order byte) and column (low-order byte) to place in the cursor address bytes of the I/O control block. The keyboard/display uses this address as the screen position pointer to determine the placement of fields and prompts on the screen during formatted key entry.

If this operation is performed before an ENTR command that specifies a format control string with CNTL = CP (continue at current screen position) in the .SFMTST statement, the format is initialized at the screen position specified in the binary register (BRn).

If this operation is performed while an ENTR command is being processed, all subsequent screen definitions for fields and prompts originate at the position specified by the binary register (BRn). The cursor does not move over intervening fields and prompts.

This operation not meaningful for IBM 3270 mode.

**Note:** Use of this operation while an ENTR command is being processed is not recommended.

| Mnemonic | Keyboard<br>Op Code | Parameter1 | Operation                               |
|----------|---------------------|------------|---|
| KEYOP    | (X'0A'              | ,BRn)      | Pass scan code from storage to keyboard |

The binary register (BRn) contains the address in main storage of one scan code. This scan code is moved to the keyboard associated with the partition. The scan code is then processed as if it originated from the keyboard.

This operation is not valid for IBM 3270 mode.

| Mnemonic | Keyboard<br>Op Code | Parameter1 | Operation                            |
|----------|---------------------|------------|--------------------------------------|
| KEYOP    | (X'0B'              | ,BRn)      | Pass EBCDIC from storage to keyboard |

The binary register (BRn) contains the address in main storage of one EBCDIC character. The EBCDIC character is moved to the keyboard associated with the partition. If the EBCDIC character corresponds to a data key EBCDIC or a function key EBCDIC, it is then processed as if it originated from the keyboard. The associated scan code is set to 0.

EBCDIC codes 29 (clear screen) and 2A (clear status line) are ignored in this operation because they are not function key EBCDICs. These functions can be performed by specifying keyboard operation code 11 (Perform Keyboard Function).

This operation is not valid for IBM 3270 mode.

| Mnemonic | Keyboard<br>Op Code | Parameter1 | Operation                 |
|----------|---------------------|------------|---------------------------|
| KEYOP    | (X'11'              | ,BRn)      | Perform keyboard function |

The low-order byte of the binary register (BRn) contains a function EBCDIC (hex 01 to 2C). If the EBCDIC is a function key EBCDIC, it is processed as if the key corresponding to that function has been pressed, with the following two exceptions.

1. The keyboard bit map is not checked to determine whether the program handles the function.
2. If the keyboard is in software error mode, the function executes if the state of the keyboard is appropriate. If the function is 29 (clear screen) or 2A (clear status line), it executes regardless of the state of the keyboard. If a function EBCDIC other than hex 01 to 2C is specified, an invalid operation external status condition occurs. The codes for the functions are described in the *Functions Reference Manual*.

This operation is not valid for IBM 3270 mode.

| Mnemonic | Keyboard<br>Op Code | Parameter1 | Operation             |
|----------|---------------------|------------|-----------------------|
| KEYOP    | (X'15')             |            | Open keyboard/display |

The keyboard/display is initialized as follows:

1. The clear screen function (29) is performed.
2. The clear status line function (2A) is performed.
3. The cursor is erased from the screen.
4. The blink attribute of the top line displayed on the screen is cleared unless a keystroke error or software error is outstanding.

If this operation is issued from an unattached partition, an external status for invalid operation occurs.

This operation is automatically performed during a partition load operation, and should not normally be used by an application program.

This operation is not valid for IBM 3270 mode.

| Mnemonic | Keyboard<br>Op Code | Parameter1 | Parameter2 | Operation                              |
|----------|---------------------|------------|------------|--|
| KEYOP    | (X'80',             | BRa,       | BRb[(4)])  | Keyboard/display read<br>buffer assist |

Parameter1 specifies a binary register that contains the address of the data stream buffer. Parameter2 specifies a single or double binary register. If a 2-byte single binary register is specified (optional 4 not supplied, meaning the read is for a screen), it contains the information described in bytes 0 and 1. If a 4-byte double binary register is specified (optional 4 is supplied, meaning the read is for a printer), it contains the information described in bytes 0-3.

| Byte | Bit        | Meaning  |
|------|------------|--|
| 0    | 0          | Indicator of where to begin read.<br>0 = Device buffer address 0.<br>1 = Current buffer address (held in KB/CRT IOB if read is for a screen, or printer control block if read is for a printer). |
|      | 1-3        | Must be 0  |
|      | 4-7        |  |
| 1    | and<br>0-7 | Maximum length of data stream.   |
| 2-3  |            | Address of a printer control block of 11 bytes in the following format:  |

| Bytes | Contents                                  |
|-------|---|
| 1-2   | Device buffer address                     |
| 3-4   | Work buffer address                       |
| 5-6   | Current buffer address                    |
| 7-8   | Current cursor position                   |
| 9     | EBCDIC-Internal translation table address |
| 10    | Flag/translation table page number        |
| 11    | Internal-EBCDIC translation table address |

The microcode builds an IBM 3270 read buffer data stream using the data contained in the device buffer. The data stream begins with the cursor address.

At the completion of the operation, the microcode stores the length of the data stream in the 1921st and 1922nd bytes of the device buffer. The length includes the cursor address bytes. If the length of the data stream to be built is greater than the maximum length specified, or if the maximum length specified is less than 2, an external status code 17 is posted.

| Mnemonic | Keyboard<br>Op Code | Parameter1 | Parameter2 | Operation                             |
|----------|---------------------|------------|------------|---------------------------------------|
| KEYOP    | (X'81',             | BRa,       | BRb[(4)])  | Keyboard/display read modified assist |

Parameter1 specifies a binary register that contains the address of the data stream buffer. Parameter 2 specifies a single or double binary register. If a 2-byte single binary register is specified (optional 4 not supplied, meaning the read is for a screen), it contains the information described in bytes 0 and 1. If a 4-byte double binary register is specified (optional 4 is supplied, meaning the read is for a printer), it contains the information described in bytes 0-3.

| Byte | Bit | Meaning  |
|------|-----|--|
| 0    | 0   | Indicator of where to begin read.<br>0 = Device buffer address 0.<br>1 = Current buffer address (held in KB/CRT IOB if read is for a screen or printer control block if read for a printer). |
|      | 1-3 | Must be 0.   |
|      | 4-7 |  |
| 1    | and |  |
|      | 0-7 | Maximum length of data stream.   |
| 2-3  |     | Address of a printer control block of 11 bytes in the following format:  |

| Bytes | Contents                                  |
|-------|---|
| 1-2   | Device buffer address                     |
| 3-4   | Work buffer address                       |
| 5-6   | Current buffer address                    |
| 7-8   | Current cursor position                   |
| 9     | EBCDIC-Internal translation table address |
| 10    | Flag/translation table page number        |
| 11    | Internal-EBCDIC translation table address |

The microcode builds an IBM 3270 read modified data stream using the data contained in the device buffer. The data stream begins with the cursor address.

At the completion of the operation, the microcode stores the length of the data stream in the 1921st and 1922nd bytes of the device buffer. The length includes the cursor address bytes. If the length of the data stream to be built is greater than the maximum length specified, or if the maximum length specified is less than 2, an external status code of 17 is posted.

| Mnemonic | Keyboard |  | Parameter1 | Parameter2 | Operation                                     |
|----------|----------|--|------------|------------|---|
|          | Op Code  |  |            |            |   |
| KEYOP    | (X'82',  |  | BRa,       | BRb (4) )  | Keyboard/display write assist for the display |

Parameter1 specifies a binary register that contains the address of the data stream. Parameter2 specifies a 4-byte double binary register that contains the information described in bytes 0-3.

| Byte | Bit | Meaning  |
|------|-----|--|
| 0    | 0   | Indicator of where to begin write.<br>0 = cursor address.<br>1 = current buffer address.   |
|      | 1-2 | Indicator of work buffer initialization.<br>00 = no initialization.<br>01 = initialize to null.<br>10 = initialize to device buffer contents.<br>11 = no initialization. |
|      |     | If initialize to null is specified, the cursor address and the current buffer address are also set to zero before the write begins.                                      |
|      | 3   | Indicator for SSCP-SLU<br>0 = not SSCP-SLU.<br>1 = SSCP-SLU  |
|      |     | If bit 3 is set to 1, the work buffer is unconditionally initialized to the device buffer.   |
|      | 4-7 |  |
|      | and |  |
| 1    | 0-7 | Length of data stream.   |
| 2    | 0-7 | Must be 0.   |
| 3    | 0-3 | Must be 0.   |
|      | 4   | Data stream translation indicator.<br>0 = Translate data stream.<br>1 = Do not translate data stream.  |
|      | 5   | Sound alarm indicator.<br>0 = Do not sound alarm.<br>1 = Sound alarm.  |
|      | 6   | Keyboard restore indicator.<br>0 = Do not clear input inhibited indicator after write.<br>1 = Clear input inhibited indicator after write.                               |
|      | 7   | Reset MDT indicator.<br>0 = Do not reset modified data tags.<br>1 = Reset modified data tags before write.   |
|      |     | (Bits 4, 5, and 7 are ignored when SSCP-SLU is specified.)   |

The microcode processes an IBM 3270 write data stream directed to the display. It scans the data stream, modifying the contents of the work buffer as directed by the data stream.

If the data stream should be handled as an SNA SSCP-SLU data stream, only the new line (NL) order is processed. When an NL is encountered, nulls are inserted to fill the remainder of the line and position the current buffer address to the first position of the next line. Null, IFS, and IRS are treated as graphics and displayed as blank, \*, and ; respectively. Any other order encountered in the data stream causes a dash to appear at the current position in the buffer. When the write is completed, the cursor is positioned to the next available character location. This location is saved as the initial cursor position and used on a subsequent read operation.

If the data stream is not to be handled as an SNA SSCP-SLU data stream, the following orders within the data stream are processed:

**Start field**  
 Set buffer address  
 Insert cursor  
 Program tab  
 Repeat to address  
 Erase unprotected to address

While processing the data stream, if the BSC/SNA flag in bit 3 of the byte at address X'F1' in the KB/CRT IOB is set to 1 to indicate SNA, the microcode checks for invalid addresses specified in the following orders:

- Set buffer address.
- Repeat to address.
- Erase unprotected address.

The microcode also checks for invalid order length, regardless of the setting of the BSC/SNA flag. If an error occurs, external status code 17 is posted with an appropriate error code. The device buffer and the screen are unchanged. If no errors are detected during the write, the contents of the work buffer move to the refresh buffer and to the device buffer; the screen image changes.

| Mnemonic | Keyboard Op Code | Parameter1 | Parameter2 | Operation                                     |
|----------|------------------|------------|------------|---|
| KEYOP    | (X'83',          | BRa,       | BRb (4) )  | Keyboard/display write assist for the printer |

Parameter1 specifies a binary register that contains the address of the data stream. Parameter2 specifies a 4-byte double binary register that contains the information described in bytes 0-3.

| Byte | Bit          | Meaning   |
|------|--------------|---|
| 0    | 0            | Indicator of where to begin write.<br>0 = begin at cursor address.<br>1 = begin at current buffer address.  |
|      | 1-2          | Indicator of work buffer initialization.<br>00 = no initialization.<br>01 = initialize to null.<br>10 = initialize to device buffer contents.<br>11 = no initialization.<br><br>If initialize to null is specified, the cursor address and the buffer address are also set to zero before the write begins. |
|      | 3            | Reset MDT indicator.<br>0 = Do not reset modified data tags.<br>1 = Reset modified data tags before write.  |
|      | 4-7          |   |
| 1    | and<br>0-7   | Length of data stream.  |
| 2-3  |              | Address of a printer control block of 11 bytes in the following format:   |
|      | <b>Bytes</b> | <b>Contents</b>   |
|      | 1-2          | Device buffer address   |
|      | 3-4          | Work buffer address   |
|      | 5-6          | Current buffer address  |
|      | 7-8          | Current cursor position   |
|      | 9            | EBCDIC-Internal translation table address   |
|      | 10           | Flag/translation table page number  |
|      | 11           | Internal-EBCDIC translation table address   |

The microcode processes an IBM 3270 write data stream directed to the printer. It scans the data stream, modifying the contents of the work buffer as directed by the data stream.

The following orders within the data stream are processed:

- Start field
- Set buffer address
- Insert cursor (cursor position is stored into the control block following the write assist if it completes successfully)
- Program tab
- Repeat to address
- Erase unprotected to address

While processing the data stream, if the BSC/SNA flag in bit 3 of the tenth byte of the printer control block is set to 1 to indicate SNA, the microcode checks for invalid addresses specified in the following orders:

- Set buffer address
- Repeat to address
- Erase unprotected address

The microcode also checks for invalid order length, regardless of the setting of the BSC/SNA flag. If an error occurs, external status code 17 will be posted with an appropriate error code. The device buffer will be unchanged. If no errors are detected during the write, the contents of the work buffer are moved to the device buffer. If an insert cursor order is detected in the data stream, the cursor address in the device block is set to the cursor address indicated.

| Mnemonic | Keyboard Op Code | Parameter1 | Parameter2 | Operation                              |
|----------|------------------|------------|------------|--|
| KEYOP    | (X'84',          | BRa,       | BRb)       | Keyboard/display null non-print fields |

Parameter1 specifies a binary register that contains the address of an IBM 3270 printer control block of 11 bytes in the following format:

| Bytes | Contents                                  |
|-------|---|
| 1-2   | Device buffer address                     |
| 3-4   | Work buffer address                       |
| 5-6   | Current buffer address                    |
| 7-8   | Current cursor position                   |
| 9     | EBCDIC-Internal translation table address |
| 10    | Flag/translation table page number        |
| 11    | Internal-EBCDIC translation table address |

Parameter2 specifies a binary register that contains the following:

| Byte | Meaning   |
|------|---|
| 0    | 0 = null all characters in non-display fields.<br>Not 0 = null all characters in non-display fields except forms feed characters. |
| 1    | Not used.   |

The microcode processes an IBM 3270 print image. It copies the device buffer byte for byte into the work buffer, storing a null (X'00') into the work buffer for each attribute encountered in the device buffer. In addition, if an attribute that specifies a nondisplay field is encountered, it stores nulls into the work buffer for each device buffer data byte in the field with the following exception: If the high-order byte of BRb is nonzero and a byte in the nondisplay field is a forms feed character, that

forms feed character is not changed. The device buffer is not modified. This enables the printer software to suppress a print line that contains only attributes, nonprint fields, and nulls.

| Mnemonic | Keyboard Op Code | Parameter1 | Operation                                     |
|----------|------------------|------------|---|
| KEYOP    | (X'85',          | BRa)       | Keyboard/display erase all unprotected assist |

Parameter1 specifies a binary register that contains either:

- X'0000' if the operation is directed to a display.
- the address of an IBM 3270 printer control block of 11 bytes in the following format if the operation is directed to a printer.

| Bytes | Contents                                  |
|-------|---|
| 1-2   | Device buffer address                     |
| 3-4   | Work buffer address                       |
| 5-6   | Current buffer address                    |
| 7-8   | Current cursor position                   |
| 9     | EBCDIC-Internal translation table address |
| 10    | Flag/translation table page number        |
| 11    | Internal-EBCDIC translation table address |

The following functions are performed in the device buffer in the order given:

1. The unprotected buffer character locations are set to nulls in the device buffer.
2. The MDTs for all unprotected fields are set to 0.
3. If this KEYOP is directed to a screen, the contents of the device buffer are moved into the work buffer and onto the screen. The cursor is repositioned to the first character location in the first unprotected field of the buffer. If no unprotected fields exist, the cursor is positioned to buffer location 0. The input inhibited condition is cleared.
4. If this KEYOP is directed to a printer, the contents of the device buffer are moved into the work buffer. The current cursor position in the control block is set to point to the first character location in the first unprotected field of the buffer. If no unprotected fields exist, the current cursor position is set to point to buffer location 0.

| Mnemonic | Keyboard Op Code | Parameter1 | Parameter2 | Operation  |
|----------|------------------|------------|------------|--|
| KEYOP    | (X'86',          | BRa,       | BRb)       | Keyboard/display read buffer assist for SNA SSCP-SLU owned session |

Parameter1 specifies a binary register that contains the address of the data stream buffer. Parameter2 specifies a binary register that contains the maximum data stream length, in the following format:

| Byte | Bits | Meaning                     |
|------|------|-----------------------------|
| 0    | 0-3  | Must be zero.               |
|      | 4-7  |                             |
|      | and  |                             |
| 1    | 0-7  | Maximum data stream length. |

The microcode scans the device buffer beginning at the initial cursor address (saved from the last write assist) for a maximum of 256 bytes or to the end of the buffer, whichever occurs first. It builds an IBM 3270 data stream from the bytes scanned, with nulls suppressed.

If the length of the data stream to be generated is greater than the maximum length specified in BRb, external status code 17 is posted.

At successful completion of the operation, the microcode stores the length of the data stream generated in the 1921st and 1922nd bytes of the device buffer.

| Mnemonic | Keyboard |            | Operation                                       |
|----------|----------|------------|---|
|          | Op Code  | Parameter1 |   |
| KEYOP    | (X'87',  | BRa)       | Keyboard/display indicator and keyboard control |

Parameter1 specifies a binary register that contains the information described below:

| Byte | Bit | Meaning  |
|------|-----|--|
| 0    | 0-7 | Must be 0.   |
| 1    | 0   | Must be 0.   |
|      | 1   | Insert mode indicator.<br>1 = turn off.<br>0 = no change.                        |
|      | 2-3 | Hard lock.<br>00 = no change.<br>X1 = turn on.<br>10 = turn off.                 |
|      | 4-5 | Input inhibited indicator.<br>00 = no change.<br>X1 = turn on.<br>10 = turn off. |

| Byte | Bit | Meaning   |
|------|-----|---|
|      | 6-7 | System available indicator.<br>00 = no change.<br>X1 = turn on.<br>10 = turn off. |

**Note:** X denotes either a 1 or a 0.

This operation manipulates the status of the system available indicator, the input inhibited indicator, the insert mode indicator, and the keyboard hard lock. When an indicator is turned on, a solid rectangle appears in the corresponding indicator position on the screen. When an indicator is turned off, a dash appears in the corresponding indicator position on the screen. The indicators are located to the right of the screen on the following rows:

| Indicator                  | Row |
|----------------------------|-----|
| System available indicator | 10  |
| Insert mode indicator      | 12  |
| Input inhibited indicator  | 14  |

| Mnemonic | Keyboard Op Code | Operation                                    |
|----------|------------------|--|
| KEYOP    | (X'88')          | Keyboard/display clear and initialize screen |

This operation clears the device buffer to nulls, blanks the screen, positions the cursor to character location 0, sets the current buffer address to character location 0, resets the MDT bits, and sets the initial cursor address (used for SSCP-SLU owned SNA session) to 0.

#### *READMG (Read Magnetic Stripe Reader)*

| Mnemonic | Operand1 | Operand2 |
|----------|----------|----------|
| READMG   | (BRa,    | BRb)     |

where:

*Operand1* indicates a binary register that contains the number of bytes that are to be read, minus 1.

*Operand2* indicates a binary register that contains the address in main storage into which the bytes are read.

When a badge is inserted into a magnetic stripe reader, the badge characters are read into a buffer in the reader. External status condition 11 occurs for the partition associated with the reader. Only one partition can be associated with one reader. If the partition is background, or if it is available to the loader, the badge data is ignored. If external status is already outstanding at the time of the status 11 condition, the IBM 5280 waits until the current external status condition is cleared before it notifies the partition.

Once a badge is inserted into the reader and the reader buffer accepts the badge data, no other badge data is accepted until the buffer data is read or reset by a READMG or RSTMG instruction. After the execution of a READMG instruction, the reader is automatically reset to enable the reader to accept another badge. The RSTMG instruction is used only when you wish to ignore the current badge data, or at the beginning of a program to ready the reader for the first badge.

Magnetic stripe data consists of a string of from 3 to 128 bytes or characters. The first character of a data group must be a start of message (SOM) control character. The next-to-the-last character must be an end-of-message (EOM) control character. The last character must be a longitudinal redundancy check (LRC) control character of even parity for the entire data group. Any character can be issued in the other bytes except an EOM character.

The format of the reader characters is as follows:

| Name        | Bit | Meaning if 1  |
|-------------|-----|---|
| Device flag | 0   | A magnetic stripe reader is installed.  |
| Error flag  | 1   | One of the following conditions has occurred: <ul style="list-style-type: none"> <li>● Parity error</li> <li>● LRC error</li> <li>● EOM missing</li> <li>● Improper badge insertion or removal</li> <li>● Speed error</li> <li>● Buffer address overflow</li> </ul> |
| LRC flag    | 2   | LRC character.  |
| Parity bit  | 3   | Odd parity for bits 4-7.  |
| Data        | 4-7 | If hex 0 to 9, this is a data character.<br>If hex B, this is an SOM character.<br>If hex F, this is an EOM character.  |

If any byte has an error, the error flag is set in all bytes so your program can find the error by checking only the first byte. If this flag is set, the data is invalid and the badge must be reinserted.

The device flag should be set on for each data byte in the stripe when the READMG instruction is executed after the external status 11. If the program executes a READMG instruction when status 11 has not occurred, this bit is on if a reader is installed or off if a reader is not installed.

*Examples:*

| Reader Character | Description      |
|------------------|------------------|
| 10001011         | An SOM character |
| 10011111         | An EOM character |
| 10100000         | An LRC character |
| 10011001         | A data character |

### *RSTMG (Reset Magnetic Stripe Reader)*

#### **Mnemonic**

#### **RSTMG**

The RSTMG instruction enables the magnetic stripe reader to read a badge.

This instruction should be included at the beginning of a program that requires magnetic stripe data. This ensures that the reader is ready to accept data from a badge.

Although the magnetic stripe reader is automatically reset after the execution of each read (READMG) instruction, the RSTMG instruction may be required in the following instance. Once a badge is inserted into the reader, the IBM 5280 will not accept data from a second badge until the current badge data is read by the READMG instruction, or the reader is reset by an RSTMG instruction. If a RSTMG instruction is executed, the current badge data is ignored and the reader is ready to accept another badge.

### *REPFLD (Replace Field on Screen)*

#### **Mnemonic**

#### **REPFLD**

The REPFLD instruction uses the contents of three system binary registers as parameters. Parameter1 is taken from BR19, parameter2 is taken from BR20, and parameter3 is taken from BR21. When an external status condition occurs during the processing of an ENTR, the IBM 5280 places the following information, about the last field processed, into the system binary registers:

| <b>Register</b> | <b>Information</b>   |
|-----------------|--|
| BR19            | Address of the beginning of the field in the current record buffer.                      |
| BR20            | Absolute address of the beginning of the field in the keyboard/<br>display refresh area. |

BR21 Length of the field, minus 1, and character set specifications, as follows:

| Bits | Meaning  |
|------|--|
| 0-3  | Character set<br>0000 = picture check field<br>0001 = alphabetic shift field<br>0010 = numeric shift field<br>0011 = hex field<br>0100 = special characters field<br>0101 = format level zero field<br>1001 = alphabetic-only field<br>1010 = numeric-only field<br>1011 = digits-only field<br>1100 = special-characters-only field |
| 4    | 1 = signed numeric field   |
| 5-15 | Length minus 1   |

When the REPFLD instruction is executed, the number of bytes indicated by BR21 are moved from main storage, starting at the relative address stored in BR19, to the absolute address in the screen refresh area stored in BR20.

If BR21 specifies a signed numeric field, or a digits-only field or numeric-only field that is not signed numeric, and the rightmost byte of the bytes moved is a hex D0 through D9, a minus sign is displayed in the sign position of the field, to the right of the rightmost character. Otherwise the sign position is blank.

The data is translated and displayed as for a CRTMM instruction that specifies the S for the attributes parameter. (See *Move Bytes Between Storage and Screen*, later in this chapter.)

REPFLD can be used when data is entered, then calculated or changed in the current record buffer, then redisplayed.

*Example:* Enter 2 fields: Hours worked + Rate of pay. The program multiplies them to find Gross pay. An RG (return to program) bit is included at end of the field for Gross pay (a bypassed field since the program computes it). Binary register BR19, BR20, and BR21 are set to the field for Gross pay when the RG external status occurs. The subroutine for the RG condition computes the Gross pay, and places it in the I/O buffer in the Gross pay field, then issues REPFLD to display the computed value.

This operation is not meaningful for IBM 3270 mode.

#### *RTIMER (Read Elapsed Time Counter)*

| Mnemonic | Operand |
|----------|---------|
| RTIMER   | (BRn)   |

where:

*Operand* specifies a binary register that contains the address of a 3-byte storage area into which a timer value is read.

The elapsed time counter is an optional feature of the 5280. It measures elapsed real time and can be used to compute production statistics. Approximately every 1.6 seconds the timer overflows. Each time it overflows, a 2-byte counter in the system control block is incremented. The RTIMER instruction reads the current value ( $\pm .05$  seconds) of the counter and the interval timer into the 3-byte storage area pointed to by the operand binary register. The high order two bytes are read from the counter in the system control block. The interval timer value is read into bits 4-7 of the low-order byte; bits 0-3 are always zero.

## DATA MOVEMENT INSTRUCTIONS

The 5280 assembler language provides instructions to:

- *Load* a register with a constant or with the contents of another register or storage.
- *Store* a constant or the contents of a register.
- *Convert* binary and EBCDIC data.
- *Exchange* the contents of two locations.
- *Move bytes* of a specified length to storage or a screen.
- *Move formatted data* to storage or a screen.

For data movement instructions, registers can be specified by the register number or label. Storage locations can be specified by a label or base displacement address.

### Load Binary Register

One or two bytes of a binary register can be loaded from another binary register or from a storage address. The register can also be loaded with a constant or with the address of a labeled area. A binary register always contains binary notation with no sign. The format for loading a binary register is as follows:

| Result | Operand  |
|--------|--|
| BRa =  | $\left\{ \begin{array}{l} \text{BRb}[(1)] \\ \text{storage} \\ \text{constant} \end{array} \right\}$ |

where:

*Result* indicates the binary register into which data is loaded.

*Operand* specifies the data to be loaded into the result binary register. The operand may be 1 or 2 bytes in length. If the operand is 1 byte, it is loaded into the rightmost byte of the result register, and the leftmost byte of the result register is set to zero. The operand is unchanged by the operation.

### *Load with Contents of Binary Register*

| <b>Result</b> | <b>Operand</b> |
|---------------|----------------|
| BRa           | = BRb [(1)]    |

The contents of the operand register are loaded into the result register. If the operand register specifies a length of 1-byte (BRb(1)), the contents of the rightmost byte of the operand register are loaded into the rightmost byte of the result register, and the leftmost byte of the result register is set to zero. If the result register specifies a length of 1-byte (BRa(1)), the contents of the rightmost byte of the operand register are loaded into the rightmost byte of the result register, and the leftmost byte of the result register is unchanged. It is invalid to specify a 1-byte length for both registers.

### *Load with Labeled Storage*

| <b>Result</b> | <b>Operand</b> |
|---------------|----------------|
| BRa           | = label [(1)]  |

The labeled byte and the next consecutive byte are loaded into the binary result register. If a length of 1 is specified (label(1)), only the labeled byte is loaded into the rightmost byte of the binary register; the leftmost byte of the register is set to zero. If the length is omitted, the declared length of the label is used; a length greater than two bytes is invalid.

### *Load with Base Displacement Storage*

| <b>Result</b> | <b>Operand</b>       |
|---------------|----------------------|
| BRa           | = [disp] ([len],BRb) |

When this instruction is executed, the displacement (disp) is added to the contents of the binary register (BRn) to find the relative address of the leftmost byte specified by length (len). The contents of this 1 or 2 bytes are moved into the result register. The data in the storage location remains unchanged. If the length is omitted, 1-byte is used.

### *Load with a Constant*

| <b>Result</b> | <b>Operand</b> |
|---------------|----------------|
| BRa           | = constant     |

The constant specified by the operand is loaded into the binary register. Any constant less than 16 bits in length is padded on the left with zeros.

## Load Decimal Register

One to 16 bytes may be loaded into a decimal register from another decimal register or from a storage location. A signed constant may be loaded into the full register, or a 1-byte constant may be loaded into a specified offset into the register. Data in a decimal register is represented in EBCDIC notation.

The format to load a decimal register is as follows:

| Result | Operand   |
|--------|---|
| Ra =   | $\left\{ \begin{array}{l} \text{Rb} \\ \text{storage} \\ \pm \text{constant} \\ \text{constant} \end{array} \right\}$ |

### *Load with Decimal Register*

| Result | Operand |
|--------|---------|
| Ra =   | Rb      |

The contents of the operand register (Rb) replace the contents of the result register (Ra). The operand register remains unchanged.

### *Load with Labeled Storage*

| Result | Operand       |
|--------|---------------|
| Ra =   | label [(len)] |

The result register is set to blanks (hex 40s). Then the bytes beginning at label are loaded into the result register.

If length (len) is specified, the specified number of bytes are loaded into the decimal register and right adjusted.

If length is not specified, the number of bytes assigned as the length of the label (by the .DC control statement that declared the label) are loaded into the decimal register and right adjusted. A length greater than 16 is invalid.

**Note:** The label must be assigned to a storage location within the first 32 K of storage.

Example:

**Statements**

**Registers**

.DC LABEL=QTY LEN=10 INIT=4455667788;

LOADX: RX=QTY(5); Length specified

RX ▯ ▯ ▯ ▯ ▯ ▯ ▯ ▯ ▯ ▯ ▯ 4 4 5 5 6

LOADY: RY=QTY ; Length omitted

RY ▯ ▯ ▯ ▯ ▯ ▯ 4 4 5 5 6 6 7 7 8 8

*Load with Base Displacement Storage*

**Result      Operand**

Ra =          [disp] ([len], BRn)

The operand specifies a binary register that acts as a base address register. The displacement (disp), if specified, is added to the contents of the base address register. The result is taken as the address of the first byte of data to load into the result register. If length (len) is included, the number of bytes specified, starting at the first byte of the storage area, is moved into the decimal register and right adjusted. If length is omitted, it defaults to 1 byte. The result register is set to blanks (hex 40s) before the data is transferred. A length greater than 16 is invalid.

Example:

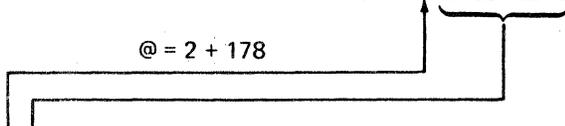
BASREG = 178

Storage Positions

|   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|
| 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 7 | 7 | 8 | 8 | 8 | 8 | 8 | 8 |
| 8 | 9 | 0 | 1 | 2 | 3 | 4 | 5 |
| 1 | 3 | 3 | 5 | 5 | 7 | 7 | 9 |

Data

@ = 2 + 178



The instruction REGX = 2(5,BASREG) would result in:

REGX ▯ ▯ ▯ ▯ ▯ ▯ ▯ ▯ ▯ ▯ ▯ 3 5 5 7 7

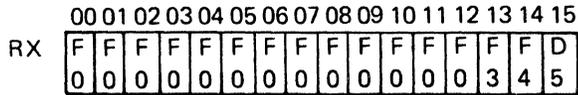
*Load with a Signed Constant*

**Result      Operand**

Ra =          ±constant

The value specified by the operand is loaded into the result decimal register and left filled with zeroes (hex F0s). The constant may be specified as decimal, hexadecimal or binary immediate data. The constant may also be an equated constant. The sign must be included in all cases regardless of the form of the constant. The maximum value the constant can be is +65535. If the value of the operand is positive, the rightmost byte of the decimal register has hex F in the zone portion. If the value of the operand is negative, the rightmost byte of the decimal register has hex D in the zone portion.

*Example:* The instruction `RX = -345` results in:



**Note:** If you attempt to load a register with a character, the register is loaded with the decimal EBCDIC equivalent of the character. For example, if the specified constant is `C'A'` (hex C1), the register is loaded with 13 leading zeros (hex F0s) and `F1F9F3` (decimal 193).

*Load with an Unsigned Constant*

|               |                |
|---------------|----------------|
| <b>Result</b> | <b>Operand</b> |
| Ra =          | constant       |

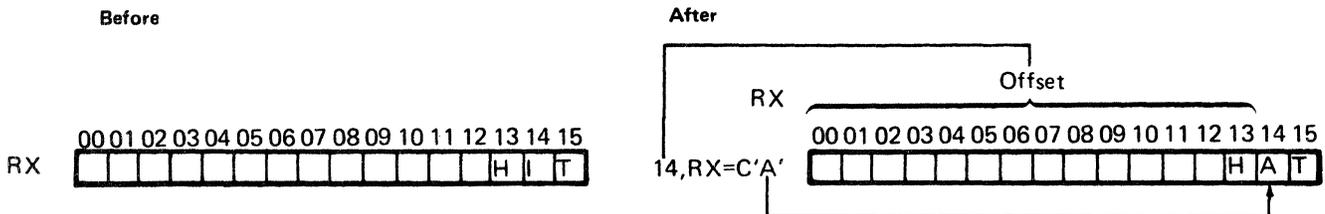
The value of the constant, which must be in the range 0-9, is loaded into the rightmost byte of the result decimal register. The remaining high-order bytes are filled with blanks.

*Insert a 1-Byte Constant*

|               |                |
|---------------|----------------|
| <b>Result</b> | <b>Operand</b> |
| [offset],Ra = | constant       |

The constant specified by the operand is inserted into the decimal result register at the byte specified by offset. The constant may be specified in any of the forms described under *Constant Specifications* in this chapter.

*Example:*



**Store at a Labeled Address**

One or more bytes of a decimal or binary register can be stored into a labeled storage area. Also, a 1-byte constant can be stored into a labeled storage byte. See *Addressing Methods* under *Partitions* in Chapter 1 for more information about labeled addressing.

Data is stored into a labeled address using the following format:

**Result**                      **Operand**

label [(len)] =  $\left\{ \begin{array}{l} Rn \\ BRn \\ \text{constant} \end{array} \right\}$

where:

*Result* indicates the label of the storage location into which data will be stored.

*Operand* indicates data to store.

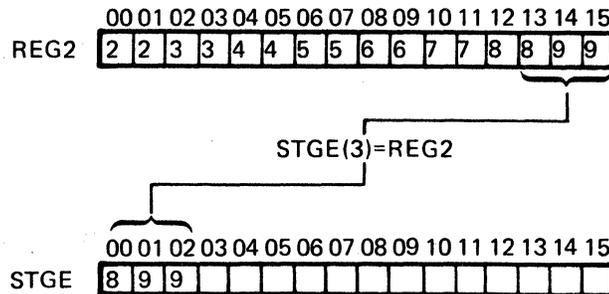
**Store a Decimal Register**

**Result**                      **Operand**

label [(len)] = Rn

If a length (len) is included, the rightmost number of bytes specified by the length are moved into the labeled storage area. The storage area begins at the byte assigned to label and continues for the number of bytes specified by length. If length is omitted, it defaults to the length specified by the .DC control statement that assigned the label. The labeled location must be within the first 32 K bytes of a partition.

*Example:*



**Store a Binary Register**

**Result**                      **Operand**

label [(len)] = BRn

If the operand is full binary register, the contents of the two register bytes are stored into the byte specified by label, and the next consecutive byte. The result storage location must not be specified as more than 2 bytes in length. If the length is omitted, the declared length of the labeled area is used.

### Store a Constant

| Result  | Operand  |
|---------|----------|
| label = | constant |

The 1-byte constant specified by the operand is loaded into the labeled byte. Only the one labeled byte is changed. The constant may be specified in any of the forms described under *Constant Specifications* at the beginning of this chapter.

### Store at Base Displacement Address

Data can be stored at a base displacement address. Base displacement addressing is described in Chapter 1 under *Storage*.

The format for the instructions to store data at a base displacement address is as follows:

| Result                | Operand   |
|-----------------------|---|
| [disp] ([len], BRn) = | $\left\{ \begin{array}{l} Rn \\ BRn \end{array} \right\}$ |
| [disp] (BRn) =        | constant  |

where:

*Result* indicates the storage area into which data is stored. The binary register acts as a base address register. The contents of the base address register (BRn) are added to the displacement (0-255), and the result is the address of the first of the storage bytes into which data is stored. The number of bytes stored is determined by the length specified (len).

If the result address specifies a location outside the partition, a program check error (hex 01) occurs.

*Operand* indicates the register from which data is moved, or a 1-byte constant.

### Store a Decimal Register

| Result                | Operand |
|-----------------------|---------|
| [disp] ([len], BRn) = | Rn      |

The contents of the decimal register specified by the operand are stored into the location indicated by result. The contents of the base register (BRn) are added to the displacement (disp), and the sum is taken as the address into which the data is stored. If length is included (len), the rightmost number of register bytes specified by the length are moved into the storage area. The optional length defaults to 1 byte. A length greater than 16 is invalid.

### Store a Binary Register

| Result                | Operand |
|-----------------------|---------|
| [disp] ([len], BRa) = | BRb     |

The contents of the binary register specified as operand (BRb) are stored into the location specified by result. The displacement (disp) is added to the contents of the binary base register (BRa), and the result is taken as the rightmost byte of storage into which the contents of the binary register are stored. If the length of one is specified, the contents of the low-order byte of the operand register are stored.

### Store a Constant

| Result         | Operand  |
|----------------|----------|
| [disp] (BRa) = | constant |

The displacement is added to the address in BRa, and the 1-byte constant specified by the operand is stored in the byte at the resulting address. Do not specify length for this operation, and do not include a comma to the left of the binary register as is required for other base displacement addresses.

### Exchange Data

Data may be exchanged between two registers or between a storage area and a binary register. The contents of a binary register may be exchanged with the contents of another binary register or a storage location. The contents of a decimal register may be exchanged with the contents of another decimal register. The format for the exchange operation is as follows:

| Result | Operand                                 |
|--------|---|
| BRa    | $\langle \Rightarrow \rangle$ { BRb }   |
| Ra     | $\langle \Rightarrow \rangle$ { label } |
|        | { Rb }                                  |

where:

*Result* indicates the register into which the data from the operand register is placed upon completion of the operation. Data originally in the result register is placed in the operand register.

*Operand* indicates the register or storage area that exchanges contents with the result register.

### *BRa <=> (Binary Register Exchange)*

| Result |     | Operand          |
|--------|-----|------------------|
| BRa    | <=> | { BRb<br>label } |

The contents of the operand, which may be a binary register or a 2-byte labeled storage area, are placed into the result register (BRa). The original contents of the result register are placed into the operand register.

### *Ra <=> (Decimal Register Exchange)*

| Result |  | Operand |
|--------|--|---------|
| Ra <=> |  | Rb      |

The contents of the operand register are placed into the result register, and the original contents of the result register are placed into the operand register.

### **Convert Register Contents**

The contents of a register or labeled storage area can be converted from binary to EBCDIC or from EBCDIC to binary. The convert operation both converts and moves the data. Although all data is physically stored as binary bits, the bits may represent the binary value or the EBCDIC value of the data. Each byte of decimal EBCDIC data is divided into a zone portion and a digit portion. Therefore, to express the value of one byte of binary data requires 1 byte of EBCDIC or 2 bytes of hexadecimal EBCDIC. EBCDIC data, however, can be displayed on the screen. Binary data must be converted to EBCDIC in order to be displayed. For example, B'11111000' converts to EBCDIC F2F4F8 with decimal conversion and can be displayed as 248. With hexadecimal conversion, B'11111000' converts to EBCDIC C6F8 and can be displayed as F8. The format of the convert operations is as follows:

| Mnemonic | Result          | Operand        |
|----------|-----------------|----------------|
|          | Rn =            | BRn            |
| BINDEC   | (Rn             | ,label [(2)])  |
| BINHEX   | (label [(len)], | BRn)           |
| HEXBIN   | (BRn ,          | label [(len)]) |
|          | BRn =           | Rn             |
| DECBIN   | (label [(2)],   | Rn)            |

where:

*Result* indicates the register or labeled storage location into which the converted data is moved.

*Operand* indicates the register or storage location that holds data to be converted and placed into result register or storage location. The operand remains unchanged.

*Rn = (Convert Binary to Decimal)*

| Result | Operand |
|--------|---------|
|--------|---------|

|      |     |
|------|-----|
| Rn = | BRn |
|------|-----|

The contents of the operand register (BRn) are converted to decimal, and the decimal value is loaded into the decimal register specified as result (Rb). The decimal value is represented as positive in the result register.

The operand register contains the original binary notation upon completion of the operation.

*BINDEC (Convert Binary Storage to EBCDIC)*

| Mnemonic | Result | Operand |
|----------|--------|---------|
|----------|--------|---------|

|             |  |              |
|-------------|--|--------------|
| BINDEC (Rn, |  | label [(2)]) |
|-------------|--|--------------|

The contents of the 2-byte labeled storage area are converted from binary to EBCDIC and placed into the decimal register.

The storage area must be 2 bytes in length. You may specify the length in the instruction, or you may omit the length if the area was declared as 2 bytes in length.

*BINHEX (Convert Binary to EBCDIC Storage)*

| Mnemonic | Result | Operand |
|----------|--------|---------|
|----------|--------|---------|

|               |  |        |
|---------------|--|--------|
| BINHEX (label | [( $\left. \begin{matrix} 2 \\ 4 \end{matrix} \right\}$ )] | , BRn) |
|---------------|--|--------|

The contents of the operand binary register (BRn) are converted to an equivalent hexadecimal EBCDIC, and the hexadecimal EBCDIC value is loaded into the 2-byte or 4-byte storage area specified by the label.

The length of the storage area may be specified in the instruction. If the length is omitted, it defaults to the length specified by the .DC control statement that labeled the storage area. Only 2 or 4 are valid for length. If a length of 2 is specified, only the rightmost byte of BRn is used.

The contents of the binary register are converted to 2 or 4 (depending on the length of the storage area) hexadecimal EBCDIC characters and stored into the 2 or 4 storage bytes.

*HEXBIN (Convert Hexadecimal EBCDIC Storage to Binary)*

| Mnemonic | Result      | Operand      |
|----------|-------------|--------------|
| HEXBIN   | (BRn, label | [( { 2 } )]) |

The hexadecimal EBCDIC values at the storage location specified by the label are converted to binary and loaded into the binary register specified by BRn.

The length of the storage area may be specified in the instruction, or it may default to the length specified by the .DC statement that labeled the area. Only 2 or 4 is valid for the length.

If the length of the storage area is four bytes, the 4 hexadecimal EBCDIC characters in the storage area are converted to 2 bytes of binary notation and placed into the binary register specified by result. If the length of the storage area is 2 bytes, the 2 hexadecimal EBCDIC characters are converted to 1 byte of binary notation and placed into the rightmost byte of the binary register; the leftmost byte remains unchanged. If the storage area contains any values other than '0-9' or 'A-F', I119 is set and the register contents are unpredictable.

*BRn = (Convert Decimal to Binary)*

| Result | Operand |
|--------|---------|
| BRn =  | Rn      |

The contents of the operand register are converted to binary notation, and the binary notation is loaded into the result register. If the contents of the decimal register are negative, or if the binary register overflows, the overflow indicator (I124) is set on.

*DECBIN (Convert EBCDIC to Binary Storage)*

| Mnemonic | Result        | Operand |
|----------|---------------|---------|
| DECBIN   | (label [(2)], | Rn)     |

The contents of the decimal register are converted to binary and stored in the 2-byte labeled storage area specified by result.

The length of the storage area must be 2 bytes. You may specify the length in the instruction, or you may omit the length if the area was declared as 2 bytes in length.

## Move Bytes Between Decimal Registers

One or more bytes can be moved from one decimal register to another. The bytes can be moved to corresponding positions or to different positions in the receiving register.

The format to move the partial contents between decimal registers is as follows:

| Mnemonic           | Result | Operand | Offset  | Length |
|--------------------|--------|---------|---------|--------|
| {<br>MOFF<br>MVER} | (Ra,   | Rb      | [,0-15, | 1-16]) |

where:

*Results* indicates the decimal register to which the data is moved.

*Operand* indicates the decimal register from which the data is moved.

*Offset* specifies the offset of the leftmost byte to which data is moved. Offset defaults to zero.

*Length* specifies the number of bytes (1-16) that are moved. Length defaults to 1.

### MOFF (Move to an Offset)

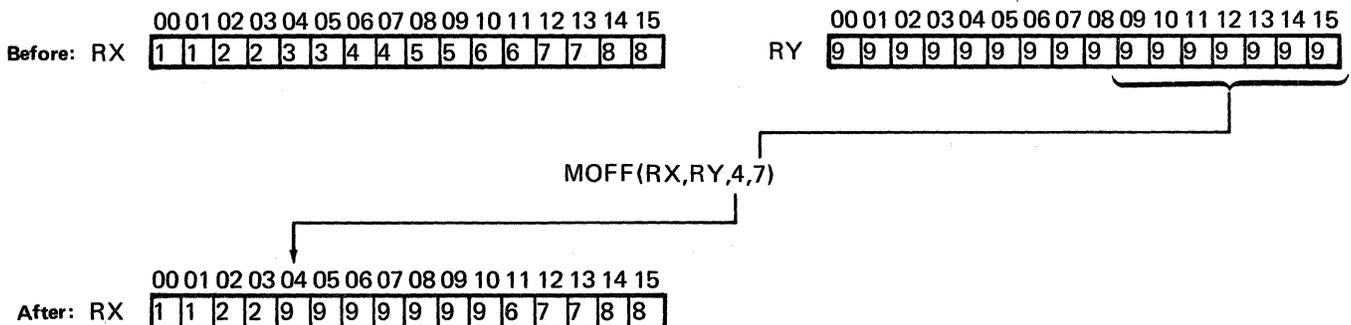
| Mnemonic | Result | Operand | Offset | Length  |
|----------|--------|---------|--------|---------|
| MOFF     | (Ra,   | RB      | [,0-15 | ,1-16]) |

The rightmost number of bytes specified by length are moved from the operand register to the result register. The data is moved from left to right and placed in the result register at the byte specified by offset.

The offset applies only to the result register (Ra), so the move is not limited to corresponding byte positions.

**Note:** If the sum of offset and length is greater than 16, bytes are moved into the register following the result register.

*Example:*



**MVER (Move to Corresponding Bytes)**

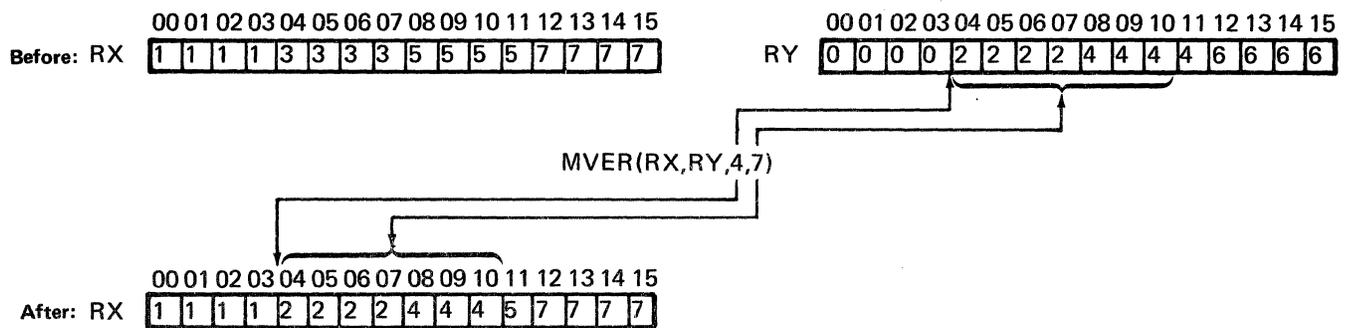
| Mnemonic | Result | Operand | Offset | Length  |
|----------|--------|---------|--------|---------|
| MVER     | (Ra,   | Rb      | [,0-15 | ,1-16]) |

The number of bytes specified by length are moved from the operand register to the result register. The data is moved from left to right, starting at the offset in the operand register and placed in corresponding positions, starting at the offset, in the result register.

The offset applies to both the operand and the result register, so the data is always moved to corresponding positions in the result register.

**Note:** If the sum of offset and length is greater than 16, bytes are moved into the register following the result register.

*Example:*



**Move Bytes in Storage**

From one to 256 bytes can be moved to another storage address within the same partition.

| Mnemonic                | Result | Operand | Length |
|-------------------------|--------|---------|--------|
| { MVC<br>MVCR<br>MVCV } | (BRa,  | BRb,    | 1-256) |

where:

*Result* indicates a binary register that contains the address, relative to the start of the partition of the leftmost byte into which data is moved.

*Operand* specifies a binary register that contains the address of the leftmost byte from which data is moved.

*Length* specifies the number of bytes to move. The specification is a decimal constant (1-256).

### MVC (Move Characters)

| Mnemonic | Result | Operand | Length |
|----------|--------|---------|--------|
| MVC      | (BRa,  | BRb,    | 1-256) |

The number of consecutive bytes specified by length are copied from the storage location, beginning at the byte specified by the contents of the operand register (BRb), into the location beginning with the byte specified by the contents of the result register (BRa).

Data is moved starting with the leftmost byte and continuing to the rightmost byte.

#### Example:

```
.DC LABEL=FLD1@ TYPE=BIN INIT=ADDR(FLD1);  
.DC LABEL=FLD2@ TYPE=BIN INIT=ADDR(FLD2);
```



### MVCR (Move Characters Right to Left)

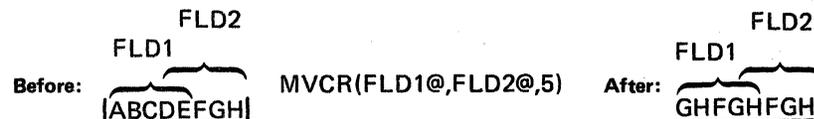
| Mnemonic | Result | Operand | Length |
|----------|--------|---------|--------|
| MVCR     | (BRa,  | BRb,    | 1-256) |

The number of bytes specified by length are copied from the storage location indicated by the operand register (BRb) into the location specified by the result register (BRa).

Data is copied starting with the rightmost of the operand bytes specified by length and continuing to the leftmost byte. This instruction is useful for moving data into overlapped fields.

#### Example:

```
.DC LABEL=FLD1@ TYPE=BIN INIT=ADDR(FLD1);  
.DC LABEL=FLD2@ TYPE=BIN INIT=ADDR(FLD2);
```



**MVCV (Move Characters Reverse)**

| Mnemonic | Result | Operand | Length |
|----------|--------|---------|--------|
| MVCV     | (BRa,  | BRb,    | 1-256) |

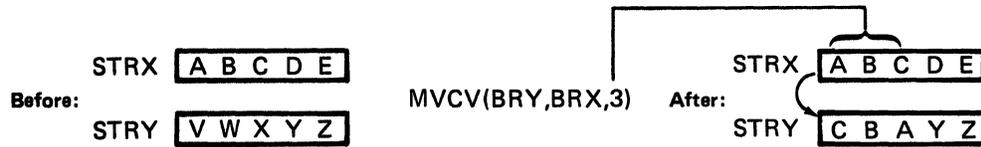
The number of bytes specified by length are copied from the storage location specified by the contents of the operand register (BRb) into the location specified by the contents of the result register (BRa).

While the data is copied from one location to the other, the order of the bytes is reversed.

Data is copied from the location specified by the operand register (BRb) from left to right; the data is copied into the address specified by the result register (BRa) from right to left.

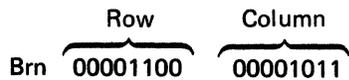
*Example:*

```
.DC LABEL=BRX TYPE=BIN INIT=ADDR(STRX);
.DC LABEL=BRY TYPE=BIN INIT=ADDR(STRY);
```



**Move Bytes Between Storage and Screen**

Data bytes may be moved directly between the display screen refresh area and main storage. No formatting is done. Bytes are moved to or from any position in the refresh area. A binary register is specified to contain the absolute address within the refresh area, or the position on the screen. If a screen position is used, the high-order byte contains the row number and the low-order byte contains the column number.



If row 0 is indicated, the bytes are moved to the status line. If row 1 is specified, the bytes are moved to the top line of the screen, which is not displayed when the status line is being displayed. If the position specifies column 0, external status condition 10 (invalid operation) occurs.

Be careful to specify a valid screen position. If the position operands point outside your screen, they may point to an area of control information within the keyboard/display; this results in external status condition 10. Invalid position operands may also point to the area of another display screen; in this case, the move proceeds and overwrites the data on the other screen.

The formats of instructions to move data between storage and screen are as follows:

| Mnemonic | Result | Operand | Length | Attributes  |
|----------|--------|---------|--------|---|
| CRTMM    | (BRa,  | BRb,    | BRc    | [, { $\begin{matrix} \text{NC} \\ \text{S} \end{matrix}$ } ]) |
| MMCRT    | (BRa,  | BRb,    | BRc    | )   |

where:

*Result* is a binary register that contains the address of the first storage byte or screen position to which data is moved.

*Operand* is a binary register that contains the address of the first storage byte or screen position from which data is moved.

*Length* is a binary register that contains the number of bytes, minus one, to move and a flag to indicate whether BRa is a row, col specification or an absolute address specification. The number may be from one to the maximum number of positions on the screen.

*Attributes* may be included with the CRTMM instruction to determine the display of the bytes. They are described with the CRTMM operation description.

#### CRTMM (Move to Screen)

| Mnemonic | Result | Operand | Length | Attributes  |
|----------|--------|---------|--------|---|
| CRTMM    | (BRa,  | BRb,    | BRc    | [, { $\begin{matrix} \text{NC} \\ \text{S} \end{matrix}$ } ]) |

The number of bytes indicated by the length register (BRc) bits 1-15 are moved from storage to the screen. The bytes are moved beginning at the address indicated by the contents of the operand register (BRb) to the address or screen position specified by the contents of the result register (BRa). If bit 0 of BRc is 0, BRa is assumed to contain a row, col specification. If bit 0 of BRc is 1, BRa is assumed to contain an absolute address. If the attributes parameter is omitted in the instruction, the bytes are translated from EBCDIC notation to display codes (using a translate table in keyboard/display storage) before they are placed into the screen refresh buffer and displayed on the screen. Most data that is moved to the screen requires this translation so the proper display graphics appear on the screen. EBCDIC values from hex 20 through 3F are translated to display attributes that alter the appearance of the screen, such as blank and underscore. Therefore, any data that may contain display attributes, such as prompts, should be moved to the screen in this way.

If NC is specified for the attributes parameter, the bytes are not translated as they are moved from main storage to the screen refresh buffer. Data that has been moved directly from the screen refresh buffer can be returned in this way.

If the S is specified for the attributes parameter, the bytes are translated through the translate table; however, before translation all EBCDIC values in the attribute range (hex 20 through 3F) are changed to hex 1F. Hex 1F translates to a solid rectangle in the translate table. This allows actual data, which may contain hex values between hex 20 and 3F, to be moved to the screen without affecting the appearance of the screen. All hex values between 20 and 3F appear as solid rectangles on the screen; the EBCDIC data in main storage is not altered. The hex value position on the status line displays the hex value of the positions that are displayed as solid rectangles if they appear within a screen format field when an ENTR is being processed.

**MMCRT (Move to Storage)**

| Mnemonic | Result | Operand | Length |
|----------|--------|---------|--------|
| MMCRT    | (BRa,  | BRb,    | BRc)   |

The number of screen positions indicated by the length register (BRc) are moved from the screen to storage. The data is moved beginning with the location specified by the operand register (BRb) to storage, beginning at the address specified by the result register (BRa). The display on the screen remains unchanged.

*Example:* The following code moves data from the screen to storage locations labeled LETRS1, LETRS2, and NUMBS.

|   |  |  |                |  |                         |   |  |   |  |   |                 |   |  |               |           |   |  |   |  |   |  |   |  |  |                |        |         |        |     |       |       |
|---|--|--|----------------|--|-------------------------|---|--|---|--|---|-----------------|---|--|---------------|-----------|---|--|---|--|---|--|---|--|--|----------------|--------|---------|--------|-----|-------|-------|
| <pre>.DCLBR LABEL=TO@,FROM@,LNGTH; TO@ = ADDR(LETRS1); FROM@ = X'0302'; screen position row 3, column 2 LNGTH = 7-1; MMCRT(TO@,FROM@,LNGTH); moves 'ABC DEF' into LETRS1 TO@ = ADDR(LETRS2) + 4; FROM@ = X'0306'; screen position row 3, column 6 LNGTH = LENG(LETRS2) - 4; MMCRT(TO@,FROM@,LNGTH); moves 'DEF' into LETRS2 TO@ = ADDR(NUMBS) + 2; FROM@ = X'0502'; screen position row 5, column 2 LNGTH = 5-1; MMCRT(TO@,FROM@,LNGTH); moves 12345 into NUMBS</pre> | <table border="0"> <tr> <td></td> <td style="text-align: right;"><b>Columns</b></td> </tr> <tr> <td></td> <td style="text-align: right;">1 2 3 4 5 6 7 8 9 . . .</td> </tr> <tr> <td style="text-align: right;">1</td> <td></td> </tr> <tr> <td style="text-align: right;">2</td> <td></td> </tr> <tr> <td style="text-align: right;">3</td> <td style="text-align: center;">A B C     D E F</td> </tr> <tr> <td style="text-align: right;">4</td> <td></td> </tr> <tr> <td style="text-align: right;"><b>Rows</b> 5</td> <td style="text-align: center;">1 2 3 4 5</td> </tr> <tr> <td style="text-align: right;">6</td> <td></td> </tr> <tr> <td style="text-align: right;">7</td> <td></td> </tr> <tr> <td style="text-align: right;">8</td> <td></td> </tr> <tr> <td style="text-align: right;">9</td> <td></td> </tr> <tr> <td></td> <td style="text-align: right;"><b>Storage</b></td> </tr> <tr> <td style="text-align: right;">LETRS1</td> <td style="border: 1px solid black; padding: 2px;">ABC DEF</td> </tr> <tr> <td style="text-align: right;">LETRS2</td> <td style="border: 1px solid black; padding: 2px;">DEF</td> </tr> <tr> <td style="text-align: right;">NUMBS</td> <td style="border: 1px solid black; padding: 2px;">12345</td> </tr> </table> |  | <b>Columns</b> |  | 1 2 3 4 5 6 7 8 9 . . . | 1 |  | 2 |  | 3 | A B C     D E F | 4 |  | <b>Rows</b> 5 | 1 2 3 4 5 | 6 |  | 7 |  | 8 |  | 9 |  |  | <b>Storage</b> | LETRS1 | ABC DEF | LETRS2 | DEF | NUMBS | 12345 |
|   | <b>Columns</b>   |  |                |  |                         |   |  |   |  |   |                 |   |  |               |           |   |  |   |  |   |  |   |  |  |                |        |         |        |     |       |       |
|   | 1 2 3 4 5 6 7 8 9 . . .  |  |                |  |                         |   |  |   |  |   |                 |   |  |               |           |   |  |   |  |   |  |   |  |  |                |        |         |        |     |       |       |
| 1   |  |  |                |  |                         |   |  |   |  |   |                 |   |  |               |           |   |  |   |  |   |  |   |  |  |                |        |         |        |     |       |       |
| 2   |  |  |                |  |                         |   |  |   |  |   |                 |   |  |               |           |   |  |   |  |   |  |   |  |  |                |        |         |        |     |       |       |
| 3   | A B C     D E F  |  |                |  |                         |   |  |   |  |   |                 |   |  |               |           |   |  |   |  |   |  |   |  |  |                |        |         |        |     |       |       |
| 4   |  |  |                |  |                         |   |  |   |  |   |                 |   |  |               |           |   |  |   |  |   |  |   |  |  |                |        |         |        |     |       |       |
| <b>Rows</b> 5   | 1 2 3 4 5  |  |                |  |                         |   |  |   |  |   |                 |   |  |               |           |   |  |   |  |   |  |   |  |  |                |        |         |        |     |       |       |
| 6   |  |  |                |  |                         |   |  |   |  |   |                 |   |  |               |           |   |  |   |  |   |  |   |  |  |                |        |         |        |     |       |       |
| 7   |  |  |                |  |                         |   |  |   |  |   |                 |   |  |               |           |   |  |   |  |   |  |   |  |  |                |        |         |        |     |       |       |
| 8   |  |  |                |  |                         |   |  |   |  |   |                 |   |  |               |           |   |  |   |  |   |  |   |  |  |                |        |         |        |     |       |       |
| 9   |  |  |                |  |                         |   |  |   |  |   |                 |   |  |               |           |   |  |   |  |   |  |   |  |  |                |        |         |        |     |       |       |
|   | <b>Storage</b>   |  |                |  |                         |   |  |   |  |   |                 |   |  |               |           |   |  |   |  |   |  |   |  |  |                |        |         |        |     |       |       |
| LETRS1  | ABC DEF  |  |                |  |                         |   |  |   |  |   |                 |   |  |               |           |   |  |   |  |   |  |   |  |  |                |        |         |        |     |       |       |
| LETRS2  | DEF  |  |                |  |                         |   |  |   |  |   |                 |   |  |               |           |   |  |   |  |   |  |   |  |  |                |        |         |        |     |       |       |
| NUMBS   | 12345  |  |                |  |                         |   |  |   |  |   |                 |   |  |               |           |   |  |   |  |   |  |   |  |  |                |        |         |        |     |       |       |

## Move Formatted Data

The following instructions move data according to a specified edit format. The edit format is set up by the .FMT control statements, which specify:

- The storage areas to or from which data is moved (DCLBL)
- The length of the data (LEN)
- The editing of the data fields (EDIT)

The edit format can convert data to decimal, to binary, or to hexadecimal notation. Field punctuation, such as a fixed dollar sign, a floating dollar sign, a decimal point, sign control, and fill characters are specified by the format editing.

The instructions for formatted moves are as follows:

| Mnemonic | Operand1 | Format         | Operand2 | Operand3         |
|----------|----------|----------------|----------|------------------|
| REBF     | (BRa,    | { label<br>* } | )        |                  |
| WRBF     | (BRa,    | label          | [,BRb ]  | )                |
| WFMCR    | (BRa,    | label          | [,BRb ,  | { B<br>ADD } ] ) |

where:

*Operand1* specifies a binary register that contains an address.

*Format* indicates the formatting of the data.

| Entry | Description |
|-------|-------------|
|-------|-------------|

|       |   |
|-------|---|
| label | is the LABEL parameter of the .FMTST control statement that set up the edit format you wish to use. |
|-------|---|

|   |   |
|---|---|
| * | may be specified instead in the REBF instruction to indicate that formatting is data directed. See <i>Data Directed Formatting</i> in Chapter 2 for more information. |
|---|---|

*Operand2* is an optional entry that may be specified for WRBF or WFMCR and is discussed in the operation descriptions.

*Operand3* may be specified only for WFMCR and is discussed in the WFMCR operation description.

**REBF (Read to Format Specifications)**

| Mnemonic | Operand1 | Format           |
|----------|----------|------------------|
| REBF     | (BRa ,   | { label }<br>* ) |

Data bytes, beginning at the address specified by the contents of the binary register (BRa), are moved into the location specified by the DCLBL parameter of the edit format. The number of bytes moved is determined by the LEN parameter of the format, and editing is controlled by the EDIT parameter of the format.

In normal formatting, the instruction specifies the format label (LABEL parameter from the .FMTST statement). For data directed formatting, the instruction specifies an asterisk (\*) instead of a label.

*Example:* The following code moves data from a storage location pointed to by BUFR according to a format labeled FMT6.

Before: 01 02 03 04 05 06 07 08 09 10 11 12 13 14 15 16 17 18 19 20 21 22  
 BUFR T 1 7 0 6 4 C 4 \$ 1 4 . 2 9 1 4 7 8

```
.FMTST LABEL = FMT6 ; set up format
.FMTFLD DCLBL = ITEM# TYPE=DEC LEN = 8 COL = 1;
.FMTFLD DCLBL = ITEM$ TYPE=DEC LEN = 7 COL = 9 EDIT = $$,DP.02;
.FMTEND DCLBL = ONHAND TYPE=DEC LEN = 5 COL = 18;
```

REBF (BUFR, FMT6) ; move data

After:

|   |        |                  |
|---|--------|------------------|
| } | ITEM#  | T17064C4         |
|   | ITEM\$ | 0000000000001429 |
|   | ONHAND | 1478             |

**WRBF (Write Format to Storage)**

| Mnemonic | Operand1 | Format  | Operand2 |
|----------|----------|---------|----------|
| WRBF     | (BRa,    | [label] | [,BRb])  |

You can omit the format and include the binary register if you want to blank the buffer. This is the best way to blank a buffer that is longer than 256 bytes. The binary register can specify up to 65 535 bytes to blank. The 5280 does not check to ensure that you do not exceed your partition size.

The contents of the locations specified by the labeled format are moved to the address specified by the operand1 binary register (BRa). If the operand2 binary register (BRb) is included, the contents of this register are taken as the number of bytes to blank, beginning at the location specified by the first binary register, before the move is performed. If the second binary register is not included, no bytes of the storage location are blanked.

### WFMCRT (Write Format to Screen)

| Mnemonic | Operand1 | Format | Operand2 | Operand3         |
|----------|----------|--------|----------|------------------|
| WFMCRT   | (BRa ,   | label  | [,BRb ,  | { B<br>ADD } ] ) |

Data is moved to the screen, beginning at column 1 of the row specified by the low-order byte of the operand1 register (BRa). Data is moved from the locations specified by the labeled edit format, for the number of bytes specified by the format. The format also specifies any punctuation that should appear on the screen, such as a dollar sign, decimal point, or minus sign. The format must not use more than 200 screen positions. If row 0 is specified, data is moved to the status line; if row 1 is specified, data is moved to the extra line. If operand2 specifies a binary register (BRb), the contents of this register are taken as the number (1-200) of screen positions to alter before the formatted data is moved to the screen. Operand3 specifies how to alter the screen.

If B is coded for operand3, all characters on the screen between the data fields that are defined in the edit format are blanked for the number of bytes specified in operand2.

If ADD is coded for operand3, only the fields that are defined in the edit format are changed on the screen; the characters between the edit format fields remain on the screen for the number of bytes specified by operand2.

If operand2 and operand3 are omitted, and if the edit format does not account for all the positions on the screen within the edit format, the results are unpredictable.

**Note:** The fields must be in the order they appear on the screen.

### PARTITION LOAD AND EXIT INSTRUCTIONS

During IPL, the IPL utility can load an object program into any IBM 5280 partition. The IPL utility prompts the operator to enter the appropriate load parameters via the keyboard. At any time after IPL, a program currently executing within a partition can load another full or partial program into any available partition. The executing program either prompts the operator to enter the load parameters via the keyboard or obtains the load parameters from main storage. The LOAD instruction, issued by the executing program, specifies which method is used to obtain the load parameters.

The EXIT instruction makes a partition available to be loaded with a program.

If an error occurs during a load operation, the IBM 5280 can handle error recovery, or you can write your own error recovery procedures.

At IPL, every partition can be loaded with a LOAD instruction. Then, any partition which has executed an EXIT instruction is available to be loaded by any partition that has a LOAD instruction.

A LOAD instruction can be included at the end of every program. Then, when the program has completed its execution, the current partition can be reloaded with another program. Or, the EXIT instruction can be included at the end of a program, making the partition available to be loaded by another partition. In either case, the load parameters may be obtained from storage or from the keyboard.

An EXIT instruction should not be issued from a foreground partition. I110 is on if the partition is a background partition. The normal way to handle the End of Job key is to test I110; if it is on issue an EXIT instruction; if off, call the Standard Load Processor. See Chapter 6 under *Common Function Routines* for a description of the standard load processor.

### Load a Partition

The format for the LOAD instruction is as follows.

| Mnemonic | Parameters | Operand   |
|----------|------------|-----------|
| LOAD     | ([label]   | [,P A,E]) |

where:

*Parameters* may specify the label of the storage area where the load parameters have been stored. The load parameters are described in this chapter. If the label is omitted, the operator will be prompted to enter the load parameters from the keyboard, using a global load.

*Operand* describes the kind of load to execute, as follows.

| Entry | Description  |
|-------|--|
| P     | Specifies that the load is a partial overlay into the current partition. If P is omitted, a full partition load is executed. If P is coded, the load parameters label must be specified, and must include the relative address where the partial overlay begins. This 2-byte address must be on a 256-byte boundary and must be greater than hex 0100. You may not perform a partial overlay if you are using the standard-load-processor prompt. (The standard load processor is described in Chapter 6 under <i>Common Function Routines</i> .) See <i>Partial Overlay</i> later in this chapter for more information. |
| A     | Specifies that, if a foreground partition is loading a background partition, the foreground partition attempts to attach the background partition after it is loaded. If the attach fails, the load is not affected. If the A is not specified, no attach is attempted.  |
| E     | Indicates that, in case of error during the execution of the LOAD instruction, you have provided error recovery instructions. If the error occurs, control passes to the first instruction following the LOAD. If the load operation is successful, control passes to the second instruction following the LOAD instruction.   |

If E is omitted, system error recovery is used. If E is specified and you load the current partition, E is ignored.

If an executing program loads a program (other than a partial overlay) into a different partition, the newly loaded program begins execution in the newly loaded partition, and the next sequential statement following the LOAD instruction is executed in the current partition.

If an executing program loads a different program into the partition in which it is currently executing, all open data sets within the current partition are closed, all outstanding keyboard operations or enter commands are completed, the new program is loaded, and the newly loaded program begins execution.

A newly loaded program begins execution with the statement specified by the ENTRY parameter of the .START statement. If the ENTRY parameter is not specified, the execution begins with the first executable instruction in the program.

### **Exit a Partition**

The EXIT instruction may be issued in a foreground or background partition. The instruction is used to make a partition available to be loaded.

The format of the EXIT instruction is as follows:

#### **Mnemonic**

#### **EXIT**

If the EXIT instruction is issued in a background partition, a flag is set in the system control area to indicate that the partition is available to be loaded. This flag must be set for the background partition to be loaded by another partition.

If the EXIT instruction is issued in a foreground partition, a flag is set in the partition control area to indicate that the partition is available to be loaded. The flag in the system control area is not set; this allows the IBM 5280 to process key-strokes in the exited partition.

In both foreground and background partitions, the exit operation also detaches the partition if it was attached to a keyboard. It closes all open data sets, and it forces a system unlock in case the partition was locked when the EXIT instruction was issued.

### **The Load Parameters**

When the load parameters are placed into a main storage location (instead of being read from the keyboard), they are placed into a 34 byte area. The load parameters include the partition number, the device ID or address, an optional start address, and the data set name.

**Partition Number:** Two bytes in length, the partition number specifies the partition to load. The first byte contains the partition number (hex F0 to F7). If both bytes contain blanks, the current partition is reloaded.

*Examples:*

X'F640'    Load the partition number 6.  
X'4040'    Reload the current partition.

**Device ID:** Four bytes in length, the device identification specifies the device that contains the data set identified by the Name parameter. The device ID consists of either a 2-character logical ID defined in the resource allocation table or a 4-byte physical device address. If the last 2 bytes contain blanks, the first 2 bytes contain the logical device ID that is stored in the resource allocation table. Otherwise the 4 bytes contain a physical address. The first 2 bytes of the physical address contain the last 2 bytes of the storage address of the IOB pointer, which is in the system control area, for that device. The last byte contains the device subaddress or zeros. Only the printer uses device subaddresses.

*Examples:*

C'D1'        The physical address of D1 is in the resource allocation table.  
C'4400'      This is the physical address of a diskette.

**Start Address:** Two bytes in length, the start address is used only when loading a partial overlay. The start address must be greater than hex 100 and must fall on a 256-byte boundary.

If you are not loading a partial overlay you must include these 2 bytes but they are ignored.

*Examples:*

X'0C00'    256 byte boundaries always have hex 00  
X'1F00'    as the last two hex digits.

**Data Set Name:** A maximum of 26 bytes in length, the data set name consists of an optional volume ID and the mandatory name of the data set to load. No blanks are allowed within the data set name, but the data set name must end with a blank.

*The volume ID* is specified only if volume checking is desired. The volume ID may be up to six alphanumeric characters, must be preceded by an asterisk, and must be followed by a period.

*The name* may be up to 8 alphanumeric characters for an H, I, or basic exchange data set. For an E exchange data set the name may be up to 17 bytes consisting of one or more simple names with each simple name separated by a period. A simple name consists of one to 8 alphanumeric characters with the first always alphabetic. For example, *PROG1.PART1.A* is a data set name consisting of three simple names.

*The final blank* must follow the name.

## Partial Overlay

By using a partial overlay, you can spot load a section of object code or data into a partition without destroying the program already in the partition. You load a partial overlay by specifying a start address in the load parameters. The start address is the storage address where you want the overlay to begin. The original contents of the partition remain unchanged except in the area of the overlay. When the partial overlay load operation is completed, control returns to the instruction following the LOAD instruction. The first 8 bytes of each partial overlay contain information that is added by the assembler.

Specify OPTION=SUB on the .START control statement when you code a partial overlay.

## Error Recovery

Two methods of error recovery can be used if an error occurs during the execution of a LOAD instruction. The first method is to write your own error recovery procedures. The second is to let the IBM 5280 handle the errors.

### *User-Defined Error Recovery*

When a program instruction loads a data set into another partition, the load instruction can indicate that user defined error recovery procedures will handle error recovery. If the load operation is successful, control returns to the *second* instruction following the load instruction. If an error occurs during the load operation, the system places the error code into a system binary register (BR16) and returns control to the *first* instruction following the load instruction. This instruction usually branches to the error recovery procedures.

### *System Error Recovery*

There are four types of error recovery procedures, depending on the type of load taking place when the error occurred. When any type of error occurs, the system sends an error message to the screen and waits for the operator to press the Reset key. After the reset, error recovery is as follows for the different types of loads:

*Global load*, using the standard load prompt from the common function area to prompt for the load parameters to be entered from the keyboard. After reset, the load prompt is redisplayed with the error information that was entered. The operator can then enter the correct information.

*Program instruction reloading the same partition*, with the standard load prompt in the common functions area available.

After reset, the standard load prompt is displayed, which prompts for the load parameters to be loaded from the keyboard.

*Program instruction reloading the same partition*, with no standard load prompt available. There is no way to retry this type of load. The main microprocessor issues an exit instruction and goes to the next partition. The partition that was being loaded is available to be loaded by another partition.

*Program instruction loading another partition*. After reset, the load instruction is not retried. The partition that was being loaded is made available to be loaded by another load instruction following the load instruction.

## TABLE INSTRUCTIONS

The table instructions are divided into global table, table read, table write, and table search instructions. These instructions operate on table set up by `.TABLE` control statements. See the *IBM 5280 Functions Reference Manual* for a description of these tables.

The table instructions do not operate on label tables set up by the `.LABTAB` control statements. The `.LABTAB` label tables are used only to make indexed branches, as with a `CALLTB` or `GOTAB` instruction.

The `.TABLE` control statements build the system table that handles the data tables. Each data table in your program is represented by an entry in this system table. Each system table entry contains the data table address, argument length, number of bytes between arguments, maximum number of arguments, and the number of arguments currently within the table. A source table instruction specifies the table label. The assembler places in the object code instruction the index where the entry for that label is found in the system table. The system table address is stored in the partition control area. The IBM 5280 has access to all the table parameters each time a table instruction is executed.

Table instructions for read, write, and search operations specify the table label, a binary register, and a decimal register. The binary register holds the table index, and the decimal register holds the table argument. More than one decimal register may be needed, depending on the length of the argument. The IBM 5280 uses the smallest number of decimal registers necessary to hold the argument. (The decimal register specification refers to the leftmost decimal register if more than one is used.) The argument for a table operation is the rightmost *n* bytes of the decimal register or registers, where *n* is the value given to the `ARGL` parameter of the `.TABLE` control statement.

If an error occurs during a table operation, an indicator is set on. The indicators used for table errors are as follows:

| Indicator | Meaning If 1   |
|-----------|--|
| I125      | Table search is unsuccessful.  |
| I126      | Invalid index, or table is too small to accept an added entry.               |
| I127      | A table error occurred. When I127 is on, either I125 or I126 is also set on. |

**Note:** The 5280 does not turn these indicators off after the operation has completed.

## Table Read Operations

Table arguments are read into one or more decimal registers. Either the last table entry or the entry at a specified index can be accessed. The format for table read instructions is as follows:

**Result = Mnemonic Table**  
 $R_n = \left\{ \begin{array}{l} \text{TBRD} \\ \text{TBRL} \end{array} \right\} (\text{label}, \text{BR}_n)$

where:

*Result* indicates a decimal register, or the leftmost of the decimal registers, into which the table argument is read.

*Table* indicates the table and the table index. The entry is the table label (LABEL parameter from the .TABLE statement that entered the table into the system table) and a binary register to hold the appropriate table index.

### *TBRD (Read Table Entry at Specified Index)*

**Result = Mnemonic Table**  
 $R_n = \text{TBRD} (\text{label}, \text{BR}_n)$

The argument of the table specified by label, at the index specified by the binary register (BR<sub>n</sub>), is read into the register specified by result (R<sub>n</sub>). If the argument length is greater than 16 bytes, consecutive registers to the right of R<sub>n</sub> are used. Data is right adjusted into the smallest number of decimal registers necessary to hold the argument. The unfilled leftmost bytes of the decimal register are not changed. The argument length is specified by the ARGL parameter of the .TABLE statement.

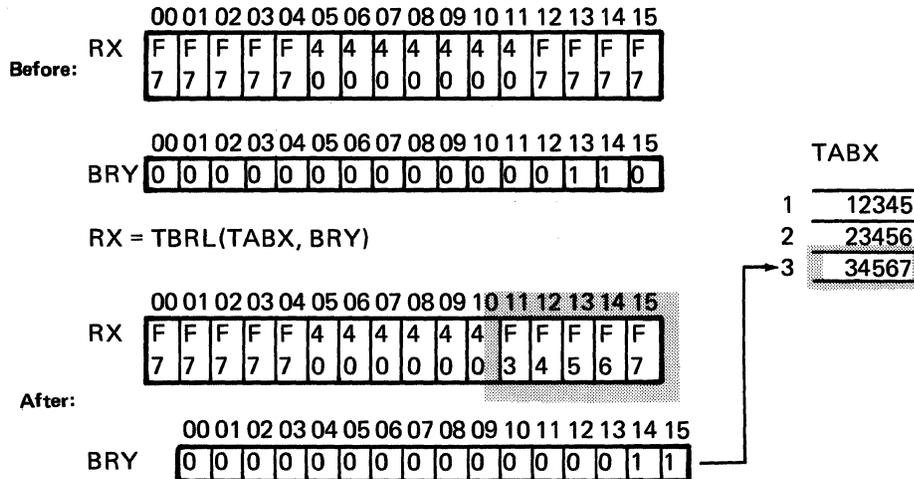
### *TBRL (Read Last Table Entry)*

**Result = Mnemonic Table**  
 $R_n = \text{TBRL} (\text{label}, \text{BR}_n)$

The last argument of the table specified by label is read into the decimal register specified by result (R<sub>n</sub>). If the argument length is greater than 16 bytes, consecutive decimal registers are used. The argument is right adjusted into the smallest number of decimal registers necessary to hold the argument. The unfilled leftmost bytes of the decimal register are not changed. The argument length is determined by the ARGL parameter of the .TABLE statement.

The index of the last table argument is loaded into the binary register specified (BR<sub>n</sub>).

Example:



### Table Write Operations

Table arguments can be inserted, deleted, and written at a specified index. Arguments can be added to variable length tables. The format for these instructions is as follows:

| Mnemonic                       | Table             | Argument |
|--------------------------------|-------------------|----------|
| TBDL                           | (label, BRn)      |          |
| {<br>TBIN<br>TBWE<br>TBWT<br>} | (label, BRn) = Rn |          |

where:

*Table* indicates the table and the table index. The entry is the table label (LABEL parameter from the .TABLE statement that entered the table into the control table), and a binary register (BRn) to hold this table index.

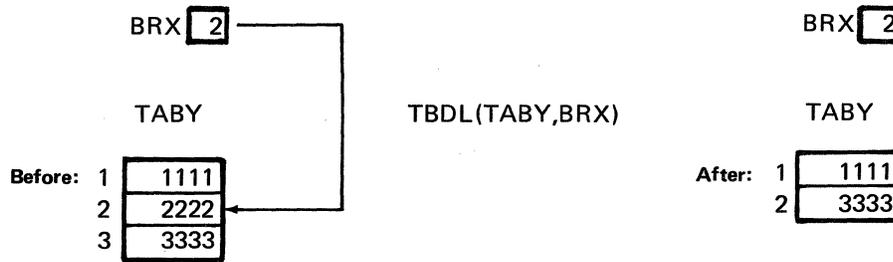
*Argument* indicates a decimal register, or the leftmost of the decimal registers, that hold(s) the table argument to be written. The length of the argument is determined by the ARGL parameter of the TABLE. The argument register (Rn) is not indicated for the delete operation (TBDL).

#### TBDL (Delete Table Entry at Specified Index)

| Mnemonic | Table        |
|----------|--------------|
| TBDL     | (label, BRn) |

The table entry at the index specified by the binary register (BRn) is deleted from the table. Entries below the point where the deletion is made are moved upward to fill in the space made by the deletion, as well as any bypass fields. It is the user's responsibility to update the number of entries in any system table for data tables that describe the bypass fields as active data arguments.

Example:



*TBIN (Insert Table Entry at Specified Index)*

| Mnemonic | Table          | Argument |
|----------|----------------|----------|
| TBIN     | (label, BRn) = | Rn       |

The argument held in the decimal register or registers (Rn) is inserted into the table specified by label, at the index specified by the binary register (BRn). The number bytes inserted is determined by the ARGL parameter of the .TABLE statement.

All table entries currently below the inserted entry are moved downward as well as any bypass fields. This includes the entry that was previously in the table position where the new entry is inserted. The number of entries in the table is increased by 1, and the index to the new table entry is loaded into the binary register specified by BRn. It is the user's responsibility to update the number of entries in any system table for data tables that describe the bypass fields as active data arguments.

If the number of entries in the table equal 0, TBWE must be used to insert the first entry.

*TBWE (Extend Table and Write Entry)*

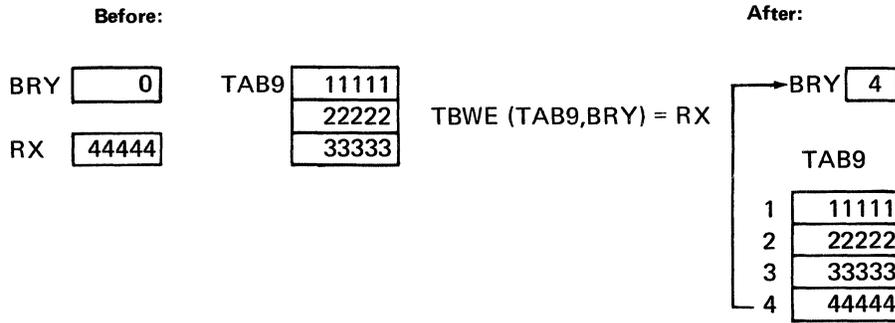
| Mnemonic | Table          | Argument |
|----------|----------------|----------|
| TBWE     | (label, BRn) = | Rn       |

This instruction extends a variable length table and writes the contents of the decimal register (Rn) beyond the current last entry in the table. The length of the argument to be written is determined by the ARGL parameter of the .TABLE statement.

The number of entries in the table is increased by 1, and the index to the new table entry is loaded into the binary register specified by BRn.

**Note:** Maximum table size is determined by the MAXM parameter of the .TABLE statement. Any attempt to extend the table beyond MAXM results in an error. Use caution in estimating maximum sizes of variable length tables.

Example:



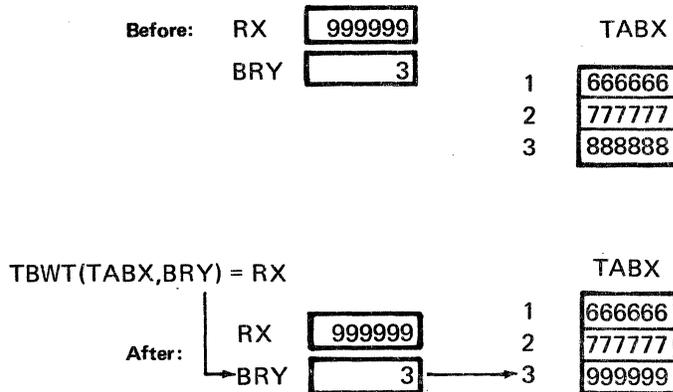
*TBWT (Write Table Entry at Specified Index)*

**Mnemonic    Table    Argument**

TBWT        (label, BRn) = Rn

The contents of the decimal register (Rn) are copied into the table specified by label, at the entry position indicated by the contents of the binary register (BRn). The number of bytes written into the table is determined by the ARGL parameter of the .TABLE statement.

Example:



## Table Search Operations

Tables can be searched to find an entry equal to, lower than, or higher than a specified argument. A table can also be searched using a binary search. The following indicators are turned on to indicate the result of the search.

| Indicator | Result                    |                                   |
|-----------|---------------------------|-----------------------------------|
| I101      | ON if higher entry found. | } (Only one of these will be on.) |
| I102      | ON if lower entry found.  |                                   |
| I103      | ON if equal entry found.  |                                   |
| I125      | ON if entry not found.    |                                   |

I125 should be the first indicator tested after a table search. The indicators should be reset before another table search is issued.

The number of bytes used in the search are the rightmost n bytes of the decimal register or registers specified, where n is the value given to the ARGL parameter of the .TABLE control statement.

The format of the search instructions is as follows:

| Result | = | Mnemonic  | Table           |
|--------|---|---|-----------------|
| BRn    | = | TBBS  | (label,Rn)      |
| BRn    | = | $\left\{ \begin{array}{l} \text{TBFH} \\ \text{TBFL} \\ \text{TBFX} \end{array} \right\}$ | (label,Rn [,N]) |

where:

*Result* indicates a binary register (BRn) into which the index of the table entry selected by the search operation is placed upon completion of the operation.

*Table* specifies the search parameters.

| Parameter | Description   |
|-----------|---|
| label     | The label of the table to search.   |
| Rn        | A decimal register that contains the argument to be searched for. The argument data is right adjusted within the register.  |
| N         | May be specified on any search except the TBBS. If N is coded, the search begins with the next entry after the current index contained in the binary register (BRn). If N is not coded, the search begins with the first entry. |

### *TBBS (Binary Search for Equal Table Entry)*

| Result = | Mnemonic | Table      |
|----------|----------|------------|
| BRn =    | TBBS     | (label,Rn) |

The labeled table is searched, by binary search, to find a table entry equal to the data contained in the decimal register (Rn). The entries of the table being searched must be in ascending order.

If an entry is found in the table that equals the search argument in the decimal register, the index of that entry is placed into the specified binary register (BRn). Indicator I103 is set on.

If no equal entry is found, I125 and I127 are set on. BRn remains unchanged.

*Example:*



**TBFH (Search for Equal or Higher Entry)**

**Result Mnemonic Table**

BRn = TBFH (label,Rn [,N])

The labeled table is searched for an entry equal to or higher than the data contained in the decimal register (Rn). The entries in the table to be searched must be in ascending order.

If N is not coded, the search begins with the first table entry (index 1). If N is coded in the instruction, the search begins with the next sequential index after the index currently held in the binary register (BRn).

The search ends when the first equal or higher entry is found or when all table entries between the beginning of the search and the end of the table have been searched. If an equal or higher entry is found in the table, the index of that entry is placed into the binary register (BRn). If no equal or higher entry is found, I125 and I127 are set on. BRn remains unchanged.

*Example:*



The first entry searched was at index 3.

**TBFL (Search for Lower Entry)**

**Result = Mnemonic Table**

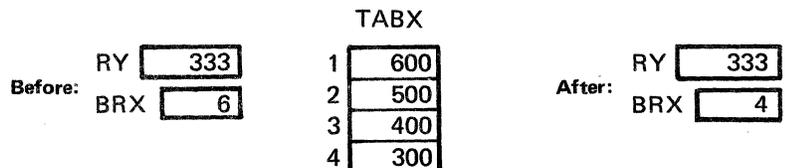
BRn = TBFL (label, Rn [,N])

The labeled table is searched for an entry that is lower than the data contained in the decimal register (Rn). The entries in the table to be searched must be in descending order.

If N is coded, the search begins at the next sequential index after the index contained in the binary register. If N is not coded, the search begins at index 1.

The search ends when the first lower entry is found, or when all table entries between the beginning of the search and the end of the table have been searched. If a lower entry is found, the index of that entry is placed into the binary register (BRn). If no lower entry is found, I125 and I127 are set on. BRn remains unchanged.

*Example:*



The instruction BRX = TBFL (TABX,RY) would result in BRX containing 4. The search would begin with entry 1 because N is not specified.

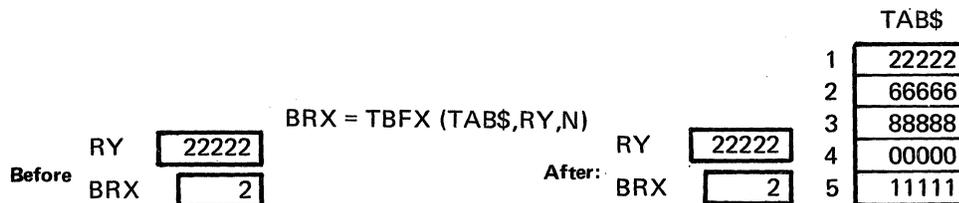
*TBFX (Search for Equal Entry)*

| Result = | Mnemonic | Table            |
|----------|----------|------------------|
| BRn =    | TBFX     | (label, Rn [,N]) |

The labeled table is searched for an entry equal to the data contained in the decimal register (Rn). The entries in the table to be searched do not have to be ordered, because all entries are searched for an equal entry.

Unless N is coded, the search begins at index 1. If N is coded, the search begins at the next sequential index after the index held in the binary register (BRn).

If any equal entry is found, the index for that entry is placed into the binary register (BRn). If no equal entry is found, I125 and I127 are set on. BRn remains unchanged.



The search begins at index 3. After index 4 and 5 are searched and no equal entry is found, I125 and I127 are set on and the operation terminates.

The following code searches an input record for a specific character:

```
.  
.  
.  
.DATASET DSN=2 LBUF=INPUT TYPE=SR ...  
.DC LABEL=INPUT LEN=256;  
.TABLE LABEL=INTAB DCLBL=INPUT ARGL=1 MAXM=256;  
    READ(2); Get input record.  
    15, RY=X'40'; Search argument is a blank.  
    BR40(4) SR 32; Zero BR40 and BR41.  
TLOOP: BR40=TBFX(INTAB, RY,N); Search.  
    IFIR 125 IS ON GOTO ENDLOOP;  
    BR41 += 1; Increment recorder.  
    GOTO TLOOP; Go to resume search.  
ENDLOOP: SOFF(127) ; Turn off indicator.  
.  
.  
.
```

### Global Tables

Tables in the common function area can be accessed by any partition. The label of the table must also be included in the .XTRN control statement. The partition can then access the table by specifying the label in a TABLE instruction.

Global tables can be read, searched and updated by any partition.

Whenever a partition accesses a global table in an instruction, the table is locked to any other partition until the instruction is completed. However, the table can be locked indefinitely by the TLCK instruction. The program that issued the TLCK instruction has exclusive use of the table and must issue a TUNLCK instruction before any other partition can access the table. The format of the instructions to lock and unlock global tables is as follows.

| Mnemonic | Table   |
|----------|---------|
| TLCK     |         |
| TUNLCK   | (label) |

where:

*Table* specifies the label of the global table to lock or unlock.

### *TLCK (Lock Shared Table)*

| Mnemonic | Table   |
|----------|---------|
| TLCK     | (label) |

The table specified by label is locked for exclusive use by this partition. The table will remain assigned exclusively to this partition until a table unlock (TUNLCK) instruction is encountered.

If the table is already locked by another partition, the program will remain at this instruction until the other partition issues a TUNLCK instruction.

This instruction is used only with global tables located in the common function area.

**Note:** There is no automatic deadlock detection or recovery. If two partitions are waiting for tables held by each other, the system will lock up and require IPL.

#### *TUNLCK (Unlock Shared Table)*

| Mnemonic | Table |
|----------|-------|
|----------|-------|

|        |         |
|--------|---------|
| TUNLCK | (label) |
|--------|---------|

This instruction frees a shared table that has been locked by the TLCK instruction. Whenever a table has been locked, this instruction must be used in order to make the table available to other partitions.

This instruction is used only with global tables located in the common function area.

### MISCELLANEOUS INSTRUCTIONS

#### Compare Logical Character Strings

The contents of two character strings are compared. An indicator is set on to signify an equal to, greater than, or less than relationship. The character strings can be from 1 to 256 bytes in length. The format of the CLC instruction is as follows.

| Mnemonic | String1 | String2 | Length |
|----------|---------|---------|--------|
| CLC      | (BRa ,  | BRb ,   | 1-256) |

where:

*String1* indicates a binary register that contains the address of the first byte of string1.

*String2* indicates a binary register that contains the address of the first byte of string2.

*Length* specifies the number of bytes to compare. Length may be from 1 to 256 bytes.

The two strings are compared byte for byte until the first miscompare, according to the standard EBCDIC collating sequence. One of the following indicators is set to indicate the result of the compare. The other 2 indicators are reset.

| Indicator | Condition                       |
|-----------|---------------------------------|
| I103      | String1 is equal to string2     |
| I102      | String1 is less than string2    |
| I101      | String1 is greater than string2 |

The contents of the 2 byte strings remain unchanged by the operation.

*Example:*

|         |   |         |         |   |   |   |   |   |   |
|---------|---|---------|---------|---|---|---|---|---|---|
| STRNG1  | <table border="1"><tr><td>X'0970'</td></tr></table> | X'0970' | X'0970' | <table border="1"><tr><td>A</td><td>B</td><td>C</td><td>D</td><td>E</td></tr></table> | A | B | C | D | E |
| X'0970' |   |         |         |   |   |   |   |   |   |
| A       | B   | C       | D       | E   |   |   |   |   |   |
| STRNG2  | <table border="1"><tr><td>X'0975'</td></tr></table> | X'0975' | X'0975' | <table border="1"><tr><td>A</td><td>B</td><td>D</td><td>D</td><td>D</td></tr></table> | A | B | D | D | D |
| X'0975' |   |         |         |   |   |   |   |   |   |
| A       | B   | D       | D       | D   |   |   |   |   |   |

The instruction CLC (STRNG1, STRNG2, 2) would set indicator I103.

The instruction CLC (STRNG1, STRNG2, 4) would set indicator I102.

### Generate a Self-Check Digit

The instruction to generate a self-check digit is used when you create a new file of numbers that you want to self-check each time an operator enters them. For more information about self-check numbers, see *Self Check* in Chapter 2.

| Mnemonic | Operand |
|----------|---------|
| GSCK     | (Rn)    |

where:

*Operand* indicates the decimal register (Rn) or double register (Rn(32)) that contains the self-check field.

During the execution of the GSCK operation, the IBM 5280 uses the self-check algorithm (specified in the .SELFCHK control statement) to generate a unique self-check number from the foundation contained in the decimal register or double register. The IBM 5280 places the one or two generated digits into the self-check digit position in the register to complete the self-check field.

**Note:** When using the Standard Modulus 11 Self-Check algorithm, if a self-check digit of X'10' is generated, system indicator number 117 will be set to indicate that this particular foundation is invalid for this self-check algorithm.

### Modification for Indirect Instruction Execution

Indirect instruction execution is accomplished by instruction modification during program execution. An operand of the object code, which was generated by a source instruction, is modified. The instruction is then executed with the operand modification, although the original object code for the operand, which is stored in main storage, is not changed.

Each source instruction generates 4 bytes of object code. This code is stored in main storage. The first byte (byte 0) always contains the operation code, and the next 3 bytes (bytes 1-3) contain the operands. See Appendix A for the operation codes generated from each assembler mnemonic. The contents of the remaining 3 bytes of each object code instruction is described in the *Function Reference Manual*.

The format of the INXEQ instruction is as follows:

| Mnemonic | Modifier   | Statement | Byte   |
|----------|------------|-----------|--------|
| INXEQ    | (BRn[(4)], | label,    | [0-3]) |

where:

*Modifier* indicates a binary register that contains the statement modifier.

If a binary double register is specified, all 4 bytes of the double register are used to modify the 4 bytes of the instruction. If a 2-byte binary register is used, only one byte of the instruction is modified, with the data in the rightmost byte of the binary register.

*Statement* specifies the label of the statement to modify.

*Byte* specifies which byte of the object code to modify. If a binary double register is used for the modifier, this operand is omitted and all 4 bytes are modified. If a 2-byte binary register is used for the modifier and this operand is omitted, it defaults to byte 1 (the operation code is byte 0).

When the INXEQ statement is executed, the byte or bytes of the object code are logically ORed with the contents of the binary register or binary double register. The modified statement is then executed. After the modified statement is executed, control returns to the next sequential statement following the INXEQ statement, unless the modified statement is a branch instruction. If the modified statement is a branch and the branch is taken, control branches to label specified by the branch instruction. If the modified instruction is a skip and the skip is taken, the instruction skipped is the one immediately following the INXEQ instruction. This instruction should not be used to modify a short branch instruction.

The original contents of the object code operand byte remain unchanged.

Any object code instruction may be modified with this instruction. A list of the most commonly modified instructions is as follows:

#### Instructions that May be Modified, and Valid Modification

| Instruction        | Modifiable Operand | Modifiable Object Code |                                |
|--------------------|--------------------|------------------------|--------------------------------|
|                    |                    | Byte                   | Object Code Value              |
| MVC                | length             | 1                      | length-1                       |
| MVCR               | length             | 1                      | length-1                       |
| MVCV               | length             | 1                      | length-1                       |
| CLC                | length             | 1                      | length-1                       |
| TRANS              | length             | 1                      | length-1                       |
| TRT                | length             | 1                      | length-1                       |
| label = constant   | constant           | 1                      | constant                       |
| Rn = constant      | constant           | 1                      | constant                       |
| IFC                | constant           | 1                      | constant                       |
| IFHI/IFLO          | mask               | 2                      | hex AND mask                   |
|                    |                    | 3                      | hex exclusive-OR mask          |
| READ               | format, dsn        | 2, 1                   | format number, data set number |
| WRT                | format, dsn        | 2, 1                   | format number, data set number |
| Table instructions | index              | 2                      | index                          |

### Duplicate a Byte in Storage

The contents of a byte in a main storage partition can be duplicated one to 256 times.

The format of the DUP instruction is as follows:

| Mnemonic | Address      | Number  |
|----------|--------------|---------|
| DUP      | ([disp,] BRn | ,1-256) |

where:

*Address* specifies a base displacement address of the storage byte to duplicate. The contents of the base register (BRn) are added to the displacement (0-255) to find the address of the byte.

*Number* specifies the number of times to duplicate the contents of the storage byte specified by result.

The contents of the byte at the base displacement address are duplicated for the number of times indicated by number. The duplicated data is placed into storage in the bytes immediately following the byte that is being duplicated.

### Search Resource Allocation Table

| Mnemonic | Data Set | Operand |
|----------|----------|---------|
| SRAT     | (1-15    | ,BRn)   |

The SRAT operation loads the physical address for the specified data set into the operand binary register (BRn). When the SRAT operation is executed, the IBM 5280 takes the logical device identifier from hex 60 and 61 of the IOB for the data set specified in the instruction. The IBM 5280 searches the resource allocation table for a matching logical device identifier entry. If a match is found, the IBM 5280 loads the binary register specified in the instruction with the corresponding physical device address, which is stored with the logical device identified in the resource allocation table.

If no logical device identifier is present in the IOB, or if no match is found in the resource allocation table, or if there is no resource allocation table, I118 is set on and the IBM 5280 finds the physical device address in the partition logical I/O table entry that corresponds to the specified data set.

## Set Bits with Mask

The bits of 1 byte in storage can be set on or off by using a mask and logical operations.

The format for these instructions is as follows:

| Mnemonic            | Result        | Mask      |
|---------------------|---------------|-----------|
| { SETON<br>SETOFF } | ([disp], BRn, | constant) |

where:

*Result* specifies a base displacement address of the byte to be masked. The contents of the base register (BRn) are added to the displacement (0-255) to find the address of the byte.

*Mask* is a 1-byte constant, as indicated under *Constant Specifications* at the beginning of this chapter.

When the SETON operation is executed, the storage byte at the address specified by result is logically ORed with the mask, and the result replaces the original contents of the storage byte.

When the SETOFF operation is executed, the storage byte at the address specified by result is logically ANDed with the complement of the mask, and the result replaces the original contents of the storage byte.

## Set Indicators

Up to three indicators may be turned on or off with a single instruction.

The format of the instructions to set the indicators is as follows:

| Mnemonic        | Operand                              |
|-----------------|--------------------------------------|
| { SON<br>SOFF } | { (Ia)<br>(Ia, Ib)<br>(Ia ,Ib ,Ic) } |

where:

*Operand* specifies the indicator or indicators to set on or off. Indicators are referred to by number (I0-I254) or by label. One, two, or three indicator numbers may be specified, or the operands may be omitted.

When the SON operation is executed, the indicator or indicators specified by operand are set on (B'1'). When the SOFF operation is executed, the indicator or indicators are reset off (B'0').

## Exclusive-OR Write, Skip on AND Mask

The following instruction specifies that two logical operations be performed upon the same data. These operations can change the original contents of a specified byte; they can also test the original contents to determine if a skip should take place. This instruction is useful to test and set/reset lock bits with a single instruction.

The format of the instruction is as follows:

| Mnemonic | Mask1     | Test | Mask2      |
|----------|-----------|------|------------|
| RXORW    | (constant | ,BRn | ,constant) |

where:

*Mask1* specifies a 1-byte constant to be exclusively-ORed with the contents of the test byte. See *Constant Specifications* near the beginning of this chapter for the forms that the constant may have.

*Test* specifies a binary register that contains the address of the byte to test.

*Mask2* specifies a 1-byte constant to be ANDed with the contents of the test byte. The constant may be expressed in any of the forms indicated for mask1.

The 1-byte constant specified by mask1 is exclusively-ORed with the contents of the test byte at the address indicated by the binary register. The original contents of the test byte are ANDed with mask2. If the result of the AND operation is 0, a skip results. The result of the exclusive-OR operation replaces the original contents of the test byte. If the result of the AND operation is not 0, the next sequential instruction is executed, and the original contents of the test byte remain unchanged.

## System Lock and Unlock

An executing program can issue an instruction to cause the IBM 5280 to be locked into the current partition. The IBM 5280 remains in the partition until the executing program issues an instruction to release the IBM 5280. The format of the instructions to lock or unlock current partition is as follows:

**Mnemonic**    **Operand**

SYSLCK

SYSUNL    [(\*)]

where:

*Operand* may be specified for SYSUNL. If the asterisk is included, the IBM 5280 unlocks the partition but executes within the partition for the normal time period before it exits. If the asterisk is omitted, the IBM 5280 exits the partition as soon as it executes the SYSUNL instruction.

When the SYSLCK instruction is executed, the IBM 5280 is locked into the current partition. Whenever the time limit for the IBM 5280 to work within the partition is exhausted, the IBM 5280 resets the timer and continues to work within that partition until a SYSUNL instruction is encountered. After the SYSLCK is executed, no other partition is processed by the IBM 5280 until the SYSUNL instruction is encountered. You should exercise great care when using this instruction. Use this instruction only when you want a series of instructions to be executed without interruption (such as a time-out interruption). Do not execute I/O instructions or access global tables while the IBM 5280 is locked into your partition.

When the SYSUNL instruction executes, the IBM 5280 leaves the current partition for a time-out, unless the asterisk is included in the instruction. If the asterisk is included, the IBM 5280 executes instructions within the partition for the normal time slice before it exits the partition.

## Translation

The IBM 5280 processes all data in EBCDIC notation. However, other notation can be read into the I/O buffer and translated to EBCDIC. Or data can be moved from main storage to the I/O buffer, translated from EBCDIC, and written in another notation to a diskette. This translation requires a translate table. You can code your own translate table with DC control statements. If your IBM 5280 common function area contains the optional ASCII Translate Table, you may use that table from any partition, the label is CF ASCII.

If you code your own translate table, you must declare the table to begin on a 256-byte boundary. The length must be 512 bytes, divided into two separate 256-byte tables. The first 256 bytes are used for the input translate table, and the next 256 bytes are used for the output translate table.

In the input translate table, the EBCDIC value of each input character is stored at a displacement that corresponds to the value of the translated code. For example, the character P is represented by D7 in EBCDIC and by hex 50 in ASCII. The value of hex D7 is stored at a displacement of hex 50 in the input translate table. When the character P is read into the input buffer, the ASCII value of hex 50 is translated to the EBCDIC value stored at displacement hex 50.

The output translate table stores the code value of the translated code at the displacement into the output translate table that corresponds to the EBCDIC value. To translate the character P to ASCII, the ASCII value of hex 50 is stored in the output translate table at displacement hex D7.

The EBCDIC values between hex 00 and 3F are used for various keyboard/display and printer control characters. For the printer, SCS control characters are very difficult to translate. For example, using the printer SCS control characters 34 C8 06 (skip 6 columns), the hex 34 and 06 may be easily translated to themselves by placing hex 34 and 06 at locations 34 and 06 respectively in the translate table. However, the hex C8 (EBCDIC H) would have a hex 48 (ASCII H) in the translate table causing hex 34 C8 06 to be translated to hex 34 48 06, which will cause a printer error. Thus, it is not recommended to translate SCS data sets.

You can initiate translation in two different ways in your source program. If you want translation performed on every record read from or written to a data set, you can use the TRANS parameter of the .DATASET control statement to specify the label of the appropriate translate table. The translation will occur automatically, immediately after a physical record is read or immediately before a physical record is written.

If you do not use the TRANS parameter, you can issue a translate instruction to translate one record. The instruction translates the data that is pointed to by the Operand1 register.

The format for the translate instructions is as follows:

| Mnemonic | Operand1 | Operand2 | Length | Reverse |
|----------|----------|----------|--------|---------|
| TRANS    | (BRa,    | BRb,     | 1-256) |         |
| TRT      | (BRa,    | BRb,     | 1-256  | [,R]    |

where:

*Operand1* indicates a binary register that contains the address of the leftmost byte of the data stream to be translated.

*Operand2* indicates a binary register that contains the address of the leftmost byte of the translate table.

*Length* is a constant between 1-256 that indicates how many bytes are to be translated.

*Reverse* is specified only for TRT for reverse testing.

When the TRANS operation is executed, the number of bytes specified by length is translated, character by character. The translation begins at the address stored in the operand1 binary register (BRa). The translate table used for the translation begins at the address specified by the operand2 binary register (BRb). Characters in the data stream are replaced by their translated value.

When the TRT operation is executed, the test begins at the address stored in the operand1 binary register (BRa) and continues for the number of bytes specified for length. The translate table used for the test begins at the address stored in the operand2 binary register (BRb).

Beginning with the data byte specified, the EBCDIC representation of the character at each byte is used as an offset into the translate table. If the entry at that offset into the translate table is a zero, the test continues to the next data byte. If the entry at that offset into the translate table is not zero, binary register sixteen (BR16) is loaded with the relative address of the data byte, and the rightmost byte of binary register seventeen (BR17) is loaded with the translate table entry for the data stream character.

Data characters are tested one by one until the first nonzero translation occurs or until the entire data stream has been tested. If each character in the data stream translates to a zero, BR16 and BR17 contain zero.

The original characters in the data stream are not changed.

The following indicators may be turned on according to the specified conditions.

| <b>Indicator</b> | <b>Condition</b>   |
|------------------|--|
| I101             | Nonzero byte not found   |
| I102             | Nonzero byte found   |
| I103             | Nonzero byte found in last position of data. (If R is coded, the nonzero byte is found in first position). |

### THE IBM 5280 ASSEMBLER

The IBM 5280 assembler is stored as a data set and its overlays on the assembler diskette. It must be loaded into a main storage partition in order to process your source program. The partition may be a foreground partition or a background partition that has a keyboard attached. The partition must be a minimum of 9 K bytes in size. You load the assembler by entering the assembler data set name in response to the load prompt. The assembler data set name is SYSASM.

When the assembler begins executing, it displays a series of prompts. The prompts request the labels of up to six data sets. These data sets include the:

1. *Source data set*, which contains your source program.
2. *Extern data set*, which is an optional data set that defines common function labels. You must include this data set if your program uses common function routines. The extern data set is stored on the assembler diskette.
3. *Work data set 1*, which is used by the assembler. This data set is mandatory. It may be allocated during the assembly. If allocated during the assembly, it will be deleted by the assembler upon normal termination of the assembly.
4. *Work data set 2*, which is required if you request a cross reference listing. This data set may be allocated during the assembly. If allocated during the assembly, it will be deleted by the assembler upon normal termination of the assembly.
5. *Object data set*, for the object code output. This data set may be allocated during the assembly.
6. *Print data set*, which is required if you write your listing to a diskette. This data set may be allocated during assembly.

These data sets are not required to be on the same diskette, but they can be. If you preallocate any of the last four data sets, they should be specified as I exchange, with 256-byte records and with the delete character omitted.

During execution, the assembler makes multiple passes over the source code in order to generate the object code output. For a 9 K partition, about 285 symbols may be processed. Any additional available storage is automatically added to the symbol table.

If you requested a cross-reference listing, the symbols from the symbol table are copied into work data set 2. The symbols are later sorted into alphabetic order for output.

After the symbols are resolved, subsequent passes over work data set 1 develop the object code. The final pass, over work data set 1 and the source code together, produces the object code and assembly listing. The object code is written to the specified object data set. The listing may be written to a diskette as the print data set, or may be written to a printer. Printed output is formatted with 128 print positions per line and requires a minimum print width of 12.8 inches. Work data set 1 and work data set 2 are used only during the assembly process. If allocated by the assembler, they are deleted when the assembler has completed its execution. Otherwise, they remain on the diskette to be used for the next assembly.

### **Loading the Assembler into a Partition**

The assembler may be loaded into a foreground or a background partition. If it is loaded into a foreground partition, the keyboard/display assigned to the partition is not available to another partition until the assembler is finished executing. All errors detected by the assembler are included in the assembly listing. When the assembler encounters certain errors, the assembly stops and the error code is displayed on the status line. You can press the Reset key to continue the assembly, or press the End of Job command function key to terminate the assembly.

If the assembler is loaded into a background partition, a keyboard/display must be attached until the prompts have been displayed and the necessary input has been accepted. The keyboard/display is then detached and available to another partition while the assembler runs in the background partition. Error messages are included on the program listing.

The assembler diskette must remain in the diskette drive while the assembler is executing.

### **The Assembler Prompts**

When the assembler begins execution, a series of prompts is displayed. Many of the prompts display a default response. If you wish to accept the default, simply press the Enter key. The cursor is positioned at the beginning of each response field; to change the default response, key over the displayed default. When a separate response is required on two or more lines, the cursor will be positioned at the first position of the first response field. Key in your response and press the Field Exit key. The cursor will move to the first position of the next response field.

The status line is always active while prompts are being displayed. If an error occurs, the status line blinks and the appropriate error message is displayed. Press the Reset key to stop the blinking, and key over the error response.

If an operator who is not familiar with assembler programming is to assemble your program, be sure to provide information concerning all necessary data set labels, device identification, and options you want for your assembly.

The assembler prompts are as follows.

*Prompt 1*

```
0 0001      D 01 F1
5280 assembler
Options are:
  1. List to printer          3. No listing
  2. List to diskette
Select option: _ Press ENTER
```

10-01

If you select option 1, the assembly listing will be printed with 128 print positions per line. The printer must have a minimum print width of 12.8 inches. If the paper width is not sufficient for the printout, part of the listing is lost and damage to the printer may occur.

If you select option 2, the assembly listing will be written to a diskette data set. A later prompt will request the name of this data set, referred to as the print data set.

If you select either option 1 or option 2, Prompt 2 and Prompt 3 are displayed. If you select option 3, Prompt 4 is the next prompt displayed.

*Prompt 2*

```
0 0001      D 01 F1
Enter assembler print options
Cross reference (1=Yes,2=No):   Included lines (1=Yes,2=No):
Full data print (1=Yes,2=No):   Errors only (1=Yes,2=No):
Literal spacing (1=Yes,2=No):   Page size (Lines/Page):
                                Press ENTER
```

10-02

The assembler print options default to 66 lines per page (the maximum is 255) and a cross reference with a full data print. If you enter no (2) for the full data print option, the listing will print no more than the first 8 bytes of object code generated from each source line.

If you choose yes (1) for included lines, any program code inserted into the program with an `.INCLUDE` control statement is printed in the listing.

If you choose yes (1) for errors only, only the error messages and the lines that caused the error messages are printed on the listing. If you specify errors only and your program assembles error-free, only the program name and title are printed on the listing.

The spacing option effects the way your .SPACE and .EJECT control statements are processed. If you choose yes (1) for this option, the printer will space and go to a new page exactly as the control statements specify. If you choose no (2) for this option, control statement specifications are adjusted to the following conditions.

If .SPACE control statement causes the printer to go to the next page, printing begins on the first line of the new page rather than at the line indicated by the control statement. For example, if the .SPACE statement specifies six lines to be skipped, and only three lines remain on the current page, printing begins on line 1 of the next page rather than on line 4.

If an .EJECT control statement is encountered when the printer is already positioned to print the top line of a page, the .EJECT statement is ignored.

*Prompt 3*

```
0 0001      D 02 40
Enter date
Day:
Month:
Year:
                Press ENTER
                10-03
```

The date prompt is presented if Prompt 2 was used and if the date is not available from the system control area.

*Prompt 4*

```
0 0001      A 25 E2
5280 assembler
Enter the following information for - SOURCE data set
Data set name:
Device address:
                Press ENTER
                10-04
```

The source data set name is the label you assigned to the diskette data set in which you recorded your source statements. Your source statements are entered as 80-byte logical records, although only the first 72 bytes are used. Source records may be stored on a diskette of any exchange type. Any block size may be used, providing the block size is valid for the exchange type and sector size used.

The device address identifies the diskette drive that is to read in the source data set. Enter the four-character physical address.

Before you press the Enter key, be sure the diskette with the specified data set is in the diskette drive at the specified device address.

*Prompt 5*

```
0 0006      A 66 40
Diskette is volume protected.
Device DDDD. Data set SSSS.
Enter owner identifier to access volume.
-----
                                Press ENTER                                05-01
```

If the protect flag is set in the diskette label, Prompt 5 is displayed. Enter the 14-character owner ID, as described in the *System Concepts Manual*. If you terminate the assembly at this point, by leaving the owner ID blank and pressing the Enter key, the assembler data set is closed and the partition becomes available to be loaded again.

After the data set name and device address for the source data set have been obtained, the assembler will attempt to open the data set. If an error occurs during open, the status line blinks and the appropriate error code is displayed in the status line. Press the Reset key to stop the blinking, then reenter the parameters. If the data set is successfully opened, Prompt 6 is displayed.

*Prompt 6*

```
0 0001      A 25 E2
5280 assembler
Enter the following information for - EXTRN data set
Data set name:
Device address:      (leave blank if none)
                                Press ENTER                                10-05
```

The extern data set is a list of common function routine labels. If your source program makes any calls to the common function area, you must enter the label of the extern data set. The extern data set is stored on the assembler diskette.

After you enter the data set name and device address, the assembler attempts to open the file. Errors that occur during open are handled as for the source data set open.

*Prompt 7*

```
0 0001      A 25 E2
5280 assembler
Enter the following information for - OBJECT data set
Data set name:
Device address:
                                Press ENTER                                10-04
```

The object data set is the data set the assembler will use to write the assembled object code. It is also used as another work data set during the assembly. After you enter the data set name and device address, the assembler attempts to open the data set. If the open is successful, Prompt 10 is displayed.

If the data set cannot be found at the specified device address, the assembler blinks the status line and displays an open error message. When you press Reset, the blinking stops and Prompt 8 is displayed.

*Prompt 8*

```
0 0026      D 01 F1
Unable to find data set
Do you want it allocated?
Options are
1. Yes      2. No
Select option: _ Press ENTER
```

10-06

If you select 2, Prompt 7 is displayed again.

If you select 1, Prompt 9 is displayed with the data set name and device address you entered for Prompt 7.

*Prompt 9*

```
0 0030      D 04 40
Enter data set allocation parameters
Data set name:
Device address:
Number of records:      (0 for MAX)
Press ENTER
```

10-07

If you wish to change the data set name or device address, use the ← (Field Backspace) key to move the cursor to the field you want to change. Enter the number of 256-byte records that you estimate will be written. This typically allows 4 records per 1024 bytes of the partition the program will run in.

After you enter the parameters, the assembler will attempt to allocate the data set as indicated. If the allocation fails, the status line blinks and displays an error message, as for an open error. Press the Reset key and reenter the parameters.

*Prompt 10*

```
0 0001      A 25 E2
5280 assembler
Enter the following information for - work data set # 1
Data set name:
Device address:
                                     Press ENTER
                                     10-04
```

The work data set 1 is used by the assembler only during the assembly process. After you enter the parameters, the assembler attempts to open the data set.

If the data set is not found at the specified address, the assembler blinks the status line and displays an open error message. When you press the Reset key, the blinking stops and Prompt 8 is displayed, as for the object data set. If you select 1, Prompt 9 is displayed again. Allocate work data set 1 as for the object data set. Work data set 1 should be about one-half the size of the source data set. If you assemble more than one source program at the same time, be sure to specify a different work data set for each assembly.

After work data set 1 has been successfully opened, the subsequent prompts depend upon your responses to the first two prompts.

If you requested an assembly listing in Prompt 1 and a cross reference in Prompt 2, the next prompt requests the name of the work data set 2. The prompt is as for Prompt 10, except the first line contains:

**ENTER THE FOLLOWING INFORMATION FOR – WORK DATA SET 2**

Work data set 2 is opened or allocated as for work data set 1. Work data set 2 is used only during the assembly process. For work data set 1, one 16-byte record is required for each symbol definition or reference. Estimate one record per two source lines.

If you requested an assembly listing to a diskette in Prompt 1, the next prompt requests the print data set name. The prompt is as for Prompt 10, except the first line contains:

**ENTER THE FOLLOWING INFORMATION FOR – PRINT FILE**

The print data set is used by the assembler to write an assembly listing to a diskette. The print data set is opened or allocated as for work data set 1.

If you requested a printed assembly listing, Prompt 11 is displayed.

*Prompt 11*

```
0 0001      N 04 F8
5280 assembler

Enter printer device address:
                Press ENTER

10-08
```

The printer address must be the 4-character physical address.

The assembler listing is printed with 128 print positions per line. This requires a minimum print width of 12.8 inches.

When all necessary data sets have been opened, the assembler is ready to process the source data set. If the assembler is executing in a background partition, the keyboard is now detached. The status line will indicate the partition number of the currently attached partition.

After the assembler has developed the object code, it writes the data sets and listings according to your responses to the prompts. The object data set may be loaded into any main storage partition for execution.

Normal error recovery is provided for the printer. However, if the printer is interrupted while it is printing your listing and it cannot continue, you can recover without reassembling your program. Press the Cmd key, then the End of Job command function key; the load prompt is displayed by the standard-load routine. Load data set SYSASM8 from the assembler diskette. This data set then prints out your assembly listing. The diskette(s) must be in the same diskette drive(s) as when the assembly was stopped. The same partition must be used as was used for the assembly.

If the printer is found to be in use, Prompt 12 appears.

*Prompt 12*

```
1 0001      D 01 F1
5280 assembler

Printer is unavailable. Options are:
  1. Re-enter printer device address
  2. Continue with assembly
Select option:  1 Press ENTER
```

If you select 1, Prompt 11 is displayed. You can enter a new device address or retest the availability of the current device address.

If you select 2, the assembly continues. If the printer is still unavailable when needed (pass 7 or 8), a message code will be displayed on the status line. Options at this point will be RESET or EOJ.

## THE ASSEMBLY LISTING

The IBM 5280 assembler produces a conventional parallel column source-object listing. If the assembler detected an error in a source statement, it flags the statement with asterisks and includes an error message. The edit format and screen control format messages are printed first, then the other messages are printed together, usually in sequence by line number. See Appendix E for a list of the IBM 5280 assembler error messages. Following is an example of a printed listing and the kinds of information it contains.

### A Printed Assembly Listing

```

5280 ASSEMBLER 01.00          SAMPLE2 - ORDER ENTRY SAMPLE PROGRAM;

ADDR      OBJECT CODE      LINE      SOURCE CODE
1 0080      E2C1D4D7D3C5F240      4 .START PNAM='SAMPLE2' ENTRY=START MCHK=CFPGMCHK;          00560000
0080      0010000000000000      *NBCRT CRBA=IOF          ETAB=ERRKB;          00570000
00A6      * 58 .DC LABEL=SCNCL          EVL=2 DISP=X'A6';          SCAN CODE & CONVERTED EBCDIC          00580000
00BE      * 59 .DC LABEL=MODESW          LEVL=2 DISP=X'BE';          KB/CRT MODE SWITCH          00590000
0140      * 60 .DCLIND LABEL=EODSW,FRNTSW,QUANSW;          00600000
0001      * 61 .EQUATE NUMB=(1,TRANS),(2,PRNT);          .DATASET NUMBERS          00610000
3 * 62 * THE FOLLOWING EQUATES ARE OPTIONAL, BUT WILL PUT ENTRIES IN THE          00620000
* 63 * CROSS REFERENCE LISTING FOR SYSTEM REGISTERS AND INDICATORS.          00630000
0114      * 64 .EQUATE REG=(BR10,BR10),(BR18,BR18),(BR19,BR19),(BR22,BR22);          00640000
0000      * 65 .EQUATE IND=(I115,I115),(I118,I118),(I125,I125),(I158,I158);          00650000
0115      * 66 .XTRN LABEL=CFPGMCHK,CFGIDERR,CFPERATT,CFLOAD01,CFERCD5M,          00660000
* 67          CFATFBGD,CFDEVCHK;          00670000
0140      0800      * 68 .DC LABEL=CFWKPTR          TYPE=BIN INIT=ADDR(CFSAVE);          C.F. WORK AREA PTR          00680000
0142      * 69 .DC LABEL=CFPARM1          TYPE=BIN;          C.F. PARAMETER REG          00690000
0144      * 70 .DC LABEL=CFPARM2          TYPE=BIN;          C.F. PARAMETER REG          00700000
0146      * 71 .DCLBR LABEL=WKBR1;          BINARY WORK REGISTERS          00710000
0148      0435      * 72 .DC LABEL=RWCOL          TYPE=BIN INIT=X'0435';          ROW & COL FOR CRTMM          00720000
014A      08C2      * 73 .DC LABEL=BUF@          TYPE=BIN INIT=ADDR(IOBUF);          I/O BUFFER PTR          00730000
014C      08E4      * 74 .DC LABEL=TXT@          TYPE=BIN INIT=ADDR(SHPCD);          PTR FOR CRTMM          00740000
014E      0150      * 75 .DC LABEL=PRCNTL@          TYPE=BIN INIT=ADDR(PRCNTL);          PTR TO PRINT CONTROL          00750000
0150      8000      * 76 .DC LABEL=PRCNTL          TYPE=BIN INIT=X'8000';          PRINTER SPACE CONTROL          00760000
0152      0006      * 77 .DC LABEL=K6          TYPE=BIN INIT=6;          00770000
0154      0022      * 78 .DC LABEL=K22          TYPE=BIN INIT=X'22';          00780000
0156      0029      * 79 .DC LABEL=K29          TYPE=BIN INIT=X'29';          00790000
0158      3701      * 80 .DC LABEL=K3701          TYPE=BIN INIT=X'3701';          CODE FOR EOD          00800000
0160      * 81 .DCLDR LABEL=WKDR1,RQUAN,RPRICE;          DECIMAL WORK REGISTERS          00810000
0190      F0F0F0F0F0F0F0      * 82 .DC LABEL=LIMIT          TYPE=DEC INIT=1000;          QUANTITY LIMIT CHECK          00820000
01A0      F9F1F1F1FF      * 83 .DC LABEL=MSG9111          INIT=X'F9F1F1F1FF';          9111 - SELF CHECK INCORRECT          00830000
01A5      F9F9F9F8FF      * 84 .DC LABEL=MSG9998          INIT=X'F9F9F9F8FF';          9998 - UNEXPECTED KB EXT STATUS          00840000

```

- 1** Hexadecimal address of the object code.
- 2** Object code generated from the source line.
- 3** Comment lines included in the source program.
- 4** Assembler adds sequential statement number, which is not included in the source code.
- 5** Source line.
- 6** The assembler includes whatever is coded in columns 73-80 of the source line.

## The Cross-Reference Listing

The cross-reference listing lists your program symbols in alphabetic order. It lists the references for each symbol, combining as many of the individual reference records as will fit into a single line, but starting a new line for each new symbol. The references indicate the relative record number of the source record that used the symbol. Following is an example of a cross-reference listing.

```

5280 ASSEMBLER 01.00          SAMPLE2 - ORDER ENTRY SAMPLE PROGRAM;

DEFINED SYMBOL      COMP@ LENG VALUE TYPE      REFERENCES
 1      00002      AMT      0000 000D 08E6  L...A      5      00120 00373 00401
00360    ATTN      0000 0004 0DEB  INST      00346
00417    BKRG      0000 0004 0EAB  INST      00321
00211    BLANKS    0014 0019 0BE5  PRMT      00177 00178 00179
00258    BLNK      0000 0004 0CB0  INST      00248 00358
00064    BR10      0014 0002 0114  BIN      00274 00292
00064    BR18      0024 0002 0124  BIN      00414
00064    BR19      0026 0002 0126  BIN      00370
00064    BR22      002C 0002 012C  BIN      00307
00073    BUF@      004A 0002 014A  BIN      00258 00259 00286 00287
00373    BUFAD     0000 0008 0E0C  DATA     00120
00094    BUFF1     0000 0400 0300  DATA     00087
00095    BUFF2     2      0000 0100 0700  DATA     00091
00066    CFATFBGD * 0000 0000 012D  INST      00232 00245 00260 00265 00288 00364
00066    CFDEVCHK * 0000 0000 0139  INST      00236 00251
00066    CFERCDSM * 0000 0000 0125  INST      00332 00379
00066    CFGIOERR * 0000 0000 0119  INST      00309
00066    CFLOAD01 * 0000 0000 0141  INST      00341 00365

```

- 1 The number of the line where the symbol is defined.
- 2 Common function labels are marked with asterisks.
- 3 Compressed address or index into the system table where the address is stored. (See *Functions Reference Manual* for a description of compressed addresses for registers).
- 4 Data type.
- 5 The number of each line that refers to the symbol.

## Error Messages

Error messages are printed at the beginning of your assembly listing, in the following format.

```
5280 ASSEMBLER 01.00          GALTEST6 - ASSEMBLY ERRORS
```

| ERROR         | LINE       | ## | DESCRIPTION                       |
|---------------|------------|----|-----------------------------------|
| A             | 3002-00041 |    | TYPE is invalid (too large, etc)  |
| AS            | 0050-00031 |    | F T2 is an undefined symbol       |
| ASM0050-00032 |            |    | MDUP is an undefined symbol       |
| ASM0077-00034 |            |    | MS and MD must be identical       |
| ASM3002-00035 |            |    | MD is invalid (too large, etc)    |
| ASM0077-00035 |            |    | MS and MD must be identical       |
| ASM0077-00036 |            |    | MS and MD must be identical       |
| ASM3002-00036 |            |    | MS is invalid (too large, etc)    |
| ASM3002-00037 |            |    | MD is invalid (too large, etc)    |
| ASM3002-00037 |            |    | MS is invalid (too large, etc)    |
| ASM0050-00020 |            |    | MDUP is an undefined symbol       |
| ASM1001-00027 | 34         |    | Invalid delimiter at column ##    |
| ASM0036-00057 |            |    | Option or modifier not recognized |
| ASM0036-00058 |            |    | Option or modifier not recognized |
| ASM2003-00058 | 02         |    | Operand ## is wrong type          |
| ASM2005-00066 | 02         |    | Operand ## must not be omitted    |
| ASM2003-00071 | 01         |    | Operand ## is wrong type          |
| ASM2003-00072 | 01         |    | Operand ## is wrong type          |
| ASM0050-00078 |            |    | IS is an undefined symbol         |
| ASM0050-00078 |            |    | SKIP is an undefined symbol       |

**1** Error code.

**2** Line in the assembler listing where the error occurred.

**3** Description of the error.

**4** When a number appears in this column, it corresponds to the ## in the description. For example, the 34 indicates an invalid delimiter at column 34 in line 00027. The first 02 indicates that the second operand in line 00058 is the wrong data type.



## SYSTEM INDICATORS WITHIN A PARTITION

The first 100 indicators within a partition may be used as you wish. The other indicators, however, are used by the system during program execution. Indicators effected by an instruction should be set to 0 by the programmer before use of the instruction. The indicator assignments are as follows.

|          |                              |   |
|----------|------------------------------|---|
| I100     |                              | Reserved  |
| I101     | Table search                 | Result is higher  |
|          | TRT                          | Byte not found  |
|          | CLC                          | String 1 is greater than string 2   |
| I102     | Table search                 | Result is lower   |
|          | TRT                          | Byte is found   |
|          | CLC                          | String 1 is less than string 2  |
| I103     | Table search                 | Result is equal   |
|          | TRT                          | Byte found in last position (EOF)   |
|          | CLC                          | String 1 is equal to string 2   |
| I108     | External status              | Restricted external status processing. (See <i>Restricted External Status Indicator</i> later in this chapter.) |
| I109     | Program check                | Program check error   |
| I110     |                              | Background partition  |
| I111-114 |                              | Reserved  |
| I115     | SCS conversion data set      | Last line (LSTLN) overflow  |
| I116     |                              | Reserved  |
| I117     | Self-check operation         | Self-check error  |
| I118     | SRAT operation               | Resource allocation table search error  |
| I119     | HEXBIN operation             | Attempt to convert invalid EBCDIC to hex  |
| I120     | Divide operation             | Divide error (denominator is zero)  |
| I121     | Edit format                  | Invalid conversion request in format  |
| I122     | Arithmetic operation         | Decimal to binary conversion error  |
| I123     | Multiply operation           | Multiply overflow   |
| I124     | Decimal arithmetic operation | Decimal arithmetic overflow   |
| I125     | Table search                 | Table entry not found   |
| I126     | Table write                  | Attempt to access table beyond its limit  |
| I127     | Table operation              | Table operation error   |
| I128-153 |                              | Not defined   |
| I154-159 |                              | Reserved for common function routines   |
| I160-191 |                              | Modified field indicators. See <i>Field Modification Indicators</i> in Chapter 2.                               |
| I192-254 |                              | Used with DE/RPG  |

## SYSTEM REGISTERS WITHIN A PARTITION

Several binary registers are used by the system during program execution. These registers are listed below, with the conditions or instructions that affect each register.

| Register | Condition                | Register Contents   |
|----------|--------------------------|---|
| BR16     | LOAD                     | Relative record number for relative record reads during a program read. Also contains error code after a loader error |
|          | TRT                      | Address of the last position that translated to a nonzero character   |
| BR17     | TRT                      | Function byte   |
| BR18     | Subroutine               | Address of next available entry position in the partition subroutine stack  |
| BR19     | Keyboard External Status | Current field starting address within the main storage I/O buffer   |
| BR20     | Keyboard External Status | Current field starting address within the screen refresh buffer in keyboard/display control storage                   |
| BR21     | Keyboard External Status | Field length of current field, minus 1  |
| BR22     | External Status          | Relative address of the last data set IOB to report external status. Not used for keyboard/display status             |
| BR23     | External Status          | External status condition code, to be used as the index into the external status subroutine                           |
| BR24     |                          | Reserved for system use   |
| BR25     | LOAD                     | Physical device address of the device doing the load  |
| BR26-31  |                          | Reserved  |

In addition to these system binary registers, BR32-34 are used as parameter registers by common function routines. If your program uses common function routines, you should avoid using these registers for other purposes.

## PROGRAM CHECK ERRORS

The IBM 5280 detects and reports program check errors. When a program check error occurs, the IBM 5280 calls the routine specified by the MCHK parameter of your .START control statement. If you are using the common function program check error handler routine, it places the program check error code on the status line, as described in the program check error handler description.

The conditions that are detected by the IBM 5280 as program check errors are:

| Code | Error   |
|------|---|
| 0201 | Addressing outside of partition.  |
| 0202 | Invalid operation code  |
| 0203 | Attempt to pass control to a common function routine through an inboard GOTO instruction. |
| 0204 | ENTR issued from a background partition that is not attached to a keyboard.               |
| 0205 | Subroutine stack extended beyond partition.   |
| 0206 | Invalid keyboard/display external status.   |
| 0207 | Attempt to access an undefined data set.  |
| 0208 | Attempt to access an undefined self-check control area.                                   |
| 0209 | DETACH failed during an exit or load operation.   |
| 020A | ATTACH failed during a load operation.  |
| 9924 | Common function routine not available in the common area.                                 |

## KEYSTROKE COUNTERS

The IBM 5280 maintains two keystroke counters while an ENTR is being processed. A data entry keystroke counter is incremented when the operator presses a key in enter, update, or verify mode. A verify correction keystroke counter is incremented when the operator presses a key to correct a verify mismatch character in verify mode.

### Data Entry Keystroke Counter

When one of the following keys is pressed in enter, update, or verify mode, the data entry keystroke counter is incremented *unless*:

- An error is outstanding when the key is pressed.
- The keystroke causes an error other than a data required, blank check, or mandatory enter error.
- The keystroke is a Field Exit, Field- (Field Exit Minus), or Skip key that is pressed in the first position of a field and causes a mandatory enter error.

- The keystroke is for a function key that is being handled by your own sub-routine, as specified in the keyboard bit map.

Data key  
 Hex key sequence  
 Field Advance key  
 Field Backspace key  
 Character Advance key  
 Character Backspace key  
 Record Advance key  
 Record Backspace key  
 Field Exit or Field+ key  
 Field- (Field Exit Minus) key  
 Skip key  
 Duplicate key  
 Scan code passed to the keyboard by keyboard operation 0A  
 EBCDIC passed to the keyboard by keyboard operation 0B.

### Verify Correction Keystroke Counter

The verify correction keystroke counter is incremented when the operator presses a key in verify mode that changes the original data in the record. If the mode is field correct, the counter is incremented only once for the entire field.

### COMMON FUNCTION ROUTINES

The common function routines described in this chapter are listed in alphabetic order by title, and include:

| Label    | Title                                 |
|----------|---------------------------------------|
| CFASCII  | ASCII processor                       |
| CFATFBGD | Attach partition routine              |
| CFCOMDSP | Communications status display routine |
| CFDETFGD | Allow detach routine                  |
| CFDEVCHK | Check/move device address             |
| CFDUMPTR | Dump/trace processor                  |
| CFEOJRTN | End of job routine                    |
| CFERCDSM | Error code with message display       |
| CFERCDSP | Error code display routine            |
| CFGIOERR | General I/O error handler             |
| CFKEYRT  | Keystroke router routine              |
| CFLOAD01 | Standard load processor               |
| CFMSGDSP | Message display routine               |
| CFPERATT | Operator detach routine               |
| CFPGMCHK | Program check error handler           |
| CFSECVOL | Secure volume processor               |

The common functions can be accessed by a program executing in any main storage partition. When you write a source program that uses one or more of these common functions, you must:

1. Specify the common function labels with an `.XTRN` control statement.
2. Enter the data set name `SYSACF` in response to the assembler prompt that requests the Extern data set.
3. Specify the common function label in the program instruction or control statement.
4. Declare a 128-byte area of storage and store the address of the area in `BR32`. This common function save area should not begin at a location of less than `X '01C0'`.
5. Avoid declaring IOBs or buffers at locations of less than `X '01C0'`.

If your program is displaying the extra line when a common function routine is called, the routine replaces it with the status line. When the routine completes execution, it replaces the status line with the extra line. (See *Nondisplay of the Status Line* in Chapter 2.)

The common function labels and the function descriptions are included in this chapter. Several routines require input in addition to the address in `BR32`. Before you use one of the common functions in your program, you must be familiar with any input or output pertaining to the function.

#### *Registers Used by the Common Functions*

During the execution of certain common functions, the IBM 5280 uses binary registers `BR32-95`, which are located within the partition that accessed the common function. You must load the address of a 128-byte register save area into `BR32`. When a routine begins execution, the contents of the binary registers are copied into the location pointed to by `BR32`.

While the common function routine is executing, it uses the 128 bytes as temporary work registers and buffers.

When the common function routine completes execution, the original contents of the binary registers are restored except for `BR33` and `BR34`. The original contents of `BR33` and `BR34` are often destroyed.

When a common function routine invokes a call to another common function routine, the partition subroutine stack is used to store the return address. For this reason, the stack must contain more entries than the maximum number of nested subroutine calls in your program. The default stack size is usually sufficient.

### *Common Function Error Routines*

Several subroutines accept as input information about an error and display it on the status line. These subroutines are CFERCDSP, CFERCDSM, CFGIOERR, and CFPGMCHK.

When one of these subroutines is called, the following operations are performed:

1. If the program is being executed in a partition that is not attached to a keyboard, the attach routine (CFATFBGD) is called to attach the partition.
2. The error message is built using information input to the subroutine (refer to descriptions of the individual subroutines for details on the input data) and moved to the status line using the software-request-for-error-mode operation (the KERRST instruction). If the calling program is currently displaying the extra line, its contents will be saved while the status line containing the message is displayed.
3. The system help text table is searched to find text associated with the error code being displayed. If any is found, it is moved to the status line starting in column 41. Notice that pressing the Help key is not required in order for help text to be displayed.
4. The keyboard is opened to accept response keystrokes from the operator (KACCPT operation). If an acceptable key is pressed, the keyboard is clicked. (Unacceptable keystrokes are not clicked.)
  - a. If the Reset key is pressed, the error mode will be cleared and control will return to the calling program, skipping the next sequential instruction.
  - b. If the EOJ key is pressed, the error mode will be cleared and control will return to the calling program at the next sequential instruction.
  - c. If the Attention key is pressed, it will be handled like the Reset key, except that the operator detach routine (CFPERATT) will be called before control is returned.
  - d. If the DUP/SKIP or Auto Enter keys are pressed, those functions will be disabled if they had been enabled. The error message remains on the status line until Reset, EOJ, or Attention is pressed.

**Note:** The Shift key does not function in error mode. Holding down the Shift key while pressing other keys does not affect their operation.

5. Before control is returned to the calling partition, the error message and help text (if any was displayed) are cleared from the status line. If the extra line was displayed, it is redisplayed with its contents intact.

**Note:** If a program is using the extra line, the FUNC=5 parameter should be coded on the .KBCRT statement so that status line messages will be correctly displayed.

6. If the calling program turned on Indicator 157 before calling the CFA error routine, a copy of the error message as it appeared on the status line will be returned in the extra line.

### *ASCII Processor (CFASCII)*

The ASCII processor makes the ASCII translate table in the common area available to the partition.

#### *Input*

- BR32—Address of your register save area.
- BR34—Data set number.

#### *Output*

- ASCII table index number placed into data set IOB.
- Bit 2, byte hex 4D of data set IOB set to 1.
- Registers restored except BR33.

When the routine is executed, the following operations are performed.

1. The routine checks to make sure the ASCII table is included in the common area. If there is no ASCII table in the common area, an error occurs and return is made to the next sequential instruction.
2. If there is an ASCII table in the common area, the system table index for the ASCII table is placed into displacement hex 0E of the IOB specified by BR34, and bit 2 of byte 4D is set to 1. This specifies that both the HDR1 labels and data are to be translated. If the labels are not to be translated, the program in the partition must clear this bit to 0.

Return is made, and the next sequential instruction is skipped.

### *Attach Partition Routine (CFATFBGD)*

The attach partition routine attaches the calling partition to its associated keyboard. When the keyboard is attached upon completion of this routine, the calling partition can accept input from the operator via the keyboard.

#### *Input*

- BR32—Address of your register save area.
- I156—If your program is displaying the extra line instead of the status line, turn on I156 before calling this routine so the extra line will be returned to the screen when the return is made.

### *Output*

- I158 is turned on.
- Registers are restored.
- The partition is attached.

When this routine is executed, the following operations are performed.

1. Indicator I158 is turned on.
2. If the calling partition is already attached, the status line or the extra line is refreshed (depending on the setting of I156), and return is made to the next sequential instruction.
3. If the calling partition is not attached, the CFA will wait for the partition that is currently attached to release the keyboard/CRT before refreshing the status/extra line and returning.
4. If the unattached calling partition is a background and the currently attached partition is the foreground, the keyboard alarm is sounded and the edge indicator ( ■ ) is displayed on line 6 of the screen. The edge indicator is displayed on the right side of the screen except for dual displays; for a dual display, the edge indicator is displayed on the left side of one of the screens. The operator must respond to the buzzer and edge indicator by pressing the Attn key. After the Attn key is pressed, the attach is performed and return is made.

### *Allow Detach Routine (CFDETFGD)*

This routine is called by a program that is executing in an attached partition. The program calls this routine when it reaches a point where it can execute for a period of time without needing the keyboard.

### *Input*

- I158

### *Output*

- I158 is turned off

When this routine is executed, the following operations are performed.

1. I158 is turned off to indicate that the program in the partition can execute without using the keyboard. This indicator is checked by the standard load processor (CFLOAD01) and the operator detach routine (CFPERATT).
2. Return is to the next sequential instruction.

### *Communications Status Display Routine (CFCOMDSP)*

The communications status routine displays diagnostic information about the communications line, if a CAM is loaded in the system. Refer to documentation on the communications utilities for further detail.

#### *Input*

- BR32—Address of your register save area.

#### *Output*

- Communications status displayed on screen.

This routine may be accessed directly from assembler programs, however the suggested method for supporting the status display is through CFKEYRT. If this routine is called directly, the following operations are performed.

1. A check is made to see if a communications utility which supports the status display is loaded in the system. If so, a call is made to the utility.
2. Control returns to the partition, and the next sequential instruction is skipped.

**Note:** It is the responsibility of the assembler program to assure that the partition is attached to a keyboard (by calling CFATFBGD) before attempting to display communications status.

### *Check/Move Device Address (CFDEVCHK)*

This routine checks EBCDIC input and processes it for a logical device ID or for a physical device address.

#### *Input*

- BR32—Address of your register save area.
- BR33—Address of an input field.
- BR34—Data set number.

#### *Output*

- Device ID or physical address moved to IOB or logical I/O table, or both.
- Registers restored except BR33.

When the routine is executed, the following operations are performed.

1. The address of the EBCDIC input is taken from BR33.
2. If the EBCDIC input is two characters in length, it is treated as a logical device ID. It is moved to displacement hex 60-61 of the IOB specified by BR34, and the resource allocation table is searched to find the corresponding physical device address. If not found, an error occurs.
3. If the EBCDIC input is four characters in length, it is treated as a physical device address. It is tested to determine if it is a numeric value that can be converted to a valid physical address. If it is not, an error occurs. If it is, the physical address is *not* checked to assure that a device is installed. The zones are removed from the EBCDIC bytes to convert them to four hex digits. The first two hex digits are moved to the logical I/O table in the partition, and the second two digits are moved to the data set IOB specified by BR34, to displacement hex 16.
4. If an error occurs, control returns to the next sequential instruction. Otherwise, the next sequential instruction is skipped.

#### *Dump/Trace Processor (CFDUMPTR)*

This routine must be called before any dump or trace operation is performed. The Dump/trace routine opens data set 15. You must have previously defined data set 15 with a .DATASET control statement. The data set must use a 256-byte physical buffer and a nonoverlapping 128-byte logical buffer. If you want the data set written to a diskette, the data set labeled DUMP0000 must have been previously allocated. This data set cannot be allocated on a secure diskette. The data set attributes must include type attributes that work on printer and diskette, such as sequential write or shared write (TYPE = SW, SHRW).

### *Input*

- BR32--Address of your register save area.
- DUMP0000--preallocated if you want to send output to diskette.
- Data set 15 defined.

### *Output*

- Data set 15 IOB opened.
- Device address for DUMP0000 moved to data set 15 IOB and partition logical I/O table.
- Registers restored.

When the routine is executed, the following operations are executed.

1. The routine checks to make sure the partition has a data set 15 defined. If there is no data set 15, error code 9914 is displayed. If the operator presses the Reset key to reset the error, return is made and the next sequential instruction is skipped. If the operator presses the End of Job command function key to reset the error, return is made to the next sequential instruction.
2. If data set 15 is defined, the following prompt is displayed on the screen:

```
0 0001      N 04 40  
Dump/trace file open  
Enter device address:  
  
Press ENTER
```

05-02

3. The device address is accepted from the operator.
4. An OPEN is issued to the specified device. If an open error occurs, the general I/O error handler is called. If the operator presses the Reset key to reset the error, the prompt is redisplayed. If the operator presses the End of Job command function key to reset the error, return is made to the next sequential instruction.
6. If the operator presses the Cncl (Cancel) key in response to the dump/trace prompt, return is made and the next sequential instruction is skipped.

If the operator presses the End of Job command function key in response to the dump/trace prompt, return is made to the next sequential instruction.

If the operator presses the Sys Req (System Request) key in response to the prompt, the standard load processor is called. That routine returns to the dump/trace processor, and the dump/trace prompt is redisplayed.

If the operator presses the Attn (Attention) key in response to the prompt, the allow partition detach routine is called. That routine returns to the dump/trace processor and the dump/trace prompt is redisplayed.

If the operator enters a device address and presses the Enter key in response to the dump/trace prompt, and the open is successful, return is made and the next sequential instruction is skipped.

At the conclusion of the dump, the data set is closed but the EOD is not updated. If you want to access the data set after the dump, you must use the close failure program to update the EOD.

### *End of Job Routine (CFEOJRTN)*

This routine provides a standard means of ending a program. Since control does not return, this routine should be called as the last operation performed.

#### *Input*

- None

#### *Output*

- If the calling program is in a foreground partition, CFLOAD01 is called to display the standard load prompt.

When this routine is called the following operations are performed:

1. Keyboard activity is canceled and the status line is displayed.
2. All open data sets are closed.
3. If communications is active, it is terminated.
4. If the partition is attached, the screen is cleared.
5. If the partition is locked (SYSLCK instruction), it is unlocked (SYSUNL instruction).
6. If the partition is a foreground, CFLOAD01 is called.
7. If the partition is a background, an exit operation is performed.

### *Error Code Display Routine (CFERC DSP)*

The error code display routine displays an error code on the status line.

#### *Input*

- BR32—Address of your register save area.
- BR33—Address of the error code in the following format:

CCCC — Where C is an EBCDIC digit.

*Output*

- Error code displayed on status line in the following format:

|               |   |        |         |
|---------------|---|--------|---------|
| <b>Column</b> | 0 | 7      | 41      |
|               | P | -CCCC- | TTTT... |

P = Partition number.  
C = Error code.  
T = Help text, if any is defined for this error code.

*Error Code with Message Display (CFERCDSM)*

This routine displays an error code and message on the status line. You must declare and store the message in a storage location, in the following format:

CCCCLMMMMMMMMMMMMMM  
Where CCCC is the 4-byte error code, in EBCDIC.  
L is the length-1 of the message, in binary.  
M is the message, in EBCDIC.

If the length of the message is greater than 15, no message is displayed.

*Input*

- BR32—Address of your register save area.
- BR33—Address of error code and message.

*Output*

- Error code and message are displayed on the status line in the following format:

|               |   |        |          |                  |        |
|---------------|---|--------|----------|------------------|--------|
| <b>Column</b> | 0 | 7      | 16       | 25               | 41     |
|               | P | -CCCC- | NNNNNNNN | MMMMMMMMMMMMMMMM | TTT... |

P = Partition number.  
C = Error code.  
N = Program name.  
M = Message text.  
T = Help text, if any defined for this error code.

### General I/O Error Handler (CFGIOERR)

This routine displays information on the status line when an I/O device encounters an error.

#### Input

- BR32—Address of your register save area.
- BR22—Address of the IOB for the data set that had the error. This address is placed into BR22 by the IBM 5280 when an external status condition occurs. You must call this routine in the external status subroutine before external status is enabled or the address may be lost.

#### Output

- Error information is displayed on the status line in the following format:

```
Column  0  3      7      13  16      25      41
        P  DDDD -CCCC- LL  NNNNNNNN  SSS ...  TTT ...
```

P = Partition number.  
D = Physical device address.  
C - Error code.  
L = Logical ID, if available.  
N = Program name.  
S = Data set name, if available.  
T = Help text, if any defined for this error code.

The routine does not clear the error code in the data set IOB. This is to prevent clearing a possible second external status condition.

### Keystroke Router Routine (CFKEYRT)

This subroutine interprets the type of keyboard external status that has occurred in your program. In order to use it, you should call it from the part of your program that gets control when keyboard external status occurs. For example:

```
.KBCRT          CTBA= BUFFER          ELAB=KBEXT;
.
.
.
KBEXT: CALL CFKEYRT;          PROCESS EXTERNAL STATUS
      CALL CFEOJRTN;          'EOJ' KEY WAS PRESSED (CONTROL
      RETURN;                  DOES NOT RETURN)
                                ELSE - CONTINUE PROGRAM
```

#### Input

- BR32—Address of your register save area.

## Output

- None

When this routine is called, the following operations are performed:

1. The routine interprets the following keystrokes and routes them as specified to other CFA routines.

| Key                                     | Routing                                |
|---|--|
| • Attn (Attention)                      | CFPERATT                               |
| • Sys Req (System Request)              | CFLOAD01                               |
| • Dump Trace Open command function      | CFDUMPTR                               |
| • Communications Status (CMD-Shifted S) | CFCOMDSP                               |
| • EOJ (End of Job) command function     | Return to next sequential instruction. |

2. If the external status type is Record Advance (Enter key), Record Backspace (RK BSP key), Forward pass over RG field, or Backward pass over RG field, control is returned skipping the next sequential instruction.
3. If the external status type is for the magnetic stripe reader, the reader is reset (RSTMG operation).
4. External status types other than Command key (1), Shifted Command key (2), Function key (3), RG (4 and 5), Enter (6), Record Backspace (7), and Mag Stripe (11) cause CFERCDSM to be called to display a 9998 message code. After the Reset or EOJ key is pressed in response to the message, control is returned to the next sequential instruction.
5. Before returning to the calling partition, external status is enabled. Also, if the current ENTR operation has not completed, a RESUME operation is performed. The use of RÊTEXT or RESMXT instructions is not necessary after calling CFKEYRT.
6. Notice that control always returns by skipping the next sequential instruction, unless EOJ was pressed or unexpected external status occurred (9998 error).

**Note:** If your program needs to handle some type of external status differently than is done by CFKEYRT, the external status type of interest can be explicitly checked for and processed, calling CFKEYRT to handle everything else.

## Message Display Routine (CFMSGDSP)

This routine puts a message of up to 56 characters into the extra line of the screen. You must declare and store the message in a storage location the following format:

LMMM . . .

Where L is the length-1 of the message, in binary.  
M is the message, in EBCDIC.

If you want this message to be displayed and your program is displaying the status line instead of the extra line, turn on indicator 156 before calling this routine or issue a DISPEX instruction immediately after calling this routine.

#### *Input*

- BR32—Address of your register save area.
- BR33—Address of the message.
- I156—If on, a DISPEX instruction will be issued. Otherwise a DISPST instruction is issued.

#### *Output*

- The message is moved to the extra line.

When this routine is called, the following operations are performed.

1. If the program is being executed in a partition that is not attached to a keyboard, the attach routine (CFATFBGD) is called to attach the partition.
2. The address of the message is taken from BR33.
3. The extra line of the screen contains the following:

| <b>Position</b> | <b>Content</b>   |
|-----------------|--|
| 1               | Partition number   |
| 16-23           | Program name   |
| 25-80           | Message (If the binary length specifies a number greater than 55, only the partition number and program name are moved to the extra line.) |

4. Control returns from this routine to the partition, to the next sequential instruction.

#### *Operator Detach Routine (CFPERATT)*

The operator detach routine detaches the currently attached partition from the keyboard. This routine is called via external status when the operator presses the Attn key. You must include this call in your external status subroutine if you want to allow the operator to interrupt a program executing in an attached partition. When this routine has completed executing (unless I158 is on), the keyboard is available to be attached to another partition.

#### *Input*

- I158

#### *Output*

- The partition is detached.
- The screen is cleared.

When this routine is executed, the following operations are performed.

1. The screen is cleared.
2. The KDETCH and SYSUNL instructions are issued in the attached partition to detach the partition and cancel any remaining execution time. See Chapter 4 for more information about these instructions.
3. If I158 is on, the attach partition routine (CFATFBGD) is called to reattach the calling partition prior to returning.  
  
If I158 is off, the partition remains detached when the return is made.
4. Return is made to the next sequential instruction.

#### *Program Check Error Handler (CFPGMCHK)*

The program check error handler displays detected program check errors. If you use this common function routine, you must specify the label for the MCHK parameter of the .START control statement (MCHK=CFPGMCHK). The IBM 5280 calls the routine whenever a program check error occurs. See *Program Check Errors* in this chapter for a list of the errors and corresponding error codes.

#### *Input*

- BR32—Address of your register save area.

#### *Output*

- Error message is displayed.
- A copy of BR32-95 is stored in the register save area.

When the CFPGMCHK routine is called, it performs the following:

1. If the program is being executed in a partition that is not attached to the keyboard, the attach routine (CFATFBGD) is called to attach the partition.
2. If BR32 is not zero, it copies the contents of BR32-95 into the storage area pointed to by BR32. If BR32 is zero, the contents of BR32-95 are placed into storage starting at the location of BR96. This destroys the original contents of 128 bytes starting at BR96.

3. The status line displays the following information:

| Position | Content   |
|----------|---|
| 1        | Partition number                                      |
| 7-12     | —CCCC—<br>Where CCCC is the program check error code. |
| 16-23    | The program name                                      |
| 24       | * if program check in CFA                             |
| 25-28    | The address of the current instruction                |

Positions 1-40 of the status line blink.

4. I109 is reset.
5. The address of the standard load processor (CFLOAD01) is placed into the partition control area. This ensures a return to the standard load processor from all external status in case the program check resulted from program code that destroyed the exit routines.
6. This routine accepts a response from the operator, who must press the Reset key or End of Job command function key. When the End of Job command function key or the Reset key is pressed, CFEOJRTN is called to terminate the program.

While the error code information is being displayed on the status line before reset, the operator may use the dump console function to dump the data set. See the *Functions Reference Manual* for a description of the Dump console function.

### *Secure Volume Processor (CFSECVOL)*

This routine should be called in response to a 3211 error. The secure volume processor displays a prompt requesting the operator to enter owner ID information when a secure diskette is being accessed.

#### *Input*

- BR32—Address of your register save area.
- BR34—Address of a 14-byte storage area.
- BR22—Address of the IOB of the data set that had the error. This address is placed into BR22 when an external status condition occurs. You must call this routine in the external status subroutine before external status is enabled. If you call this routine when external status has not occurred, you must place the address into BR22.

#### *Output*

- Registers are restored, except BR33.

When this routine is executed the following operations are performed.

1. The address of the data set IOB is taken from BR22.
2. The following prompt is displayed on the screen:

```
0 0006      A 66 40
Diskette is volume protected.
Device DDDD. Data set SSSSSSSS.
Enter owner identifier to access volume.
-----
                                Press ENTER
```

95-01

When DDDD is the logical device ID, or physical device address.  
SSSSSSSS is the last 8 bytes of the data set name if the name is available.

3. If the operator enters the owner identification and presses the Enter key, the routine accepts the input from the keyboard and stores it into the storage location pointed to by BR34. Control returns to the partition, and the next sequential command function instruction is skipped. If the operator presses the End of Job command function key instead of the Enter key, control returns to the next sequential instruction.

If the operator presses the Attn key instead of the Enter key, the allow attach routine (CFPERATT) is called. If the operator presses the System Request key instead of the Enter key, the standard load processor (CFLOAD01) is called. Control returns to the secure volume processor from either of these routines, and the secure volume prompt is redisplayed.

#### *Standard Load Processor (CFLOAD01)*

The standard load processor displays the standard load prompt and issues a LOAD instruction to load a partition. This routine should be called via an external status subroutine when the operator presses the System Request key or when you want to exit a foreground partition.

#### *Input*

- BR32—Address of your register save area.
- I158—This indicator may be set by your program or by the attach partition routine (CFATFBGD).

#### *Output*

- The partition is loaded.
- Registers are restored.

When the routine is executed, the following operations are performed.

1. The following prompt is displayed on the screen:

```
0001      A 17 40
Program name:
Device address:
Partition number:
                                Press ENTER
                                05-00
```

2. If the operator enters the load parameters and presses the Enter key, the input is accepted. A LOAD instruction is issued, with the input used as parameters. The LOAD is issued with the attach option. See the LOAD instruction in Chapter 4 for more information about the load operation.

If the operator presses the Cncl key or EOJ key instead of the Enter key, or if the operator is loading another partition, return is made to the partition, to the next sequential instruction.

If the operator presses the Attn key instead of the Enter key, the allow detach routine (CFPERATT) is called.

3. If I158 is on, control will not return until the newly loaded partition detaches from the keyboard (calls CFPERATT). If I158 is off, control will return immediately after the LOAD is performed. If the partition calling CFLOAD01 does not require a keyboard to continue processing, setting off I158 will allow it to run concurrently with the newly loaded partition.
4. If the load parameters entered specify the same partition which called CFLOAD01, or another partition which already is executing a program, a warning message (0105) will be displayed. The warning cannot be overridden.

## KEYBOARD/DISPLAY EXTERNAL STATUS

The I/O instruction that initiates key entry is the ENTR command. When the IBM 5280 encounters an ENTR command, it issues the command to the keyboard/display. The ENTR command specifies the format of the record as it appears on the screen. The keyboard/display uses this screen format to display prompts and accept input fields for each record. You use the .SFMT series of control statements in your source program to describe the screen format for a record. Screen formats are described in detail in Chapter 2.

Normal key entry is processed by the keyboard/display without assistance from the IBM 5280 controller. An external status condition occurs when the keyboard/display unit encounters a situation that does require processing by the controller. When such a condition occurs, the keyboard/display interrupts key entry. The controller is notified that the keyboard/display needs assistance. Indicators are turned on in associated IOBs to indicate that an external status condition is outstanding. Key entry cannot be resumed until the external status indicators are turned off.

When the controller detects the outstanding external status condition, it places certain information in the keyboard/display IOB and into certain system registers within the partition. It then calls the subroutine you have written to process the condition. Your subroutine can use the information in the IOB and system registers. The information in the IOB depends upon the particular condition and is discussed in the following external status condition descriptions. Except as noted in the condition descriptions, the registers contain the information as described in Chapter 2 under *External Status and Error Conditions*.

| <b>Register</b> | <b>Information</b>  |
|-----------------|---|
| BR19            | Used only with keyboard/display external status, it contains the relative address of the start of the current field in the I/O buffer that holds the current record. The address is relative to the beginning of the partition. The contents of this register are valid only if bits 0-3 of BR21 indicate a field specification.  |
| BR20            | Used only with keyboard/display external status, it contains the absolute address of the start of the current field in the screen refresh buffer. The screen refresh buffer is located within the keyboard/display storage, and holds the data that appears on the screen. The contents of this register are valid only if bits 0-3 of BR21 indicate a field specification. |
| BR21            | Used only with keyboard/display external status, this register contains information about the type of specification being processed when the external status condition occurred, as follows:  |

| <b>Bits</b> | <b>Meaning</b>   |
|-------------|--|
| 0-3         | Field type or format specification<br>0000 = picture check field<br>0001 = alphabetic shift field<br>0010 = numeric shift field<br>0011 = hex field<br>0100 = special characters shift field<br>0101 = format level zero field<br>0110 = fixed position prompt <sup>1</sup><br>0111 = standard prompt or constant insert <sup>1</sup><br>1001 = alphabetic only field<br>1010 = numeric only field<br>1011 = digits only field<br>1100 = special characters only field<br>1110 = display attribute <sup>1</sup><br>1111 = control specification <sup>1</sup> |
| 4           | 0 = not a signed numeric field<br>1 = a signed numeric field   |
| 5-15        | Field length of the field, minus 1   |

BR23 Used only with keyboard/display external status, this contains the ETAB index of the current external status condition.

---

<sup>1</sup> These specifications are not field specifications; when any of them is used, and the data in the remainder of BR21, and in BR19 and BR20 is not valid.

## Restricted External Status Indicator

By setting a system indicator, I108, you can restrict which external status conditions are posted to the partition and determine the way they are posted. If I108 is on when external status is enabled, the only conditions normally posted are external status conditions 4, 5, 9, 12, 14, 15, 16, and 17, which are described below. All others are treated as though an external status condition was already outstanding when they occurred.

## External Status Subroutines

External status subroutines must resolve the external status condition and turn off the external status indicators. In addition, in most cases, they must either provide for resuming key entry under the current ENTR command or cancel the current ENTR. These operations can be performed by including the following instructions in the external status subroutines:

ENABLE (turn off external status indicators)  
RESUME (resume key entry)  
RESCAL (RESUME and call a subroutine)  
RETURN (return from subroutine)  
RETEXT (RETURN and ENABLE)  
RESMXT (RESUME and RETEXT)  
CNENTR (cancel current ENTR)

The external status indicators can be cleared by either the RETEXT instruction or the ENABLE instruction. If you want to clear external status indicators before you end the subroutine, you can include an ENABLE instruction in any position in your subroutine. The last statement in your subroutine must be a RETURN instruction. If you wish to clear external status indicators when your subroutine has completed executing, you can include a RETEXT instruction as the last instruction statement in your subroutine. This acts as both the ENABLE and RETURN instructions; it clears the external status indicators and ends the subroutine.

The RESUME instruction resumes key entry under the current ENTR command. The RESUME instruction neither causes a branch nor clears external status indicators. When the RESUME instruction is executed, the keyboard is unlocked and the keyboard/display resumes processing the screen format (specified by the current ENTR) at the point where the format was interrupted. The RESCAL instruction can be used in place of RESUME to resume key entry. The RESCAL instruction performs the same operation as RESUME, and in addition calls another subroutine through a label table. Or the RESMXT instruction can be used to perform both the RESUME and RETEXT operations. The RETURN instruction is described under *Subroutine Call and Return* instructions in Chapter 4. The RESUME, ENABLE, RESCAL, RETEXT, and RESMXT instructions are described under *Key Entry Instructions* in Chapter 4.

Should you wish to cancel the current ENTR command rather than resuming key entry, you can issue a CNENTR instruction. This instruction is described under *Keyboard Operations* in Chapter 4.

If an external status subroutine issues a CNENTR, RESUME or RESCAL instruction when no ENTR command is being processed, the instruction is ignored. Therefore, you may include these instructions for conditions that may occur whether or not an ENTR is outstanding.

You can write one subroutine to handle all external status conditions, or you can write a separate subroutine to handle each condition. If you use one subroutine for all external status conditions, you must specify the label of the subroutine for the ELAB parameter of the .KBCRT control statement. Whenever any external status condition occurs, the 5280 will branch to this subroutine. This subroutine must clear external status indicators and resume or cancel the current ENTR command.

If you write a separate subroutine for each condition, you must use the .LABTAB control statement to enter the labels of the subroutines into a label table. The label of the subroutine to handle condition 0 must be entered into the table first; the label for the condition 1 subroutine must be entered next, and so on. You must specify the label of the label table in the ETAB parameter of the .KBCRT control statement. Whenever an external status condition occurs, the 5280 uses the external status condition number in BR23 as the index for an indexed subroutine call through the label table. It branches to the subroutine address that is entered at that index position.

Return from an external status subroutine depends upon the external status condition. The following descriptions of the external status conditions include how to code a return from the particular external status subroutine, and whether the current ENTR is resumed, canceled or completed. An ENTR is completed when all the specifications of the screen format have been processed.

### **External Status Conditions**

Each external status condition is specified by a condition number. The 5280 stores this number into BR23 when an external status condition occurs. If you write a separate subroutine to handle each condition, the 5280 uses this number as the index into your external status label table (ETAB) to call the appropriate subroutine.

In the following condition descriptions, an external status condition is outstanding until the external status indicators are cleared by an ENABLE, RETEXT, or RESMXT instruction.

#### ***Condition 0: Double External Status***

This condition occurs when an external status subroutine is interrupted by a second external status condition. The second condition may be condition 4, 5, 6, 7, 10, 13, 14, 15, 16, or 17. It also occurs when the Restricted External Status Indicator (I108) is set on, and external status condition 6, 7, 10, or 13 occurs. Condition 0 results from programming errors. If an ENTR command is being processed when condition 0 occurs, the external status subroutine cannot recover normal key entry by issuing a RESUME.

When condition 0 occurs, the contents of BR19, BR20, and BR21 are meaningful if there is an ENTR outstanding and the cursor is currently positioned within a field. When condition 0 occurs, the following status information is stored in the keyboard/display IOB:

| Relative Address | Status Information  |
|------------------|---|
| Hex FE           | The condition number of the external status condition that caused condition 0. This information remains valid until (1) an ENTR command is executed, (2) a keyboard operation (including RESUME, RESCAL, RETEXT, and RESMXT) is executed, or (3) an external status condition occurs. |

#### *Condition 1: Function Key*

Condition 1 results when the operator presses a function key that requires processing by a subroutine.

If condition 1 occurs while an external status condition is outstanding or while the restricted external status indicator (I108) is on, it does not cause condition 0. Keyboard error 1170, the code for a software overrun error, is displayed on the status line unless a keystroke error or program error is already outstanding. The function for the keystroke is not processed.

When condition 1 occurs the contents of BR19, BR20, and BR21 are meaningful if there is an ENTR outstanding and the cursor is currently positioned within a field.

When condition 1 occurs, the following status information is stored in the keyboard/display IOB:

| Relative Address | Status Information  |
|------------------|---|
| Hex A7           | EBCDIC code of the function key. This information remains valid until (1) an ENTR command is executed, (2) a keyboard operation (including RESUME, RESCAL, RETEXT, and RESMXT) is executed, or (3) an external status condition occurs. |

Following is a list of the functions.

#### **Attention Function**

The attention function is initiated when the operator presses the Attn (Attention) key. The purpose of the function is defined by your program. This key is normally used to call the common function routine CFPERATT. See *Allow Detach Routine* earlier in this chapter.

### **Auto-Enter Function**

The auto-enter function is initiated when the operator presses the Auto Enter key on the data entry or proof keyboard. (On a typewriter keyboard it must be initiated by a command function key sequence and handled by the external status 2 or 3 subroutine.) The purpose of the function is to set the auto-enter flag in the keyboard function control flag byte.

When this bit is set to 1, the automatic record advance function is enabled.

### **Auto-Dup/Skip Function**

The auto-dup/skip function is initiated when the operator presses the Dup Skip key on the data entry or proof keyboard. The purpose of the function is to set the auto-dup/skip flag in the keyboard function control flag byte.

### **Cancel Function**

The cancel function is initiated when the operator presses the Cncl (Cancel) key. Your program defines the purpose of the function and processes the function.

### **Erase Function**

The erase function is initiated when the operator presses the Erase Input key. The purpose of the function is defined by your program.

### **Help Function**

The help function is initiated when the operator presses the Help key. The purpose of the function is to move a message to the screen. This key is normally used to call the common function routine CFHELP01. See *Help Text Processor* earlier in this chapter.

### **Next Format Function**

The next format function is initiated when the operator presses the Next Fmt (Next Format) key on the data entry or proof keyboard. The purpose of the function is to allow the operator to leave a repetitive format.

### **Page Forward Function**

The page forward function is initiated when the operator presses the Page Fwd (Page Forward) key on the data entry or proof keyboard. The purpose of the function is to read the next record without writing the current record.

### **Print Function**

The print function is initiated when the operator presses the Print key. The purpose of the function is to specify printed output.

### **Record Correct Function**

The record correct function is initiated when the operator presses the shifted Corr (Correct) key on the data entry or proof keyboard. The purpose of the function is to change from verify mode to enter mode to allow the operator to reenter an entire record, then change back to verify mode to reverify the entire record.

### **Select Format Function**

The select format function is initiated when the operator presses the Sel Fmt (Select Format) key on the data entry or proof keyboard. The purpose of the function is to allow the operator to select a screen format for data entry.

### **System Request Function**

The system request function is initiated when the operator presses the Sys Req (System Request) key. This key is normally used to call the common function routine CFLOAD01. See *Standard Load Processor* earlier in this chapter.

### ***Condition 2: Command Key Sequence, Lowercase***

This condition occurs when the operator presses a command key sequence that requires processing by a subroutine, and the second key is in lowercase. Your program defines and processes the command functions. If condition 2 occurs while another external status condition is outstanding, or while the restricted external status indicator (I108) is on, condition 0 does not result. Keyboard error 1170, the code for a software overrun error, is displayed on the status line unless a keystroke error or program error is already outstanding. The key-sequence command is not processed.

When condition 2 occurs, the contents of BR19, BR20, and BR21 are meaningful if there is an ENTR outstanding and the cursor is currently positioned within a field.

When condition 2 occurs, the following information is stored in the keyboard/display IOB. This information remains valid until (1) an ENTR command is executed, (2) a keyboard operation (including RESUME, RESCAL, RETEXT, and RESMXT) is executed, or (3) an external status condition occurs.

| Relative Address | Status Information   |
|------------------|--|
| Hex A6           | Scan code of the second keystroke. (The scan code is a unique code generated by the keyboard.) |
| Hex A7           | EBCDIC of the second keystroke.  |

***Condition 3: Command Key Sequence, Uppercase***

This condition occurs when the operator presses a command key sequence that requires processing by a subroutine, and the second key is in uppercase. Your program defines and processes the command functions. Condition 3 is as for condition 2 except for the shift of the command key. The IOB status information and register information is as for condition 2.

***Condition 4: Forward Pass over Return (RG) Specification***

This condition occurs during formatted key entry under an ENTR command; the screen format control string is being processed in a forward direction when an RG specification is encountered. (See note following Condition 5 for an explanation of the RG specification.) Key entry is suspended; the Reset key or a shift key are the only keystrokes that may be entered. If any other key is pressed, an error occurs.

The subroutine that processes condition 4 must include a RESUME command before data keystrokes or function key sequences may be entered. If any other key is pressed, an error occurs.

If condition 4 occurs while another external status condition is being processed, condition 0 results. Key entry cannot be resumed with a RESUME command in the condition 0 subroutine.

When condition 4 occurs, the contents of BR19, BR20, and BR21 pertain to the last field exited in the forward direction. The contents of these registers are undefined if condition 4 occurs while a CI (check indicator for bypass) specification is being processed, or if the RG specification is encountered before the first field definition.

***Condition 5: Backward Pass over Return (RG) Specification***

This condition occurs during formatted key entry under an ENTR command; the screen format control string is being processed in a backward direction — such as a backspace — when an RG specification is encountered. (See note below for an explanation of the RG specification.) Key entry is suspended; the Reset key or a shift key are the only keystrokes that may be entered. If any other key is pressed, an error occurs.

The subroutine that processes condition 5 must issue a RESUME command before data keystrokes or function key sequences may be entered. If any other key is pressed, an error occurs.

If condition 5 occurs, the contents of BR19, BR20, and BR21 pertain to the last field exited in the backward direction. If the RG specification is part of a field definition, the last field exited is the next sequential field in the forward direction. The contents of the registers are undefined if condition 5 occurs while a CI (check indicator for bypass) specification is being processed, or if the RG specification is encountered before the first field definition.

**Note:** You include a return (RG) specification for a CNTL parameter in your screen control format whenever you want to temporarily interrupt key entry to execute program instruction. See *Screen Formats* in Chapter 2 and the .SFMT series of control statements in Chapter 3 for more information about the RG specification.

#### *Condition 6: Record Advance*

This condition occurs during formatted key entry under an ENTR command when all fields within the current screen control format have been processed and the end of record functions have been processed. The current ENTR command is completed. After the external status subroutine has executed, control returns to the next sequential instruction after the ENTR command, if the ENTR is non-overlap.

If this condition occurs while another external status condition is being processed, or while the restricted external status indicator is on (I108), condition 0 results.

When condition 6 occurs, the contents of BR19, BR20, and BR21 pertain to the last field defined in the screen format.

#### *Condition 7: Record Backspace*

This condition occurs during formatted key entry under an ENTR command, when the screen control format is at the first position of the first manual field and the operator presses the Home key. The ENTR command is made complete.

If this condition occurs while another external status condition is being processed or while the restricted external status indicator is on, condition 0 results.

When condition 7 occurs, the contents of BR19, BR20, and BR21 pertain to the first field defined in the screen format.

### **Condition 8: Keystroke Error**

This condition occurs when a keystroke error has occurred and you have specified the TRAP parameter of the .KBCRT control statement in Chapter 3.

If this condition occurs while another external status condition is being processed, or while the restricted external status indicator (I108) is set on, condition 0 does not result. Keyboard error 1170, the code for a software overrun error, is displayed on the status line unless a keystroke or program error is already outstanding.

When condition 8 occurs, the contents of BR19, BR20, and BR21 are meaningful if an ENTR is outstanding and the cursor is currently positioned within a field.

When condition 8 occurs, the following status information is stored in the keyboard/display IOB. This information remains valid until (1) an ENTR command is executed, (2) a keyboard operation (including RESUME, RESCAL, RETEXT, and RESMXT) is executed, or (3) an external status condition occurs.

| <b>Relative Address</b> | <b>Status Information</b>          |
|-------------------------|------------------------------------|
| Hex 84                  | Hex code of the keystroke error    |
| Hex A6                  | Scan code of the keystroke error   |
| Hex A7                  | EBCDIC code of the keystroke error |

### **Condition 9: Keyboard/Display Storage Parity Error**

This condition occurs when a keyboard/display storage parity error is encountered and logged into the hard error log. This error normally occurs when keyboard/display storage is accessed to process a function. If a keyboard operation caused the parity error, the error is not reported until after the operation is completed. You are responsible for error recovery. You may choose to either abort the job or continue on the basis of the status information stored in the keyboard/display IOB. If condition 9 occurs when the restricted external status indicator (I108) is on but no other external status condition is outstanding, condition 9 is processed normally. If a condition 9 occurs while another condition 9 is being processed, condition 0 does not result and the second condition 9 is not displayed on the status line. Normal hard error logging is processed for the second condition 9. If condition 9 occurs while another external status condition (other than condition 9) is being processed, condition 12 results.

When condition 9 occurs, the contents of BR19, BR20, and BR21 are meaningful if an ENTR is outstanding and the cursor is currently positioned within a field.

When condition 9 occurs during normal operation, the following status information is stored in the keyboard/display IOB.

| <b>Relative Address</b> | <b>Status Information</b>   |
|-------------------------|---|
| Hex FE                  | If bits 1 and 2 equal 00, it indicates an invalid keyboard/display storage address, which may have resulted from a programming error; bits 3-5 indicate the high order 3 bits of the absolute address in keyboard/display storage where the error occurred. |

***Condition 10: Invalid Operation***

This condition occurs when any invalid operation request is detected. If condition 10 occurs while another external status condition is being processed or while the restricted external status indicator is on, condition 0 results.

When condition 10 occurs, the contents of BR19, BR20, and BR21 are meaningful if an ENTR is outstanding and the cursor is positioned within a field.

***Condition 11: Magnetic Stripe Reader Request***

This condition occurs after a badge has been inserted into the magnetic stripe reader, when the badge information has been entered into the magnetic stripe reader buffer. If condition 11 occurs while another external status condition is being processed, the 5280 waits for the other external status condition processing to complete before issuing the reader request. If condition 11 occurs while the restricted external status indicator (I108) is on, the 5280 waits until it is turned off before issuing the reader request.

When condition 11 occurs, the contents of BR19, BR20, BR21 are meaningful if an ENTR is outstanding and the cursor is currently positioned within a field.

***Condition 12: Keyboard/Display Storage Parity Error Double Condition***

This condition occurs when a keyboard/display storage parity error (condition 9) occurs while another external status condition (other than a condition 9) is outstanding. The condition 9 is logged into the hard error log. The subroutine that is processing the other external status condition is interrupted, and a branch is made to the condition 12 subroutine. The condition 12 subroutine should be terminated with a RETURN command rather than a RETEXT, so control returns to the interrupted subroutine. When condition 12 occurs, the contents of BR19, BR20 and BR21 is meaningful if an ENTR is outstanding and the cursor is currently positioned within a field. The status information stored in the keyboard/display IOB is as for condition 9.

### Condition 13: Screen Format Error

This condition occurs during formatted key entry when an error is detected in the syntax of the current screen format control string. The screen format control string is the object code generated from one series of .SFMT control statements. See the *Functions Reference Manual* for a complete description of the bytes within a screen format control. When condition 13 occurs, the currently executing ENTR command is made complete. The following list indicates the kinds of errors that cause condition 13.

- The primary or secondary screen format has more than one start (.SFMTST) or end (.SFMTEND) specification.
- An end (.SFMTEND) specification is encountered while processing within a bypass specification. (See *Conditional Bypass* under *Field Control* in Chapter 2 for a description of a bypass specification.)
- The screen format control string has nested bypass specifications.
- The screen format control string has nested secondary formats. (See *Secondary Screen Format* under *Field Control* in Chapter 2 for a description.)
- The screen format control string has unmatched bypass specifications to start bypass and end bypass.
- The screen format control string has an invalid character set byte.
- The screen format control string has an invalid subfield (PIC) specification.
- The screen format control string has an invalid control byte.

If a condition 13 occurs when another external status condition is being processed, or while the restricted external status indicator is on, condition 0 results. When condition 13 occurs, the contents of BR19, BR20, and BR21 are meaningful if an ENTR is outstanding and the cursor is currently positioned within a field.

When condition 13 occurs, the following information is stored in the keyboard/display IOB. This information remains valid until (1) an ENTR command is executed, (2) a keyboard operation (including RESUME, RESCAL, RETEXT, and RESMXT) is executed, or (3) an external status condition occurs.

| Relative Address | Status Information  |
|------------------|---|
| Hex 89-8A        | Relative address of the byte within the screen format control string that caused the error. |

**Condition 14: IBM 3270 Software-Supported Function Key**

An IBM 3270 function key has been pressed which requires processing by software. KB/CRT IOB byte at offset 26 (last scan code) contains the scan code of the key; the byte at offset 27 (last translated scan code) contains a code identifying the key as indicated below.

The following software supported keys are IBM 3270 aid generating keys. The KB/CRT IOB byte 27 contains the aid code as indicated below:

| Key     | Aid |
|---------|-----|
| SYS REQ | F0  |
| PA1     | 6C  |
| PA2     | 6E  |
| PA3     | 6B  |
| ENTER   | 7D  |
| RESET   | 60  |

For the following software supported keys, the KB/CRT IOB byte 27 will contain the internal codes as indicated below:

| Key        | Internal Code |
|------------|---------------|
| PRINT      | E0            |
| ATTEN      | E1            |
| DEV CANCEL | E2            |

The input inhibited condition is set and the keyboard is hard locked.

If a software supported function key must be posted to the software and external status is already outstanding, external status code 0 — double external status — is posted. This condition indicates a fatal software error.

This external status condition can only occur while in IBM 3270 mode.

**Condition 15: IBM 3270 Command Key Sequence, Lowercase**

An IBM 3270 function command key sequence has been pressed. The IOB byte at offset 26 (last scan code) contains the scan code of the key following the command key. That key was unshifted.

The input inhibited condition is set and the keyboard is hard locked.

If a command key sequence must be posted to the software and external status is already outstanding, external status code 0 — double external status — is posted. This condition indicates a fatal software error.

This external status condition can only occur while in IBM 3270 mode.

**Condition 16: IBM 3270 Command Key Sequence, Uppercase**

An IBM 3270 function command key sequence has been pressed. The IOB byte at offset 26 (last scan code) contains the scan code of the key following the command key. That key was shifted.

The input inhibited condition is set and the keyboard is hard locked.

If a command key sequence must be posted to the software and external status is already outstanding, external status code 0 — double external status — is posted. This condition indicates a fatal software error.

This external status condition can only occur while in IBM 3270 mode.

**Condition 17: Data Stream Error**

- A. An error has been detected during translation of a write data stream. The KB/CRT IOB contains an error code in the bytes at offset X'04' — X'05' into the IOB indicating the type of error detected.

Currently defined errors are:

1190 — Invalid address specified in an SBA, EUA, or RA order.

1191 — Invalid order length. An order was detected that required one or more bytes to follow. The data stream did not contain these bytes.

- B. An error has been detected during a read, read modified, or read SSCP—SLU assist operation. The KB/CRT IOB contains an error code in bytes at offset X'04' — X'05' into the IOB indicating the type of error detected.

Currently defined errors are:

1490 — The length of the data stream exceeds the size of the data stream area specified.

1491 — The data stream area specified is less than the minimum allowable size.

If a data stream error must be posted to the software and external status is already outstanding, external status code 0, double external status, is posted. This condition indicates a fatal software error.

For screen emulation, this external status condition can occur only in IBM 3270 mode. For printer emulation, this external status condition can occur in IBM 5280 mode.

**This page is intentionally left blank**

## Chapter 7. The ACL to Assembler Language Conversion Program

The ACL (Application Control Language) to assembler language conversion program helps you convert source programs written in ACL to assembler language source programs. Use the conversion program each time you convert an ACL program to an assembler language program. The conversion program should be used only once with each program that is converted. No optimization is attempted by the conversion program. Any further changes or modifications made to the converted programs should be done directly in assembler language.

The conversion program converts most ACL instructions and control statements to assembler language. However, certain control statements and instructions cannot be converted by the conversion program; these must be converted to assembler language manually. The assembler language output generated by the ACL control statements and instructions contains a message for each line of ACL code that must be converted manually. Error messages generated by the conversion program are listed in Appendix E.

### OPERATION

The conversion program is stored on the assembler diskette. Use the standard load processor to load the conversion program. The name of the conversion program data set is SYSACLCL. The assembler diskette must remain in the diskette drive while the conversion program is executing.

When the conversion program is loaded, it displays the following prompt:

*Prompt 1*

```
0 0001      A 16 40
SYSACLCL - ACL TO ASM CONVERSION AID
Insert INPUT diskette and enter:
Data set name:
Device address:
```

Press ENTER

28-01

Insert the input diskette into the selected diskette drive. Enter the name of the data set that contains the ACL program to be converted. The data set name can contain up to 17 characters. The ACL program is assumed to be error free. If it is stored on a volume protected diskette, the prompt for an owner ID is displayed. (See Prompt 5 in Chapter 5.) The input data set must remain in the diskette drive while the conversion program is executing.

Enter the device address and press the Enter key. The device address must be four alphanumeric characters. Do not use a logical device ID.

If an invalid input data set name or an invalid device address is entered, an error recovery message is displayed.

If no error occurs the following prompt is displayed.

*Prompt 2*

```
0 0001      A 16 E2
SYSACLC - ACL TO ASM CONVERSION AID
Insert OUTPUT diskette and enter:
Data set name:
Device address:          Press ENTER
```

28-02

A default data set name, **SYSIN**, is displayed with the device address you entered for the preceding prompt, which specifies that the input and output data sets are to be on the same diskette. You can change the data set name and device address by keying over the defaults.

The output data set must remain in the diskette drive while the conversion program is executing. The output data set should be at least 2.5 times the size of the input data set, and it should be empty when the conversion program begins. If the output data set is I exchange, the header label must contain a valid delete character.

If an invalid device address is entered, an error message is displayed.

If the data set name you specify for the output data set is not found, the following prompt is displayed.

*Prompt 3*

```
0001      Y 01 40
Output data set not found. Do you want it allocated?
Options are
  1. Yes
  2. No
SELECT OPTION:  _ PRESS ENTER
```

28-05

If you select 2, prompt 2 is redisplayed with the data set name and device address you entered previously. You can change the previous entry by keying over the existing data.

If you select 1, the output data set is allocated on the diskette at the specified device address. After the data set is successfully allocated, prompt 4 is displayed.

Prompt 4

```
0 0001      Y 01 40
Do you want ACL input included as comments in output?
Options are
  1. Yes
  2. No
SELECT OPTION:  _ Press ENTER
```

28-03

Enter a 1 if you want the assembler language and the original ACL statements as output, or enter a 2 if you want only the assembler language as output. If you chose to have the original ACL included in the output, each ACL line is written as a comment line. Each ACL line precedes the assembler language lines it generated.

After you enter a valid option number and press the ENTER key, the conversion process begins. The ACL control statements and instructions are read sequentially, then translated to corresponding assembler language control statements and instructions.

If any ACL statement contains source code that the conversion program cannot convert, the conversion program inserts one of the following messages in the assembler language source at the position of the untranslatable code:

```
*MSG***Message or .MSG***Message
```

These messages allow you to assemble the rest of your assembler language source while preventing loadable object code from being produced. The message that begins with an asterisk indicates code that is loadable object code, but that may not produce the results you expect. The message that begins with a period indicates code that you must change. If your program generates a message that begins with a period, the following prompt is also displayed:

Prompt 5

```
0
UNTRANSLATABLE CODE HAS BEEN FOUND. THE ASM SOURCE CANNOT BE
ASSEMBLED. CHECK THE ASM SOURCE FOR MESSAGES.
```

Press ENTER

28-04

Each ACL statement that cannot be converted by the conversion program must be converted manually.

When the ACL to assembler language conversion program has successfully completed, the following is displayed:

*Prompt 6*

```
0 0001      Y 01 40
CONVERSION COMPLETED.
Options are
  1. Restart      3. Assemble
  2. Exit
SELECT OPTION:  _ Press ENTER
```

28-79

If you select 1, the conversion program restarts at the beginning. If you choose to exit, the standard load processor redisplay the standard load prompt. If you select 3, the conversion program loads the IBM 5280 Assembler Program Product (see Chapter 5) to process the data set output from the conversion program. When the assembler encounters the untranslatable code message, it includes an error message for that line in the assembler listing. You must convert this line to assembler language and reassemble the program.

#### NOTES ABOUT THE CONVERTED PROGRAM

The following discussion describes the format of the converted program and how the converted program may differ from the original ACL program.

##### The Format of the Converted Program

An ACL line often converts to more than one assembler language line. This can cause indexed branch instructions and dynamic instruction modification in the ACL program to produce unpredictable results in the converted program.

Each control statement in the converted program is preceded by a period in the first column of the line. Each control statement and instruction ends with a semicolon. A line that begins with an asterisk is treated as a comment line.

##### Labels and Sequence Numbers

If a label is present in an ACL line, a colon is inserted in column 5, and this label precedes the assembler language line or lines generated from the ACL line.

If a sequence number is present in an ACL line, the number is preceded by N and used as a label on the assembler language line generated from the ACL line. If more than one assembler language line is generated, the label is placed on the first assembler language line generated from the ACL line.

If an ACL line does not have a sequence number, the conversion aide generates a sequence number. The generated number is preceded by N and placed on the assembler language line, or the first of the assembler language lines, generated from the ACL line. For example, if the ACL program has the following instruction in line 0007:

```
AAAA IFD K = 0 009
```

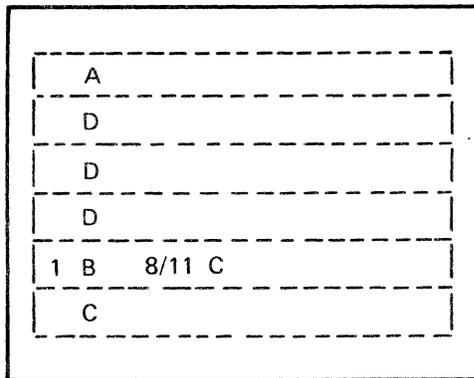
it is converted to the following assembler language code:

```
AAAA:
N0007: IFD K NE 0 GOTO N0008:
      GOTO N0009
```

### The Format of the Display Screen

The following illustration shows the format of the screen for ACL, and the format of the screen for a converted program.

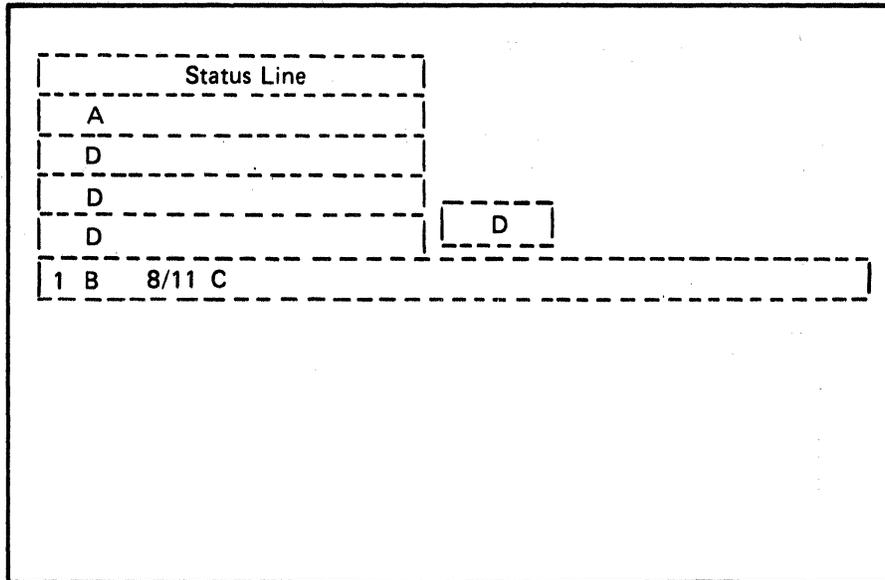
#### ACL Screen Format



| Screen Position | ACL    |           | ASM    |           |
|-----------------|--------|-----------|--------|-----------|
|                 | Buffer | Positions | Buffer | Positions |
| A               | 01     | 1-40      | 01     | 1-40      |
| B               | 01     | 41-48     | 01     | 41-48     |
| (B)             | (02)   | (121-128) |        |           |
| C               | 01     | 51-120    | 01     | 51-120    |
| D               | 02     | 1-120     | 02     | 1-128     |

( ) implies 'B' or 'M' option in col 28 on .FIELD

## IBM 5280 Screen Format



In ACL, buffers 01 and 02 contain data that is displayed. On the IBM 5280, a screen refresh buffer in keyboard/display storage maintains the screen display according to the screen control format referred to in the current ENTR command. The conversion program provides for placing the contents of buffers 01 and 02 into the screen refresh buffer by inserting the following instruction in the converted program:

```
ENTR(SFMTXXX);
```

This instruction is frequently generated before GOTO or IF type instructions if the contents of buffer 01 or 02 have been modified. The flow of instructions after an overlapped ENTR is executed might encounter one of these special ENTRs. If this occurs, overlapping is suspended and the overlapped ENTR must be completed before further instruction execution.

Users may freely delete or insert the special ENTR instruction in their converted programs.

Response areas will appear immediately following the prompt text on the display screen. If the prompt appears in two places (buffer 2 area and fixed prompt line), the response area will follow the first prompt. (On the 3741, the response was duplicated in both places.)

The response to a prompt is placed in buffer 2 but not in buffer 1 for options B, M, and D. The response is placed in the register and buffer 1 for option R, and in the register and buffer 2 for option D.

The following chart describes the responses to prompts.

| Option<br>Col 28 | ACL                  |                       |                         | ASM      |                       |                |
|------------------|----------------------|-----------------------|-------------------------|----------|-----------------------|----------------|
|                  | Prompt               | Response<br>Displayed | Data<br>Stored          | Prompt   | Response<br>Displayed | Data<br>Stored |
| B or<br>Blank    | Line 5-6             | Line 5-6<br>Line 2-4  | Buff 1<br>Buff 2        | Line 6   | Line 6                | Buff 2<br>only |
| R                | Line 5-6             | Line 5-6              | Buff 1<br>Reg           | Line 6   | Line 6                | Buff 1<br>Reg  |
| M                | Line 5-6<br>Line 2-4 | Line 5-6<br>Line 2-4  | Buff 1<br>Buff 2        | Line 3-5 | Line 3-5              | Buff 2<br>only |
| D                | Line 5-6             | Line 5-6<br>Line 2-4  | Buff 1<br>Buff 2<br>Reg | Line 6   | Line 6                | Buff 2<br>Reg  |

### Buffers

The conversion aid does not automatically declare all 56 buffers. Only the following buffers are declared:

- Buffers 01 and 02. These buffers are contiguous and precede all other buffers.
- Buffers initialized by a .BUFFER control statement in the ACL program. These buffers are contiguous if their associated .BUFFER control statements are contiguous.
- Buffers referred to in instructions that are not declared by methods (1) or (2).

The ACL conversion program does not always assign buffers to contiguous storage. Specifically, any buffers that are initialized are allocated when the .BUFFER statement is encountered. Any other buffers that are referenced in the program are allocated in numerical sequence following the ACL .END statement. In particular, ACL table buffers must be contiguous. They will not be if some are declared and some are not. Buffers should not be declared by two .BUFFER statements with the same buffer number (multiple definition).

### *Implied Usage of Buffers*

Implied usage of a nondeclared buffer may produce unpredictable results. For example, the ACL instruction PRNT allows buffer specification for a print buffer. The buffer referred to in the PRNT instruction will be declared as a 128-byte buffer. However, the ACL .PRINTER control statement could specify up to 132 characters per line. In ACL, the print buffer referred to in the PRNT instruction is used for the first 128 characters, and the next sequential buffer is used implicitly for the remaining characters. The conversion aid does not declare the implied buffer. In the converted program, the declared print buffer is used for the first 128 characters and the next sequential *declared* buffer is used for any remaining characters. If you use more than 128 characters per line, it is recommended that you increase the size of the declared print buffer in the converted program.

Unlike ACL, the converted program does not store the .FIELD and .FORMAT; information in the buffers. Therefore, any attempt to modify .FIELD statements during execution of the ACL program by referring to the buffer in which they are stored requires manual translation.

### *The Physical Buffer Size*

ACL programs use basic exchange diskettes, which require a physical buffer size of 128 bytes. If you plan to use a different exchange type for your converted program, be sure your physical buffer size is valid for that exchange. Exchange, for example, requires a physical buffer size at least twice the diskette sector size.

### *FIELD Buffers*

The 5280 does not use buffers for the .FIELD control statements in the same way the 3741 used them for ACL programs. The conversion aid converts the .FIELD control statements to a screen control format and stores the screen control format in the next available storage location.

The conversion aid does not handle all the options that were available in ACL to assign to buffers in the .FIELD control statements. This does not affect the functions available in the converted program.

The conversion aid requires that all .FIELD control statements that refer to the same buffer be contiguous in the ACL program. This means that if a series of .FIELD statements refer to a buffer, such as buffer 05, and then subsequent .FIELD statements refer to another buffer, such as 06, the following .FIELD statements cannot refer to the first buffer (buffer 05).

The overflow buffer (ACL columns 18-19) is ignored. If an ACL .FIELD control statement specifies overflow into the next buffer to which .FIELD control statements refer (this is usually the case), the first .FIELD control statement in the overflow buffer must be referenced as message number 1, not message number 2 as used by ACL.

The ACL .ENTR instructions refer to fields by field number, relative to the start of the buffer. The conversion aid assigns screen format numbers sequentially and converts the ACL references to these numbers. This may result in a field sequence error of one if a field overflowed from one buffer to another in the ACL program.

If you add any screen control formats to a converted program, place them following the last screen control format generated by the conversion aid in order to maintain proper numbering sequences.

### **The .FORMAT Control Statement**

The IBM 5280 uses a currency symbol that is *always* two characters in length. The conversion aid attempts to allow for this when converting ACL .FORMAT control statements.

### **Indexed Branch Instructions**

Because more than one line of assembler language instructions can be generated from one ACL instruction, indexed GOTO instructions in the ACL program may not produce the expected result in the converted program. You can replace the indexed GOTO instructions with GOTO instructions in the converted program. Or you can set up a label table of addresses with a .LABTAB control statement and replace the indexed GOTO with a GOTAB in the converted program. The GOTAB instruction uses the label table of addresses to make the indexed branches.

### **The OPEN Instructions**

The converted program does not automatically issue an OPEN instruction when an I/O instruction specifies an unopened data set IOB. The converted program will, however, try to open all data sets at the start of program execution.

The IBM 5280 cannot issue an OPEN instruction to a device that already has a data set IOB open for label update. Therefore, if the ACL program opened a diskette data set for label update, it must be closed before any other data sets on that diskette can be opened. Conversely, the IBM 5280 cannot open a device for label update if that device contains an open data set.

### **The ENTR Instructions**

In ACL, an indicator is set whenever an ENTR is executed. Any assembler ENTR instruction that was generated from an ACL ENTR instruction sets an indicator (I227 in the converted program) when the ENTR instruction is executed. If you add ENTR instructions to a converted program, this indicator is not automatically set when the added ENTR instruction is executed. If you want the indicator to be set, you must write code to set the indicator and include this code in the converted program in addition to the ENTR instruction.

Converted ACL programs will not support ENTR instructions that reference a .FIELD statement that was chained to a preceding .FIELD statement.

## The EXEC Instruction

An EXEC instruction in ACL is converted to a LOAD instruction. The LOAD parameters are built using the program name, not the data set name. For proper execution, the program name on the ACL EXEC instruction should be that of the data set name of the program.

## Keyed Data Sets

If the key requested when reading a keyed file is less than the first entry in the key table, the IBM 3741 sets an indicator and places the first record in the buffer. The IBM 5280 does not place the first record in the buffer, but instead leaves the contents of the buffer unchanged.

If the IBM 3741 finds multiple records with the same key (duplicate keys), it always stops at the first one. The IBM 5280 does not expect duplicate keys in a keyed data set. When reading a file by using a binary search, the IBM 5280 can find any one of a group of records with duplicate keys, not necessarily the first one. To request a sequential search rather than a binary search by an IBM 5280, delete the ORD specification on the TYPE parameter of the applicable .DATASET statement, and change the associated .TABLE statement to include the BYPASS parameter.

When adding a record to a keyed file, if the record key is outside the bounds of the index table, the IBM 5280 returns a 3702 error and does not position the file to allow a record to be written in the proper sequence. To add records beyond the bounds of an index table:

1. Change the lowest key of the index table to include all possible keys (for example, the lowest possible key is X'00').
2. When adding a record with a key greater than the index, position to end-of-data and then write current.

For additional information concerning keyed data sets, see the section titled *Diskette Data Management*.

## The ACL Deleted Record Subroutine

The 5280 does not read deleted records unless they are read by relative record number, which in turn results in an external status condition. The ACL deleted record subroutine, which is specified in columns 48-51 of the .DATASET control statement in the ACL program, is not processed by the conversion aid. The converted program will contain a message if columns 48-51 are not blank in the .DATASET statement in the ACL program.

## Physical Buffers

SYSACL was designed to create IBM 5280 assembler programs that handle basic exchange diskettes with 128-byte sector size. If any other exchange or sector size is used, physical buffers size should be increased.

## Print Instructions

The converted program does not use overlapped I/O for printer operations. The printer operations in the converted program use SCS conversion data sets. For this kind of data set, overlapping is invalid.

The IBM 3741 sets the printer overflow indicator the first time a line past the line specified as "last line" is given to the printer. The IBM 5280 sets the corresponding indicator every time page overflow occurs.

## Leading Blanks

Unlike the IBM 3741, the IBM 5280 does not always treat leading blanks like zeros. In particular, an IFD instruction is sometimes required in lieu of an IF instruction. Also, in some format specifications, a blanked register may not produce the same result as a zeroed register on a formatted write.

## Function Keys

In the converted program, certain function keys are substituted for the function keys used in ACL. The following lists ACL function keys and explains how these function keys are handled for a *converted* program. For a description of how all the function keys are handled for a program originally written in 5280 assembler language, see the *Functions Reference Manual*.

### *The System Request/Attention Key*

In a converted program this key is handled in the same way it is handled for any program on the 5280.

### *The Reset Key*

In ACL, the Reset key turns on an indicator in addition to resetting the error condition. If you want this indicator turned on during the execution of a converted program, press the Help key instead of the Reset key. When the operator presses the Reset key in a converted program, the error condition is reset: the status line stops blinking and the operator can enter correct keystrokes.

### *The Command/End of Job Key Sequence and the Help Key*

If there is an error during the execution of a converted program, the status line blinks and displays error information. While the status line is blinking, the operator can (1) press the Reset key to reset the error and stop the blinking, (2) press the Cmd key, then the End of Job command function key to terminate the job, or (3) press the Help key to have help text displayed.

## The Minus Key

The minus key on the IBM 5280 is treated as field minus in alpha shift and as a minus graphic in numeric shift. The IBM 3741 is just the opposite.

### *ACL Double-Shift Reset*

During the execution of a converted program, press the Cncl key in upper shift to emulate the function caused by pressing the Alpha shift, Num shift, and Reset keys simultaneously during the execution of an ACL program.

### *The ACL Tab Key*

There is no Tab key on the IBM 5280. During the execution of a converted program, press the Page Forward key to emulate the ACL Tab key function.

### *ACL T-Option Keys*

The conversion aid does not check the T-option in the ACL .FIELD control statement. Five of these keys are always handled as though the T-option in the ACL program were specified to redefine the keys. These keys include the:

- Dup key
- Sel Prog key
- Rec Adv key
- Rec Bksp key
- Field Adv key

When any of these keys is pressed during the execution of a converted program, the current ENTR command is terminated. Data that has been entered is preserved. Any specified right adjust function is not performed on the current field. Special keyboard indicator (1230 in the converted program) is set.

The Field Bksp key is handled as though the T-option in the ACL program were not selected for this key. When the Field Bksp key is pressed during the execution of a converted program, if the cursor is in any position of the field other than the first position, the cursor goes back to the first position of the field. If the key is pressed when the cursor is in the first position of the field, it is handled as for one of the preceding five T-option keys.

An indicator test for a function key may fail if the test occurs within a very tight loop. This problem may be circumvented by increasing the loop execution time. For example, the following two lines of 3741 ACL code

```
100 SON 162 BUZZ KEYBOARD
    IFIR 187 NOT ON 100
```

convert to the following IBM 5280 assembler code

```
NO100: BUZZ;
        IFIR I217 NOT ON GOTO NO100;;
```

This tight loop leaves insufficient time for keyboard external status to occur and should be changed to

```
NO100: BUZZ;  
        BR40 = X'1000';  
DECRXX: DECR BR40 GOTO DECRXX;  
        IFIR I217 NOT ON GOTO BUZZ:
```

### ACL Toggle Switches

During the execution of an ACL program, you could use a toggle switch to change the status of one of the following functions:

- Auto Rec Adv
- Auto Dup/Skip
- Numbers only field

The IBM 5280 does not have these toggle switches. During the execution of a converted program, you can emulate the function of the toggle switches by pressing the following keys:

- Auto Enter key—emulates the Auto Rec Adv switch
- Dup Skip key—emulates the Auto Dup/Skip switch
- Erase input key—emulates the Numbers Only setting on the Prog Num Shift switch

When one of these keys is pressed, if the function is disabled it is enabled; if the function is enabled it is disabled.

A reverse image character is placed on the status line as a flag to indicate which of the simulated switches are on. Key and flag assignment is as follows:

| Switch        | Key         | Flag |
|---------------|-------------|------|
| Numbers Only  | Erase Input | C    |
| Auto Skip/Dup | Skip/Dup    | D    |
| Auto Enter    | Auto Enter  | R    |

The converted ACL program will ignore any SON or SOFF for these indicators and will treat IFIR as IFI when testing them.

### CONVERSION CHART

This chart is intended as a preliminary guide for ACL programmers who are beginning to program in the IBM 5280 assembler language. It shows the IBM 5280 assembler language control statements and instruction mnemonics that most closely correspond to the ACL control statements and instruction mnemonics. All ACL control statements and instructions are listed, although not all ACL instructions have corresponding IBM 5280 assembler language instructions.

The IBM 5280 assembler language has many control statements and instructions that are not available with ACL. Instructions for table operations, conditional branches and skips, and data movement have been greatly expanded in the IBM 5280 assembler language. Instructions for binary arithmetic and logical operations have been added, along with numerous mask operations for bit manipulation. Therefore the control statement or instruction that corresponds to ACL is not necessarily the most appropriate IBM 5280 assembler language control statement or instruction to use.

### Control Statements

| ACL<br>Mnemonic | IBM 5280<br>Mnemonic   |
|-----------------|--|
| .NAME           | .START<br>.EDITC   |
| .DATASET        | .DATASET   |
| .PRINTER        | .DATASET   |
| .SELF-CHECK     | .SELFCHK   |
| .REGISTER       | .DC  |
| .FORMAT         | .FMTST<br>.FMTFLD<br>.FMTEND                                       |
| .BUFFER         | .DC  |
| .FIELD          | .SFMTST<br>.SFMTCTL<br>.SFMTFLD<br>.SFMTCNS<br>.SFMPMT<br>.SFMTEND |
| .END            | .END   |

## Instructions

| ACL Mnemonic | IBM 5280 Mnemonic           | Operation                         |
|--------------|-----------------------------|-----------------------------------|
| +            | +                           | Add                               |
| -            | -                           | Subtract                          |
| /            | /                           | Divide                            |
| *            | *                           | Multiply                          |
| Ø            | Rn =                        | Move register to register         |
| CCMD         |                             | I/O adapter command               |
| CKPT         |                             | Checkpoint                        |
| COMM         | TINIT                       | Communications linkage            |
| CRDP         |                             | Punch a card                      |
| CLOZ         | CLOZ                        | Close file                        |
| CRDR         |                             | Read a card                       |
| CRFL         |                             | Read file                         |
| CCRD         |                             | Read a record from attachment     |
| CSEL         |                             | I/O adapter control               |
| CSTR         |                             | Start character                   |
| CWFL         |                             | Write file                        |
| CWRD         |                             | Write a record to attachment      |
| ENTR         | ENTR                        | Keyboard input                    |
| EXCH         | <=>                         | Exchange                          |
| EXEC         | LOAD                        | Execute next program              |
| EXIT         | EXIT                        | Exit program                      |
| GETB         | MVC<br>Rn = disp (len, BRn) | Move data from buffer to register |

| ACL Mnemonic   | IBM 5280 Mnemonic   | Operation                  |
|--|---|----------------------------|
| GOTO   | GOTO  | Unconditional branch       |
| GSKC   | GSKC  | Generate self check        |
| ICBR   | label = constant  | Insert character in buffer |
| IF $\left\{ \begin{matrix} AN \\ SN \end{matrix} \right\}$ | IF Rn $\left\{ \begin{matrix} AN \\ SN \end{matrix} \right\}$ | Test register numeric      |
| IF >   | IF Rn GT  | If register greater than   |
| IF <   | IF Rn LT  | If register less than      |
| If =   | IF Rn EQ  | Compare register logical   |
| IF -   | IF Rn -   | If register minus          |
| IF 0   | IF Rn 0   | If register zero or blank  |
| If CHK   | If Rn Ck  | Self-check test            |
| IF CRD   |   | Test if card is busy       |
| IF FMT   | If fmt  | Test format number         |
| IF PRT   |   | If printer busy            |
| IFC  |   | Test for attachment busy   |

**Note:** This mnemonic is used for a different operation in the IBM 5280 assembler language.

|          |                  |                                      |
|----------|------------------|--------------------------------------|
| IFD >    | IFD LT           | Compare decimal for less than        |
| IFD <    | IFD GT           | Compare decimal for greater than     |
| IFD =    | IFD EQ           | Compare decimal for equal            |
| IFI      | IFI              | Test indicator                       |
| IFIR     | IFIR             | Test and reset indicator             |
| (load) + | Rn = +n          | Load positive constant into register |
| (load) - | Rn = -n          | Load negative constant into register |
| L        | SL               | Shift left                           |
| LS       | SLS              | Shift left signed                    |
| LOAD     | Rn = label (len) | Load register from buffer            |

**Note:** This mnemonic is used for a different operation in the IBM 5280 assembler language.

|      |      |   |
|------|------|---|
| MOFF | MOFF | Move partial contents to register offset    |
| MOVE | MVC  | Move buffers                                |
| MVER | MVER | Move partial contents, register to register |
| NOP  | NOP  | Null operation                              |
| OPEN | OPEN | Open a file                                 |

| ACL Mnemonic | IBM 5280 Mnemonic          | Operation                           |
|--------------|----------------------------|-------------------------------------|
| PCTL         |                            | Printer skip or space               |
| PRNT         | WRT                        | Print a line                        |
| PUTB         | MVC<br>disp(len, BRn) = Rn | Move from register to buffer        |
| R            | SR                         | Shift right                         |
| RBLK         | REBF                       | Reformat registers at buffer offset |
| READ         | READ                       | Read from [diskette]                |
| REFM         | REBF                       | Formatted read from buffer          |
| RGO          | CALL<br>RETURN             | Return transfer                     |
| RR           | SRR                        | Shift right and round               |
| RS           | SRS                        | Shift right signed                  |
| SCE          | IFC . . . IS               | Skip if character equal             |
| SCN          | IFC . . . NOT              | Skip if character not equal         |
| SOFF         | SOFF                       | Set indicators off                  |
| SON          | SON                        | Set indicators on                   |
| STOR         | label(len) = Rn            | Store register to buffer            |
| TBFN         | TBFH                       | Search table for equal/high entry   |
| TBFX         | TBFX                       | Search table for equal entry        |
| TBRD         | TBRD                       | Read table entry                    |
| TBWT         | TBWT                       | Write table entry                   |
| WAIT         | WAIT                       | Wait for I/O completion             |
| WBLK         | WRBF                       | Reformat buffer at offset           |
| WRFM         | WRBF                       | Formatted write to buffer           |
| WRT          | WRT                        | Write record to diskette            |
| WRTE         | WRT                        | Write extend                        |
| WRTS         | WRTS                       | Delete a record                     |
| ZONE         | ZONE                       | Zone part of register               |

## Indicator Conversion

Many ACL Indicators have direct counterparts in IBM 5280 assembler language, as shown in the following table. When these indicators appear in SON, SOFF, IFI, or IFIR instructions, the IBM 5280 assembler language equivalent will be substituted. Other ACL Indicators, particularly those set from the keyboard, are simulated by generated software. They are shown by numbers in parentheses. The remaining indicators are not translatable or require special handling, as shown in the footnotes. The following table lists all the ACL indicators:

| ACL Indicator | Definition             | Set On By   | Set Off By                | Assembler Indicator |
|---------------|------------------------|---|---------------------------|---------------------|
| 1-99          | User specified         | User program  | User program              | 1-99                |
| 100-146       | Reserved               |   |                           |                     |
| 147           | Printer error          | Any printer error   | User program              | None                |
| 148           | Print page overflow    | Printer reaches overflow line spec'd in col 23-25 of .PRINTER | Next prnt inst            | 115                 |
| 149           | Card I/O EOD           | ?/card read   | User program              | None                |
| 150           | Card I/O EOJ           | ?*card read   | User program              | None                |
| 151-153       | Reserved               |   |                           |                     |
| 154           | Invalid GSCK -MOD 11   | GSCK result = 10  | User program              | 117                 |
| 155           | RBLK/WBLK overflow     | Low-order of reg = 000  | User program              | (192)               |
| 156           | Table index error      | Read/write past end of table                                  | User program              | 126                 |
| 157           | Division error         | Divide by 0   | User program              | 120                 |
| 158           | Multiply overflow      | Carry from mult   | User program              | 123                 |
| 159           | Add/subtract overflow  | Carry from +/-  | User program              | 124                 |
| 160           | Machine check          | Ind 155-159 ON  | User program              | Note 1              |
| 161           | Error line             | System error causing keyboard lock or screen flash            | User program or Reset key | None                |
| 162           | Short keyboard buzz    | User program  | User program              | Note 2              |
| 163           | Table high entry found | = Not found but > entry found                                 | User program or hdwr      | 101                 |
| 164           | Printer busy           | Print command   | Print done                | Note 3              |
| 165           | Disk drive busy        | Disk busy   | Disk not busy             | Note 4              |
| 166           | Auto rec adv switch    | Switch on   | Switch off                | (212)               |
| 167           | Prog num shift         | Switch on   | Switch off                | (213)               |
| 168           | Auto dup/skip switch   | Switch on   | Switch off                | (214)               |
| 169-184       | Reserved               |   |                           |                     |
| 185           | Sel pgm                | Key pressed   | User program/<br>ENTER    | (215)               |

| ACL Indicator | Definition                | Set On By                                  | Set Off By  | Assembler Indicator |
|---------------|---------------------------|--|---|---------------------|
| 186           | Dup                       | Key pressed                                | User program/<br>ENTER  | (216)               |
| 187           | *Field cor                | Key pressed                                | User program  | (217)               |
| 188           | *New line                 | Key pressed                                | User program/*  | (218)               |
| 189           | *Tab                      | Key pressed                                | User program/*  | (219)               |
| 190           | Rec bksp                  | Key pressed                                | User program/<br>ENTR   | (220)               |
| 191           | *Char adv                 | Key pressed                                | User program/*  | (221)               |
| 192           | *Reset                    | Key pressed                                | User program/*  | (222)               |
| 193           | Field adv                 | Key pressed                                | User program/<br>ENTR   | (223)               |
| 194           | Skip                      | Key pressed                                | User program/<br>ENTR   | (224)               |
| 195           | Right adj                 | Key pressed                                | User program/<br>ENTR   | (225)               |
| 196           | Neg right adjust<br>(-)   | Key pressed                                | User program/<br>ENTR   | (226)               |
| 197           | Keyboard open             | ENTR                                       | ENTR complete   | (227)               |
| 198           | Field bksp                | Key pressed                                | User program/<br>ENTR   | (228)               |
| 199           | Rec adv                   | Key pressed                                | User program/<br>ENTR   | (229)               |
| 200           | Special key-<br>board ind | Ind 185, 186, 190,<br>193, 198, and<br>199 | User program/<br>ENTR   | (230)               |
| 201           | Lower char adv            | Key pressed                                | These may be set<br>off by reset, ind<br>187-189, 191,<br>or the next<br>Func Sel or by<br>the user program | (231)               |
| 202           | Lower dup                 | Key pressed                                |   | (232)               |
| 203           | Lower field cor           | Key pressed                                |   | (233)               |
| 204           | Lower <                   | Key pressed                                |   | (234)               |
| 205           | Lower #                   | Key pressed                                |   | (235)               |
| 206           | Lower %                   | Key pressed                                |   | (236)               |
| 207           | Lower /                   | Key pressed                                |   | (237)               |
| 208           | Lower hex                 | Key pressed                                |   | (238)               |
| 209           | Upper %                   | Key pressed                                |   | (239)               |
| 210           | Upper /                   | Key pressed                                |   | (240)               |
| 211           | Upper hex                 | Key pressed                                |   | (241)               |
| 212           | Upper -                   | Key pressed                                |   | (242)               |
| 213           | Lower rec adv             | Key pressed                                |   | (243)               |
| 214           | Lower @                   | Key pressed                                |   | (244)               |
| 215           | Lower sel prog            | Key pressed                                |   | (245)               |
| 216           | Lower field adv           | Key pressed                                |   | (246)               |
| 217           | Upper @                   | Key pressed                                |   | (247)               |
| 218           | Upper sel prog            | Key pressed                                |   | (248)               |
| 219           | Upper field adv           | Key pressed                                |   | (249)               |
| 220           | Upper char adv            | Key pressed                                |   | (250)               |
| 221           | Upper dup                 | Key pressed                                |   | (251)               |
| 222           | Upper field cor           | Key pressed                                |   | (252)               |
| 223           | Upper <                   | Key pressed                                |   | (253)               |
| 224           | Upper *                   | Key pressed                                |   | (254)               |

| ACL Indicator | Definition                | Set On By      | Set Off By               | Assembler Indicator |
|---------------|---------------------------|----------------|--------------------------|---------------------|
| 225           | No record found<br>DSN 1  | Data mgmt      | User program             | (208)               |
| 226           | No record found<br>DSN 2  | Data mgmt      | User program             | (209)               |
| 227           | No record found<br>DSN 3  | Data mgmt      | User program             | (210)               |
| 228           | No record found<br>DSN 4  | Data mgmt      | User program             | (211)               |
| 229           | Key not in<br>table DSN 1 | Data mgmt      | User program<br>(Note 4) | (208)               |
| 230           | Key not in<br>table DSN 2 | Data mgmt      | User program<br>(Note 4) | (209)               |
| 231           | Key not in<br>table DSN 3 | Data mgmt      | User program<br>(Note 4) | (210)               |
| 232           | Key not in<br>table DSN 4 | Data mgmt      | User program<br>(Note 4) | (211)               |
| 240           | Continued<br>checkpoint   | CKPT completed | User or new CKPT         | None                |

**Notes:**

1. The IFI instruction converts to:  
IF BR7 IS/NOT 0;  
the other instructions generate a message.
2. The SON instruction converts to:  
BUZZ;  
the other instructions generate a message.
3. The IFI instruction converts to:  
IFDSI 2, n  
where n is a number from 1 through 214;  
the other instructions generate a message.
4. The 5280 makes no distinction between  
key not found in key table and key  
not found in data set.

## Appendix A. Mnemonic to Operation Code Conversion Chart and Instructions Format

The assembler mnemonics are listed in alphabetical order. Arithmetic operators are listed with the associated register; for example, binary add is listed as BRn+= and decimal double precision divide as Rn(4)/. The instruction format and operation code that is generated by the assembler are indicated for each mnemonic. The operation code is always stored in the first byte of the 4-byte object code instruction. The contents of the other 3 bytes will be described in the *Functions Reference Manual*.

| Mnemonic        | Op Code | Source Format                                  |
|-----------------|---------|--|
| ALLOC           | 34      | ALLOC (dsn,,BRn)                               |
| AND             | 42      | IFHI/IFLO BRn AND constant IS<br>constant SKIP |
| BINDEC          | A6      | BINDEC (Rn,label)                              |
| BINHEX          | 49      | BINHEX (label[(L)],BRn)                        |
| BRa = BRb       | 98      | BRa = BRb[(L)]/label                           |
| BRa <=> BRb     | 45      | BRa <=> BRb/label                              |
| BRn = nn        | 99      | BRn = nn                                       |
| BRn (4) -=      | 96      | BRa(4) -= BRb[(L)]/label[(L)]                  |
| BRn (4) += nn   | 95      | BRn(4) += nn                                   |
| BRn (4) -= nn   | 97      | BRn(4) -= nn                                   |
| BRn (4) +=      | 94      | BRa(4) += BRb[(L)]/label[(L)]                  |
| BRn [(4)] /=    | AB      | BRn[(4)] /= label/BRb                          |
| BRn &=          | 9A      | BRa &= BRb[(L)]/label[(L)]                     |
| BRa &= d(1,BRb) | BA      | BRa &= [d] [(L),BRb)                           |
| BRn &= nn       | 9B      | BRn &= nn                                      |
| BRn V=          | 9C      | BRa V= BRb[(L)]/label[(L)]                     |
| BR V= d(1,BRb)  | BC      | BRa V= [d] [(L),BRb)                           |
| BRn V= nn       | 9D      | BRn V= nn                                      |
| BRn X=          | 9E      | BRa X= BRb[(L)]/label[(L)]                     |
| BRa X= d(1,BRb) | BE      | BRa X= [d] [(L),BRb)                           |
| BRn X= nn       | 9F      | BRn X= nn                                      |
| BRn +=          | 90      | BRa += BRb[(L)]/label[(L)]                     |
| BRn -=          | 92      | BRa -= BRb[(L)]/label[(L)]                     |
| BRn[(L)] *=     | AA      | BRa [(4)] *= BRb                               |
| BRa += d(1,BRb) | B0      | BRa += [d] [(L),BRb)                           |
| BRa -= d(1,BRb) | B2      | BRa -= [d] [(L),BRb)                           |
| BRn=(indexed)   | B8      | BRa = [d] [(L),BRb)                            |
| BRn -= nn       | 93      | BRn -= nn                                      |
| BRn += nn       | 91      | BRn += nn                                      |
| BRn(4) +=       | B4      | BRa(4) += [d] [(L),BRb)                        |
| BRn(4) -=       | B6      | BRa(4) -= [d] [(L),BRb)                        |
| BRn = Rn        | A7      | BRn = Rn                                       |
| BUZZ            | C7      | BUZZ   |
| CALL            | 0B      | CALL [BRn,] label/d                            |

| Mnemonic            | Op Code | Source Format                            |
|---------------------|---------|--|
| CALLTB              | 0B      | CALLTB BRn , label                       |
| CLC                 | AE      | CLC (BRa,BRb,L)                          |
| CLICK               | C7      | CLICK                                    |
| CLOZ                | 23      | CLOZ (dsn)[,N/R/E/D,W/P,C/V/*,C/L/*,BRn) |
| CNENTR              | C7      | CNENTR                                   |
| CRTMM               | CA      | CRTMM(BRa,BRb,BRC [,NC/S])               |
| d(1,BRa) = BRb      | A3      | [d] ([L],BRa) = BRb                      |
| d(1,BRn) = constant | B9      | [d] ([L],BRn) = constant                 |
| d(1,BRn) = Rn       | 7L      | [d] ([L],BRn) = Rn                       |
| d,Rn = constant     | 44      | [d],Rn = constant                        |
| DECBIN              | A7      | DECBIN (label,Rn)                        |
| DECR                | 06      | DECR BRn GOTO label                      |
| DISPEX              | C7      | DISPEX                                   |
| DISPST              | C7      | DISPST                                   |
| DUP                 | BD      | DUP ([d],BRn,L)                          |
| ENABLE              | 0C      | ENABLE ([label],[POP])                   |
| ENTR                | CF      | ENTR (sfmt[,BRn,O/N])                    |
| EXIT                | 2F      | EXIT                                     |
| GOTAB               | 08      | GOTAB BRn,label                          |
| GOTO                | 00      | GOTO label                               |
| GOTO BRn (indexed)  | 08      | [BRn,] [label]                           |
| GSKC                | 48      | GSKC (Rn)                                |
| HEXBIN              | 4A      | HEXBIN (BRn,label[(L)])                  |
| IF BRn EQ           | 6E      | IF BRa EQ BRb GOTO label                 |
| IF BRn GE/LE        | 6F      | IF BRa GE/LE BRb GOTO label              |
| IF BRn GT/LT        | 6D      | IF BRa GT/LT BRb GOTO label              |
| IF BRn NE           | 6C      | IF BRa NE BRb GOTO label                 |
| IF BRn 0            | 03      | IF BRn IS/NOT 0 GOTO label               |
| IF fmt              | 02      | IF fmt IS/NOT FMT GOTO label             |
| IF Rn AN            | 0D      | IF Rn IS/NOT AN GOTO label               |
| IF Rn CK            | 0E      | IF Rn IS/NOT CK GOTO label               |
| IF Rn EQ            | 62      | IF Ra EQ Rb GOTO label                   |
| IF Rn GE/LE         | 63      | IF Ra GE/LE Rb GOTO label                |
| IF Rn GT/LT         | 61      | IF Ra GT/LT Rb GOTO label                |
| IF Rn NE            | 60      | IF Ra NE Rb GOTO label                   |
| IF Rn SN            | 0F      | IF Rn IS/NOT SN GOTO label               |
| IF Rn 0             | 01      | IF Rn IS/NOT 0 GOTO label                |
| IF Rn -             | 05      | IF Rn IS/NOT - GOTO label                |
| IFB IS              | BB      | IFB [d] (BRn) IS constant SKIP           |
| IFB OFF             | B5      | IFB [d] (BRn) OFF constant SKIP          |
| IFB ON              | Bf      | IFB [d] (BRn) ON constant SKIP           |
| IFC IS              | 4E      | IFC [d],Rn/label IS constant SKIP        |
| IFC NOT             | 4C      | IFC [d],Rn/label NOT constant SKIP       |
| IFD Rn EQ           | 66      | IFD Ra EQ Rb GOTO label                  |
| IFD Rn GE/LE        | 67      | IFD Ra GE/LE Rb GOTO label               |
| IFD Rn GT/LE        | 65      | IFD Ra GT/LT Rb GOTO label               |
| IFD Rn NE           | 64      | IFD Ra NE Rb GOTO label                  |
| IFDSI               | 25      | IFDSI n,dsn IS/NOT ON GOTO label         |
| IFI                 | 07      | IFI In IS/NOT ON GOTO label              |

| Mnemonic         | Op Code | Source Format                        |
|------------------|---------|--------------------------------------|
| IFH BRn EQ       | 6A      | IFH BRa EQ nn GOTO label             |
| IFH BRn GE/LE    | 6B      | IFH BRn GE/LE nn GOTO label          |
| IFH BRn GT/LT    | 69      | IFH BRn GT/LT nn GOTO label          |
| IFH BRn NE       | 68      | IFH BRn NE nn GOTO label             |
| IFIR             | 04      | IFIR In IS/NOT ON GOTO label         |
| INXEQ            | A5      | INXEQ (BRn[(4)] label [,n])          |
| INIT             | 33      | INIT (dsn, BRn)                      |
| INSBLK           | 32      | INSBLK (dsn,,BRn [,O/N])             |
| KACCPT           | C7      | KACCPT                               |
| KATTCH           | C4      | KATTCH                               |
| KDETC            | C5      | KDETC                                |
| KERRCL           | C7      | KERRCL (BRn)                         |
| KERRST           | C7      | KERRST (BRa, BRb)                    |
| KEYOP            | C7      | KEYOP (nn[,BRa, BRb])                |
| label = BRn      | A2      | label[(L)] = BRn                     |
| label = constant | 44      | label = constant                     |
| label = Rn       | 8L      | label[(L)] = Rn                      |
| label SL n       | A1      | label [(L)] SL n                     |
| LOAD             | 2E      | LOAD (label [,P,A,E])                |
| MMCRT            | CB      | MMCRT (BRa, BRb, BRc)                |
| MOFF             | 1A      | MOFF (Ra, Rb[, d, L])                |
| MVC              | AC      | MVC (BRa, BRb, L)                    |
| MVCR             | AC      | MVCR (BRa, BRb, L)                   |
| MVCV             | AC      | MVCV (BRa, BRb, L)                   |
| MVER             | 19      | MVER (Ra, Rb [,d,L])                 |
| NOP              | 00      | NOP                                  |
| OPEN             | 22      | OPEN (dsn [,BRn])                    |
| POSN             | 26      | POSN (dsn, BOE/CURR/LAST/EOD [,O/N]) |
| READ             | 20      | READ (dsn [,fmt, BRn/Rn/-/0/+,O/N])  |
| READMG           | C7      | READMG (BRa, BRb)                    |
| REBF             | 21      | REBF (BRn, fmt)                      |
| REPFLD           | C3      | REPFLD                               |
| RESCAL           | CD      | RESCAL (BRn, label)                  |
| RESMXT           | CD      | RESMXT [(BRn)]                       |
| RESUME           | CD      | RESUME [(B)]                         |
| RETEXT           | 0C      | RETEXT [(BRn)]                       |
| RETURN           | 0C      | RETURN [(BRn)]                       |
| RL               | A1      | BRn [(L)] RL n                       |
| Rn =             | 14      | Rn = Rn/n                            |
| Rn <=>           | 13      | Ra <=> Rb                            |
| Rn -             | 11      | Ra = Rb - n/Rc                       |
| Rn +             | 10      | Ra = Rb + n/Rc                       |
| Rn *             | 18      | Ra = Rb * n/Rc                       |
| Rn /             | 17      | Ra = Rb / n/Rc                       |
| Rn(32) *         | 15      | Ra(32) = Rb * n/Rc                   |
| Rn(32) /         | 12      | Ra = Rb(32) / Rc                     |
| Rn = BRn         | A6      | Rn = BRn                             |
| Rn = d(1, BRn)   | 7L      | Rn = [d] [(L), BRn]                  |
| Rn = label       | 8L      | Rn = label [(L)]                     |

| Mnemonic            | Op Code | Source Format                    |
|---------------------|---------|----------------------------------|
| Rn = nn             | 46      | Rn = +nn                         |
| Rn = -nn            | 47      | Rn = -nn                         |
| RR                  | A1      | BRn [(L)] RRn                    |
| RSTMG               | C7      | RSTMG                            |
| RTIMER              | C7      | RTIMER (BRn)                     |
| RXORW               | 43      | RXORW (constant, BRn, constant)  |
| SEARCH              | 24      | SEARCH (dsn, BRn, B/F/R/L)       |
| SETOFF              | B3      | SETOFF ([d], BRn,nn)             |
| SETON               | B1      | SETON ([d], BRn,nn)              |
| SKIP WHILE          | A0      | SKIP WHILE BRa LE BRb [STEP nn]  |
| SL (binary)         | A1      | BRn [(L)] SL n                   |
| SL (decimal)        | 1C      | Ra = Rb SL n                     |
| SLS                 | 1D      | Ra = Rb SLS n                    |
| SOFF                | 41      | SOFF (la [, lb, lc])             |
| SON                 | 40      | SON (la [, lb, lc])              |
| SR (binary)         | A1      | BRn [(L)] SR n                   |
| SR (blank register) | 16      | Rn = 1 SR 1                      |
| SR (decimal)        | 16      | Ra = Rb SR n/Rc                  |
| SRAT                | 2B      | SRAT (dsn, BRn)                  |
| SRR                 | 1F      | Ra = Rb SRR n                    |
| SRS                 | 1E      | Ra - Rb SRS n                    |
| SYSLCK              | 2C      | SYSLCK                           |
| SYSUNL              | 2D      | SYSUNL [(*)]                     |
| TBBS                | 55      | BRn = TBBS (label, Rn)           |
| TBDL                | 57      | TBDL (label, BRn)                |
| TBFH                | 50      | BRn = TBFH (label, Rn [,N])      |
| TBFL                | 54      | BRn = TBFL (label, Rn [,N])      |
| TBFX                | 53      | BRn = TBFX (label, Rn [,N])      |
| TBIN                | 56      | TBIN (label, BRn) = Rn           |
| TBRD                | 52      | Rn = TBRD (label, BRn)           |
| TBRL                | 52      | Rn = TBRL (label, BRn)           |
| TBWE                | 51      | TBWE (label, BRn) = Rn           |
| TBWT                | 51      | TBWT (label, BRn) = Rn           |
| TCLOZ               | 3F      | TCLOZ (dsn)                      |
| TCTL                | 3F      | TCTL (dsn, X'II' [,O/N, D])      |
| TINIT               | 22      | TINIT (dsn)                      |
| TLCK                | 58      | TLCK (label)                     |
| TOPEN               | 22      | TOPEN (dsn)                      |
| TRANS               | A8      | TRANS (BRa, BRb, L)              |
| TREAD               | 2A      | TREAD (dsn [,fmt,-,O/N,-])       |
| TRT                 | A9      | TRT (BRa, BRb, L [, R])          |
| TTERM               | 23      | TTERM (dsn)                      |
| TUNLCK              | 59      | TUNLCK (label)                   |
| TWAIT               | 36      | TWAIT (dsn)                      |
| TWRT                | 3A      | TWRT (dsn [,fmt,F,O/N,B])        |
| WAIT                | 36      | WAIT [(dsn)]                     |
| WFMCRT              | 3E      | WFMCRT (BRa [,fmt,BRb,B/ADD])    |
| WRBF                | 3C      | WRBF (BRa [,fmt,BRb])            |
| WRT                 | 30      | WRT (dsn [,fmt,BRn/-/0/+,O/N,B]) |
| WRTI                | 31      | WRTI (dsn [,fmt,,O/N,B])         |
| WRTS                | 35      | WRTS (dsn [,fmt,,O/N,B])         |
| ZONE                | 1B      | ZONE (Ra,Rb/nn [,d,L])           |

**Key**

|             |   |
|-------------|---|
| <b>d</b>    | is a displacement.  |
| <b>dsn</b>  | is the data set number.   |
| <b>fmt</b>  | is the label of the edit format.  |
| <b>L</b>    | is a length specification (except for the SEARCH instruction, which uses L as a keyword).                       |
| <b>n</b>    | is a single digit decimal value (0-9).  |
| <b>nn</b>   | is a numeric value.   |
| <b>sfmt</b> | is the label of the screen format.  |
| <b>/</b>    | separates two or more parameters when only one may be specified. Do not include this symbol in the instruction. |
| <b>[]</b>   | Enclose optional parameters. Do not include these symbols in the instruction.                                   |

**This page is intentionally left blank**

## Appendix B. SCS Control Characters

Printer formatting can be accomplished via an assembler, COBOL, or DE/RPG program or via communications. To use the SCS characters supported by communications, see the description under the desired communications access method: SNA, BSC, or MRJE. To use a program to format your printed output, code the following SCS characters that your printer supports. Code the control characters in the printer output data stream intended for the printer. The format of the data stream is:

CC Data CC Data CC Data

where CC is the control characters.

The following chart describes the general functions provided by the printer control characters. A detailed description of each control character follows the chart.

| SCS Control Character | Hex Code          | Function                      | Valid with |      |      |      |
|-----------------------|-------------------|-------------------------------|------------|------|------|------|
|                       |                   |                               | 5222       | 5224 | 5225 | 5256 |
| BEL                   | 2F                | Alarm; sound alarm on printer | ✓          | ✓    | ✓    | ✓    |
| CR                    | 0D                | Carrier return                | ✓          | ✓    | ✓    | ✓    |
| FF                    | 0C                | Forms feed                    | ✓          | ✓    | ✓    | ✓    |
| Fmt                   | 2B . . .          | Format                        |            |      |      |      |
|                       | 2BC1nnhh          | Horizontal (SHF)              | ✓          | ✓    | ✓    | ✓    |
|                       | 2BC2nnvv          | Vertical (SVF)                | ✓          | ✓    | ✓    | ✓    |
|                       | 2BC6nnld          | Line density (SLD)            | ✓          | ✓    | ✓    |      |
|                       | 2BC8nngguu        | Graphic error (SGEA)          | ✓          | ✓    | ✓    | ✓    |
|                       | 2BD10381P1        | Set CGCS (SCL)                |            | ✓    | ✓    |      |
|                       | 2BD2042900vv      | Char distance (SCD)           | ✓          | ✓    | ✓    |      |
| 2BFEnmmP1Pn           | Alt char (LAC)    |                               | ✓          | ✓    |      |      |
| IRS                   | 1E                | Interchange record separator  | ✓          | ✓    | ✓    | ✓    |
| LF                    | 25                | Line feed                     | ✓          | ✓    | ✓    | ✓    |
| NL                    | 15                | New line                      | ✓          | ✓    | ✓    | ✓    |
| NUL                   | 00                | No operation                  | ✓          | ✓    | ✓    | ✓    |
| PP                    | 34 . . .          | Print position                |            |      |      |      |
|                       | 34C0nn            | Horizontal absolute           | ✓          | ✓    | ✓    | ✓    |
|                       | 34C8nn            | Horizontal relative           | ✓          | ✓    | ✓    | ✓    |
|                       | 34C4nn            | Vertical absolute             | ✓          | ✓    | ✓    | ✓    |
| 344Cnn                | Vertical relative | ✓                             | ✓          | ✓    | ✓    |      |
| TRN                   | 35nn              | Transparent                   |            | ✓    | ✓    |      |

- **BEL**

Function: This control character stops printing, sounds the audible alarm, if installed, and turns on the Attention indicator.

Code: X'2F'

Results: When the printer detects this control character, it:

1. Allows all preceding data to be printed and all preceding control characters to be executed
2. Turn the Ready indicator off
3. Turns the Attention indicator on
4. Sounds the audible alarm, if installed
5. Stops printing
6. Stops formatting
7. Returns an unavailable status to the controller

- **CR (Carrier Return)**

Function: This control character performs a carrier return to the first print position on the same line.

Code: X'0D'

Results: The horizontal print position logically moves to the first print position on the same line. If it already is at the first print position, no operation occurs. No physical motion will occur as a result of this control character.

- **FF (Forms Feed)**

Function: This control character moves the paper to the next logical page as specified by the Set Vertical Format control character (see *Fmt*) in this topic.

Default: 1 logical page = 1 logical line.

Code: X'0C'

Results: The print position moves to the first logical print line and first logical print position of the next logical page. Physical motion of the paper will occur as a result of this control character.

- **Fmt (Format)**

Function: This control character defines forms movement, line length, and other printer functions as per the following description:

Code: X'2B'

Format:

| Code | Function Parameter | Length | Associated Parameters |
|------|--------------------|--------|-----------------------|
| 2B   | (HEX)              | (HEX)  | (HEX)                 |

**Note:** The following table shows the various function types and their associated parameters. nn is the number of bytes to follow the function type and includes the count frame in its value.

#### Function Types Available for Use with the Format (Fmt) Printer Control Character

| Function Type               | Format | Values of Parameters  | Description   |
|-----------------------------|--------|---|---|
| SHF (set horizontal format) | C1nnhh | nn = number of bytes in the SHF string<br><br>hh = maximum horizontal print position (greater than or equal to 1 and less than or equal to 198). The default is 132 (X'84') 10 cpi or 198 (X'C6') 15 cpi.   | valid entries are 01 and 02.<br><br>Sets the maximum print position (MPP), which is the value of the print line length. MPP is 132 for 10 cpi and 198 for 15 cpi. |
| SVF (set vertical format)   | C2nnvv | nn = number of bytes in the SVF string.<br><br>vv = maximum number of lines on a page greater than or equal to 1 and less than or equal to 255). The default is a page length of one line.  | Valid entries are 01 and 02.<br><br>Sets the maximum print line (MPL) on the logical page.  |
| SLD (set line density)      | C6nnld | nn = number of bytes in the SLD string. Valid entries are 01 and 02.<br><br>ld = density parameters. Line density is defined in increments of .353 mm (1/72 inch). Normal values are:<br>0C = 6 lpi (12/72 inch).<br>09 = 8 lpi (9/72 inch).<br>08 = 9 lpi (8/72 inch). | Allows selection of vertical line density of either 6, 8, or 9 lines per inch. Default is 6 lpi, 9 lpi not valid for IBM 5222.                                    |

| Function Type                   | Format          | Values of Parameters   | Description   |
|---------------------------------|-----------------|--|---|
| SGEA (set graphic error action) | C8nnnguu        | <p>nn = number of bytes in the SGEA string.</p> <p>gg = substitute character option. Default is a hyphen (X'60').</p> <p>uu = Error and status options when an unprintable character is encountered.</p> <p>01=No stop, no status.<br/>02=Defaults to 01.<br/>03=Stop, hard error status.<br/>04=Defaults to 03.<br/>The default for uu is 01.</p> | <p>nn must be at least 1 and not greater than 3 for SGEA.</p> <p>This substitute character will be printed in place of any unprintable characters in the data stream.</p>   |
| SCL (set CGCS through local ID) | D1nn81P1        | <p>nn = number of bytes in the SLC string. Must be 03.</p> <p>81 = type.</p> <p>P1 = local ID of character set to be loaded.</p>   | <p>Load a predetermined graphic character set to be used for the next print job. Default is FF.</p>   |
| SCD (set character distance)    | D2nn29<br>00vv  | <p>nn = number of bytes in the SCD string. Must be 04.</p> <p>29 = type.</p> <p>00vv = character per inch. Valid vv values are:<br/>0A=10 cpi.<br/>0F=15 cpi.</p>  | <p>Allows selection of character density of either 10 or 15 characters per inch.</p>  |
| LAC (load alternate characters) | FEnn01<br>P1-Pn | <p>nn = number of bytes to be loaded.</p> <p>P1-Pn define the character images.</p>  | <p>nn = (NX10) +2 where N is the number of alternate characters being loaded. Maximum value of nn = 252 for each command.</p> <p>Allows customer designed fonts or characters to be loaded from host for subsequent printing. Each character image requires 10 bytes.</p> |

For load alternate character, see the reference manual for the particular printer you are using.

The following table shows the characteristics of the SHF and SVF function types.

#### Valid Values for the SHF and SVF Set Types

| Set Type Code   | Parameters        | Results (MPL and MPP)   | Error                 |
|-----------------|-------------------|---|-----------------------|
| SHF<br>2BC1nnhh | nn=00             |   | Invalid SCS parameter |
|                 | nn=01             | MPP=132 for 10 cpi<br>MPP=198 for 15 cpi  | None                  |
|                 | nn=02<br>hh=00    | MPP=132 for 10 cpi<br>MPP=198 for 15 cpi  | None                  |
|                 | nn=02<br>hh=01-C6 | MPP=hh<br>Maximum line length for<br>15 cpi is 198 characters;<br>for 10 cpi it is 132<br>characters. | None                  |
|                 | nn=02<br>hh=C7-FF |   | Invalid SCS parameter |
|                 | nn=03-FF          |   | Invalid SCS parameter |
| SVF<br>2BC2nnvv | nn=00             |   | Invalid SCS parameter |
|                 | nn=01             | MPL=1   | None                  |
|                 | nn=02<br>vv=00    | MPL=1   | None                  |
|                 | nn=02<br>vv=01-FF | MPL=1-255 as specified  | None                  |
|                 | nn=03-FF          |   | Invalid SCS parameter |

- *IRS (Interchange Record Separator)*

Function: This control character does the same thing that NL does.

Code: X'1E'

Results: The print position moves to the first print position of the next line. If the current position is on the last line of the page, the new position is the first print position of the first line of the next page.

- *LF (Line Feed)*

Function: This control character moves the paper one line without altering the print position.

Code: X'25'

Results: The paper logically moves to the same print position on the following line. If you use this control character on the last line of a page, it will move the print position to the first line of the next page.

- *NL (New Line)*

Function: This control character moves the paper to the next line.

Code: X'15'

Results: The print position moves to the first print position on the next line if it is not coded on the last line of the page. If you code this on the last line, it moves the paper to the first print position on the first line of the next page.

- *NUL*

Function: No-op

Code: X'00'

Results: No characters are printed and no functions are performed.

- *PP (Print Position)*

Function: This control character moves the logical print position as determined by the associated parameters.

Code: X'34'

|         |                           |                        |
|---------|---------------------------|------------------------|
| Format: | <b>Function Parameter</b> | <b>Value Parameter</b> |
|         | (HEX)                     | (HEX)                  |

**Note:** The following table shows the various function types and their associated parameters. nn is the value parameter for the function type that precedes it.

| Function Type                    | Format | Values of Parameters  | Description  |
|----------------------------------|--------|---|--|
| AH<br>(Absolute horizontal move) | C0nn   | nn=00 – No operation. PP is unchanged; no error.<br><br>01 ≤ nn ≤ MPP – The print position becomes the value of nn. | Hex value of the horizontal position (current print position, PP, less than or equal to the end of the line, MPP). |

| Function Type                    | Format | Values of Parameters  | Description  |
|----------------------------------|--------|---|--|
|                                  |        | $nn > MPP$ – Error; invalid SCS parameter.  |  |
| AV<br>(Absolute vertical move)   | C4nn   | <p><math>nn=00</math> – No operation. PL is unchanged; no error.</p> <p><math>PL \leq nn \leq MPL</math> – the PL becomes the value of nn and remains on the logical page.</p> <p><math>01 \leq nn \leq PL</math> – The PL becomes the value of nn and remains on the logical page.</p> <p><math>nn &gt; MPL</math> – Error; invalid SCS parameter.</p> | Hex value of the vertical position (current print line, PL, less than or equal to the end of the page, MPL).                             |
| RH<br>(Relative horizontal move) | C8nn   | <p><math>nn=00</math> – No operation. PP is unchanged; no error</p> <p><math>PP+nn \leq MPP</math> – The print position becomes the value of PP+nn.</p> <p><math>PP+nn &gt; MPP</math> or <math>nn &gt; PP</math> – Error; invalid SCS parameter.</p>   | Hex value of the movement from present horizontal position (current print position, PP, less than or equal to the end of the line, MPP). |
| RV<br>(Relative vertical move)   | 4Cnn   | <p><math>nn=00</math> – No operation; PL is unchanged; no error.</p> <p><math>PL+nn \leq MPL</math> – The PL becomes the value of PL+nn.</p> <p><math>PL+nn &gt; MPL</math> or <math>nn &gt; MPL</math> – Error; invalid SCS parameters.</p>  | Hex value of the movement from present vertical position (current print line, PL, less than or equal to the end of the page, MPL).       |

**This page is intentionally left blank**

The description of *The Self-Check Algorithm* under *Self-Check* in Chapter 2 outlines general steps for an algorithm. The IBM 5280 self-check function provides variations to the general steps with an input translate table, a product table, and an output translate table. Variations in the way the input is translated, the way the products are added, and the way the output is translated are also available. With these variations, the self-check algorithm you describe with the .SELFCHK control statement may be summarized with the following steps:

1. Determine input value for each foundation position.
2. Obtain products and sum the products.
3. Convert the resulting sum.
4. Determine output for the self-check number or numbers.

These steps, and the parameters of the .SELFCHK control statement that effect each step, are illustrated in Figure C-1. The .SELFCHK parameters that describe the self-check register (FLDLEN, DISP) are not included in the figure, nor are the parameters that specify the modulus (MOD) or the label of the self-check control area (LABEL). The fields in the figure (Field 1 through Field 6) represent the positional fields of the CNTL parameter.

A detailed description of each step, and how it is effected by the .SELFCHK parameters, follows the figure.

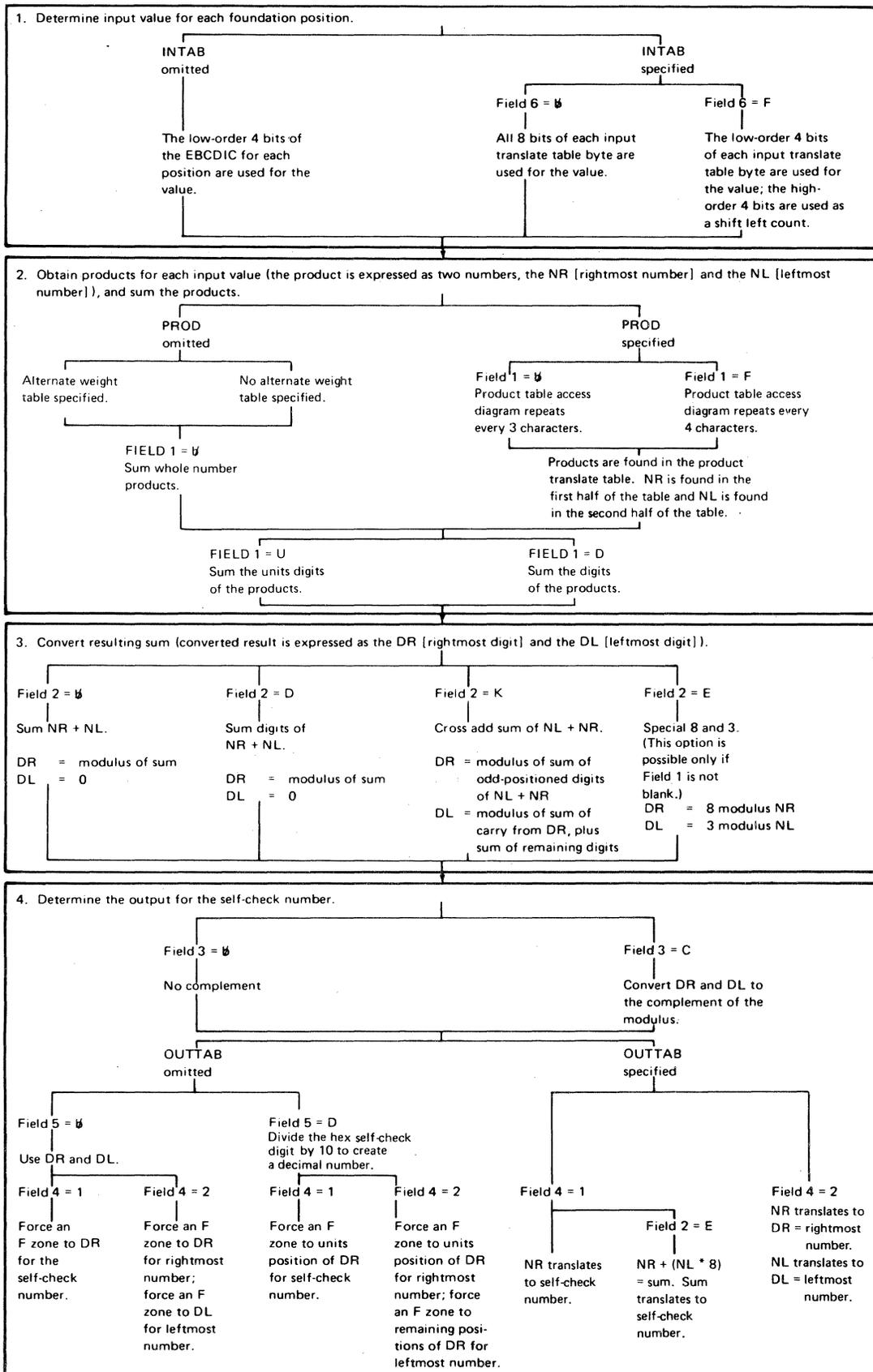


Figure C-1. Self-Check Computations According to .SELCHK Parameters

1. Determine input value for each foundation position.

INTAB  
omitted

|  
The low-order 4 bits of the  
EBCDIC for each position  
are used for the value.

If no input translate table is specified, the low order 4 bits of the EBCDIC for each foundation position are used for the input value. For example, if the foundation is:

A478B770

the EBCDIC for the foundation position is:

High order bits: CFFFCFFF  
Low order bits: 14782770

The input value for the foundation positions is: 14782770.

INTAB  
specified

If an input translate table is specified, the value must be determined from the table.

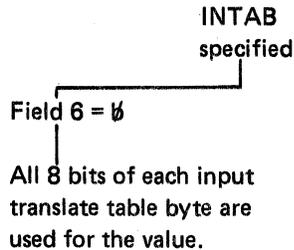
An input translate table can provide a different numeric value for each character. You must set up and initialize the input translate table with .DC control statements. The table must be 256 bytes in length. Initialize the table so that the numeric value for each character is stored into the table at the displacement that corresponds to the hex value of the character. For example, the hex value of the character A is C1; the numeric value for the character A must be stored at displacement X'C1' into the input translate table.

Figure C-2 shows, on the left, the displacement for each of the IBM 5280 keyboard characters. The right side of Figure C-2 shows a sample input translate table, with numeric values entered for each keyboard character.

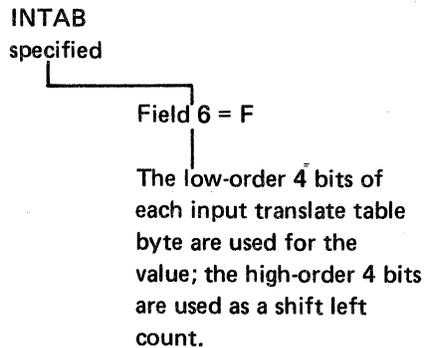
| Displacements |     |   |   |   |   |   |   |   |   |   |   |   |   |   |   | Sample Table |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
|---------------|-----|---|---|---|---|---|---|---|---|---|---|---|---|---|---|--------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
|               | 0   | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | A | B | C | D | E | F            |    | 0  | 1  | 2  | 3  | 4  | 5  | 6  | 7  | 8  | 9  | A  | B  | C  | D  | E  | F  |
| 0             |     |   |   |   |   |   |   |   |   |   |   |   |   |   |   |              | 0  |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
| 1             |     |   |   |   |   |   |   |   |   |   |   |   |   |   |   |              | 1  |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
| 2             |     |   |   |   |   |   |   |   |   |   |   |   |   |   |   |              | 2  |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
| 3             |     |   |   |   |   |   |   |   |   |   |   |   |   |   |   |              | 3  |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
| 4             | ␣   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |              | 4  | 11 |    |    |    |    |    |    |    |    | 12 | 13 | 14 | 15 | 16 | 17 |    |
| 5             | &   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |              | 5  | 18 |    |    |    |    |    |    |    |    | 19 | 20 | 21 | 22 | 23 | 24 |    |
| 6             | - / |   |   |   |   |   |   |   |   |   |   |   |   |   |   |              | 6  | 25 | 26 |    |    |    |    |    |    |    | 27 | 28 | 29 | 30 | 31 | 32 |    |
| 7             |     |   |   |   |   |   |   |   |   |   |   |   |   |   |   |              | 7  |    |    |    |    |    |    |    |    |    | 33 | 34 | 35 | 36 | 37 | 38 | 39 |
| 8             | a   | b | c | d | e | f | g | h | i |   |   |   |   |   |   | 8            | 40 | 41 | 42 | 43 | 44 | 45 | 46 | 47 | 48 |    |    |    |    |    |    |    |    |
| 9             | j   | k | l | m | n | o | p | q | r |   |   |   |   |   |   | 9            | 49 | 50 | 51 | 52 | 53 | 54 | 55 | 56 | 57 |    |    |    |    |    |    |    |    |
| A             | ~   | s | t | u | v | w | x | y | z |   |   |   |   |   |   | A            | 58 | 59 | 60 | 61 | 62 | 63 | 64 | 65 | 66 |    |    |    |    |    |    |    |    |
| B             |     |   |   |   |   |   |   |   |   |   |   |   |   |   |   |              | B  |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
| C             | }   | A | B | C | D | E | F | G | H | I |   |   |   |   |   | C            | 67 | 70 | 71 | 72 | 73 | 74 | 75 | 76 | 77 | 78 |    |    |    |    |    |    |    |
| D             | }   | J | K | L | M | N | O | P | Q | R |   |   |   |   |   | D            | 68 | 79 | 80 | 81 | 82 | 83 | 84 | 85 | 86 | 87 |    |    |    |    |    |    |    |
| E             | \   | S | T | U | V | W | X | Y | Z |   |   |   |   |   |   | E            | 69 | 89 | 90 | 91 | 92 | 93 | 94 | 95 | 96 |    |    |    |    |    |    |    |    |
| F             | 0   | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |   |   |   |   |   | F            | 00 | 01 | 02 | 03 | 04 | 05 | 06 | 07 | 08 | 09 |    |    |    |    |    |    |    |

Figure C-2. Input Translate Table Displacements and Sample Table

Many bytes in the input translate table are not initialized, such as the bytes from displacement hex 00 to hex 3F. The assembler fills any uninitialized bytes with hex 00.

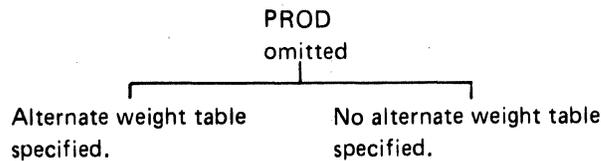


If an input translate table is specified, and if Field 6 of the CNTL parameter is blank, all 8 bits of the input translate table bytes are used for the input value. For example, the letter M is at displacement hex D4; the sample table has the value 82 at displacement hex D4. Therefore, the input value for the letter M is 82.



If an input translate table is specified and field 6 of the CNTL parameter specifies F, the low order 4 bits of each input translate table byte are used for the input value, and the high four order bits are used as a shift left count. The input value (low order 4 bits) will be located the specified number of register positions (high order 4 bits) to the left of the current position. All bypassed positions will be filled with X'00'. In the preceding example, the input translate table byte contained 82. If the current position is position zero (the rightmost in the register), the input value 2 will be placed in register position 8. The position pointer will now be pointing at position 9 for the next input translate table byte.

2. Obtain products for each input value, and sum the products.



If no product table is specified, products are obtained by multiplying the input value by the corresponding weight. The weights are specified with the WGTS parameter. Enter a hex 00 in each position that corresponds to a self-check digit and to a position that is bypassed. Enter a 1-byte weight for each position that corresponds to a foundation position. The weight must be lower in value than the modulus.

For a multiply type algorithm, random weights are used. The following example shows how to assign weights for a self-check register assuming a modulus of 9, a field length (FLDLEN) of 21, and two self-check digit positions at the right of the register.

WGTS=X'000000000000000000000000807060504030102030405060708070605040000';

Bypass  
Positions

Foundation Number  
Positions

Self-Check  
Digit positions

The input value for the rightmost foundation number position is multiplied by hex 04, and so on.

Divide type algorithms require specific calculated weights. An algorithm that involves divide operations can be expressed as follows:

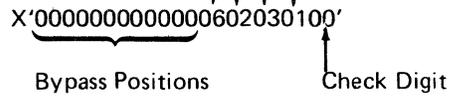
Divide the foundation by some number and the remainder is the check digit(s).

These divide type algorithms can be defined with the .SELFCHK control statement by determining the correct weights for the equivalent divide operation and using the divisor for the modulus. The weights are determined by dividing the desired divisor into 100000000000000 with the remainder of each step the weight for the corresponding position of the foundation, starting from the rightmost position.

For example:

|               |            |                  |
|---------------|------------|------------------|
| 0142 ... etc. | 7          | 1000000000000000 |
|               | 0          |                  |
|               | 10         |                  |
|               | 7          |                  |
|               | 30         |                  |
|               | 28         |                  |
|               | 20         |                  |
|               | 14         |                  |
|               | 6 ... etc. |                  |

Weighting factors for the above example, with the self-check digit in the rightmost position and the foundation number four positions in length, are:



When using a divide type algorithm, Field 1 and Field 2 of the CNTL parameter of the .SELFCHK control statement must be blank.

When weights are determined, they are multiplied by the input value of each foundation position, and the result is the product. The products are then added, and the IBM 5280 establishes two results that are stored internally. The two results are referred to as the NR (rightmost number) and the NL (leftmost number). These two results provide flexibility for numerous algorithms. When only one result is appropriate NL is forced to 0.

There are three options for adding the calculated products. Field 1 of the CNTL parameter specifies how the products are added to produce the NR and NL.

As part of the process of obtaining the modulus, the IBM 5280 internally converts the products and sums of the products to the number base of the modulus. That is, if the modulus is 11, the products and sums of products is expressed in base 11. Usually, your algorithm is such that the effects of this conversion can be ignored. Only if your algorithm deviates from the common algorithm as noted in the examples do these effects have to be considered.

Field 1 = 0  
 |  
 Sum whole number products.

If Field 1 is blank, the input value is multiplied by the weight for each foundation position, and the whole number products are added. The sum is used for the NR, and the NL is forced to 0.

Example:

|                             |    |     |     |    |    |       |
|-----------------------------|----|-----|-----|----|----|-------|
| Numeric value of characters | 3  | 12  | 10  | 4  | 6  |       |
| Weights                     | 5  | 4   | 3   | 2  | 1  |       |
| Products                    | 15 | 48  | 30  | 8  | 6  |       |
| Summation                   | 15 | +48 | +30 | +8 | +6 | = 107 |
| NR equals                   |    |     |     |    |    | 107   |
| NL equals                   |    |     |     |    |    | 0     |

Field 1 = U

↓  
Sum the units digits  
of the products.

If Field 1 specifies U, the input value is multiplied by the weight for each input position, and the units positions of the products are added. This sum is used for the NR, and NL is forced to 0.

Example:

|                             |    |    |    |   |   |      |
|-----------------------------|----|----|----|---|---|------|
| Numeric value of characters | 3  | 12 | 10 | 4 | 6 |      |
| Weights                     | 5  | 4  | 3  | 2 | 1 |      |
| Products                    | 15 | 48 | 30 | 8 | 6 |      |
| Summation                   | ↓  | ↓  | ↓  | ↓ | ↓ |      |
|                             | 5  | +  | 8  | + | 0 | +    |
|                             |    |    | 8  | + | 6 |      |
|                             |    |    |    |   |   | = 27 |
| NR =                        |    |    |    |   |   | 27   |
| NL =                        |    |    |    |   |   | 0    |

Commonly, if this option is used, the modulus is 10. The following is an example of modulus 11 and considers the effects of expressing the product and sum in the number base of the modulus.

Example: Assume modulus 11

|                             |    |    |    |    |    |              |
|-----------------------------|----|----|----|----|----|--------------|
| Numeric value of characters | 3  | 12 | 10 | 4  | 6  |              |
| Weights                     | 5  | 4  | 3  | 2  | 1  |              |
| Products (base 10)          | 15 | 48 | 30 | 8  | 6  |              |
| Products (base 11)          | 14 | 44 | 28 | 8  | 6  |              |
| Summation (base 11)         | 4  | +4 | +8 | +8 | +6 | = 28 base 11 |
| NR = 28                     |    |    |    |    |    |              |
| NL = 0                      |    |    |    |    |    |              |

Field 1 = D  
 ↓  
 Sum the digits of  
 the products

If Field 1 is D, the input value is multiplied by the weight for each foundation position, and the units positions of the products are added. This sum is used for the NR. Also, the remaining positions of the products are added; this sum is used for the NL.

Example:

|                              |    |    |    |    |    |      |
|------------------------------|----|----|----|----|----|------|
| Numeric value for characters | 3  | 12 | 10 | 4  | 6  |      |
| Weights                      | 5  | 4  | 3  | 2  | 1  |      |
| Products                     | 15 | 48 | 30 | 8  | 6  |      |
| Summation                    | 5  | +8 | +0 | +8 | +6 | = 27 |
|                              | 1  | +4 | +3 |    |    | = 8  |
| NR = 27                      |    |    |    |    |    |      |
| NL = 8                       |    |    |    |    |    |      |

Usually the weights are such that the products are less than 100 so the sum of NR and NL is equivalent to taking the sum of each digit of the product. For the example above:

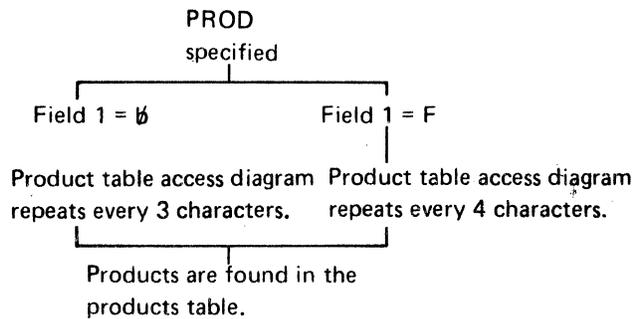
$$NR + NL = 27 + 8 = 35$$

Summing each digit of the products:

$$1+5+4+8+3+0+8+6 = 35$$

Commonly, if this option is used the modulus is 10. The following is an example if the modulus is 11 and considers the effects of expressing the product and sums in the base of the modulus.

|                              |       |    |    |    |    |              |
|------------------------------|-------|----|----|----|----|--------------|
| Numeric value for characters | 3     | 12 | 10 | 4  | 6  |              |
| Weights                      | 5     | 4  | 3  | 2  | 1  |              |
|                              | <hr/> |    |    |    |    |              |
| Products (base 10)           | 15    | 48 | 30 | 8  | 6  |              |
| Products (base 11)           | 14    | 44 | 28 | 8  | 6  |              |
|                              | ↓     | ↓  | ↓  | ↓  | ↓  |              |
| Summation (base 11)          | 4     | +4 | +8 | +8 | +6 | = 28 base 11 |
|                              | ↓     | ↓  | ↓  |    |    | = 7 base 11  |
|                              | 1     | +4 | +2 |    |    |              |



NR is found in the first half of the table.

NL is found in the second half of the table.

You can use a product table rather than weights to find the product for each foundation position. You must set up and initialize the 128-byte product table with .DC control statements. Do not use a .TABLE control statement to organize the product table. Specify the label of the product table for the PROD parameter of the .SELFCHK control statement. For the WGTS parameter, enter hex 00 in the self-check number position(s) and in any positions to be bypassed. Enter hex 01 in all foundation positions.

Although you do not organize the table with a .TABLE control statement, you can think of the product table as a data table consisting of eight 16-byte arguments. The first four arguments are used to find the NR, and the second four arguments are used to find the NL. If no NL is appropriate, initialize the second four arguments with hex 00s. If Field 1 of the CNTL parameter is blank, the product table access diagram repeats every 3 characters; if Field 1 is F the product access diagram repeats every 4 characters.

When the IBM 5280 uses the product table to find the product for a foundation character, the position of the character in the self-check register determines which argument of the product table the IBM 5280 uses.

When the appropriate argument is determined, the IBM 5280 uses the low-order 4 bits of the EBCDIC for the foundation character to determine which byte within the argument holds the product to use.

The following illustration shows which argument (line) of the product table is used for each position in a self-check register of up to 32 bytes. On the right are the register positions for a product table that repeats every 3 characters (Field 1 = ); on the left are the register positions for a product table that repeats every 4 characters (Field 1 = F).

| <u>Register Positions</u><br>(Repeat every 4 characters) | <u>Product Table Positions</u> | <u>Register Positions</u><br>(Repeat every 3 characters) |
|--|--------------------------------|--|
| 04 08 12 16 20 24 28 32                                  | 00 through 0F                  | 02 05 08 11 14 17 20 23 26 29 32                         |
| 03 07 11 15 19 23 27 31                                  | 10 through 1F                  | 01 04 07 10 13 16 19 22 25 28 31                         |
| 02 06 10 14 18 22 26 30                                  | 20 through 2F                  | 03 06 09 12 15 18 21 24 27 30                            |
| 01 05 09 13 17 21 25 29                                  | 30 through 3F                  | (not used)   |
| 04 08 12 16 20 24 28 32                                  | 40 through 4F                  | 02 05 08 11 14 17 20 23 26 29 32                         |
| 03 07 11 15 19 23 27 31                                  | 50 through 5F                  | 01 04 07 10 13 16 19 22 25 28 31                         |
| 02 06 10 14 18 22 26 30                                  | 60 through 6F                  | 03 06 09 12 15 18 21 24 27 30                            |
| 01 05 09 13 17 21 25 29                                  | 70 through 7F                  | (not used)   |

To find the product for the character in position 26 of the self-check register, the 5280 uses argument 3 (line 3) of the product table (positions hex 20 through 2F) if Field 1 = F; it uses argument 1 of the product table (positions 00 through 0F) if Field 1 is blank. The following example shows how the character in the 26th position of the self-check register is translated to the NR, assuming the product table repeats every 4 characters (Field 1 = F).

*Example:*

Register positions and contents

|   |   |   |   |   |   |   |   |   |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |   |   |  |  |  |  |  |  |  |
|---|---|---|---|---|---|---|---|---|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|---|---|--|--|--|--|--|--|--|
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 | 32 |   |   |  |  |  |  |  |  |  |
|   |   |   |   |   |   |   |   |   |    |    |    |    |    | 0  | 6  | /  | 0  | 9  | /  | 1  | 9  | 8  | 0  | *  | A  | R  | d  | c  | s  | #  | 4  | 7 | 8 |  |  |  |  |  |  |  |

Register Positions

|                         |
|-------------------------|
| 04 08 12 16 20 24 28 32 |
| 03 07 11 15 19 23 27 31 |
| 02 06 10 14 18 22 26 30 |
| 01 05 09 13 17 21 25 29 |
| 04 08 12 16 20 24 28 32 |
| 03 07 11 15 19 23 27 31 |
| 02 06 10 14 18 22 26 30 |
| 01 05 09 13 17 21 25 29 |

|   | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | A | B | C | D  | E | F |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|----|---|---|
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 0 | - | = | q | w | e  | r |   |
| t | y | u | i | o | p | a | s | d | f | g | h | j | k | l  | ; |   |
| ' | z | x | d | v | b | n | m | , | . | / | ! | @ | # | \$ | % |   |
| φ | & | * | ( | ) | _ | + | Q | W | E | R | T | Y | U | I  | O |   |
| P | A | S | D | F | G | H | J | K | L | : | " | Z | X | C  | V |   |
| B | N | M | , | . | ? | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9  | 0 |   |
| - | = | q | w | e | r | t | y | u | i | o | p | a | s | d  | f |   |
| g | h | j | k | l | ; | ' | z | x | c | v | b | n | m | ,  | . |   |

- 1** The register position determines which product table argument (line) to use.
- 2** For position 26, use the third product table argument (26 is found on the third line of the table).
- 3** The low order 4 bits of the EBCDIC representation of the character determines which byte in the argument to use. For 'c' the EBCDIC is hex 83; use byte 3.

The NR for the character is 'd', or EBCDIC hex 84. If an NL is used, it is found in the fourth byte of the seventh argument: NL = 'w', or EBCDIC hex A6.

### 3. Converting resulting sum.

The sum of the products (NR and NL) is converted to the base of a modulus, according to Field 2 of the CNTL parameter. The units positions of the result of this conversion is expressed as the DR (rightmost digit) and the DL (leftmost digit) to prevent confusing them with the NR and NL.

Field 2 =  $\emptyset$   
|  
Sum NR + NL.

DR = modulus of sum  
DL = 0

If Field 2 is blank, the NR and NL are added together, and the sum is converted to a base equal to the modulus specified in the MOD parameter. The units positions of the result is stored as the DR; only one result is appropriate so the DL is forced to 0.

Example:

The modulus is 10  
From the sum of products  
(Numbers in base 10)

|               |
|---------------|
| NR = 107      |
| NL = 0        |
| <hr/>         |
| NR + NL = 107 |

Modulus is taken on 107; the value of the units position is the result.

DR = 7  
DL = 0

Example:

The modulus is 11  
From the sum of products  
(Numbers in base 11)

|                      |
|----------------------|
| NR = 28 base 11      |
| NL = 7 base 11       |
| <hr/>                |
| NR + NL = 34 base 11 |

Modulus is taken on 34 base 11; the value of the units position is the result so:

DR = 4  
DL = 0

Field 2 = 0  
|  
Sum digits of  
NR + NL.

DR = modulus of sum  
DL = 0

If Field 1 = D, the NR and NL are added together, then each digit of the sum is added; this new sum is converted to a base equal to the modulus specified in the MOD parameter. The units positions of the result is stored as the DR; only one result is appropriate so the DL is forced to 0.

Example:

The modulus is 10

From the sum of products  
(Numbers in base 10)

NR = 107

NL = 0

NR + NL = 107

Sum of each digit

1+0+7 = 8

Modulus is taken on 8; value of units position is the result so:

DR = 8

DL = 0

Commonly, if this option is used the modulus is 10. The following is an example if the modulus is 7 and take into account the effects of expressing the values to the base of the modulus.

Example:

Modulus is 7

From the sum of products  
(Numbers in base 7)

NR = 26 base 7

NL = 16 base 7

NR + NL = 45 base 7

4+5 = 12 base 7

Modulus is taken on 12 base 7; value of units position is the result so:

DR = 2

DL = 0

Field 2 = K

Cross add sum of NL + NR.

DR = modulus of sum of odd-  
positioned digits of  
NL + NR

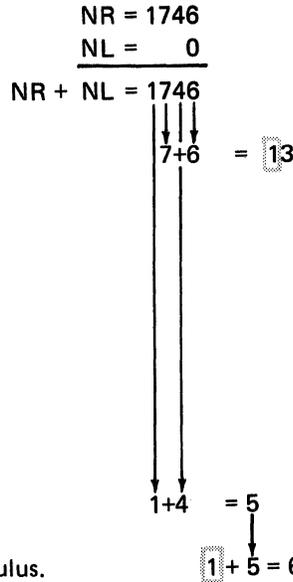
DL = modulus of sum of carry  
from DR, plus sum of  
remaining digits.

If Field 2 = K, the NR and NL are added together, then each odd-positioned digit of this sum is added; this new sum is converted to a base equal to the modulus specified in the MOD parameter, and the units position of this converted result is stored as the DR. Each even-positioned digit of the sum is added, and the remaining positions from the DR are added; this new sum is converted to a base equal to the modulus specified in the MOD parameter, and the units position of this converted result is stored as the DL.

Example:

Modulus is 10.

From the sum of products



Sum digits for first modulus (odd position).

First modulus is taken on 13.  
Result is units position value, so:

$$\text{DR} = 3$$

Remainder of first modulus equals 1.

Sum digits for second modulus.

Add remainder for second modulus.

Second modulus is taken on 1, so:

$$\text{DL} = 6$$

Commonly, if this option is used, the modulus is 10. If modulus 10 is not used, all the numbers and summations would be in the base of the modulus.

Field 2 = E

Special 8 and 3.

DR = 8 modulus NR

DL = 3 modulus NL

If Field 2 = E, the NR and NL are treated separately. The NR is converted to base 8, and the units position of this converted result is stored as the DR. The NL is converted to base 3, and the units position of this converted result is stored as the DL. This option should be used only if you use an output translate table.

Example:

Modulus is 10.  
From the sum of products

NR = 27  
Modulus 8 taken on 27  
27 base 10 = 33 base 8  
↓  
3

so result is:

DR = 3

From sum of products

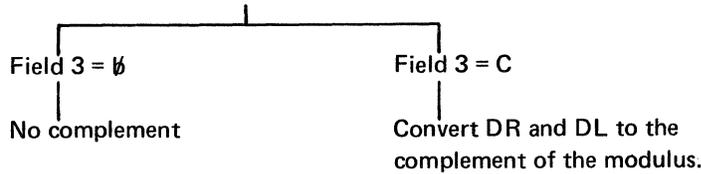
NL = 8  
Modulus 3 taken on 8  
8 base 10 = 22 base 3  
↓  
2

so result is:

DL = 2

4. Determine the output for the self-check number.

The output for the self-check number is determined by Field 3, which specifies whether to complement the DR and DL, by the OUTTAB parameter, which specifies whether an output translate table is used, and by Field 4, which specifies whether the self-check number is one or two digits. If the output translate table is omitted, Field 5 is also used to determine whether the DR and DL, or only the DR is used.



If Field 3 is blank, no complement is taken. If Field 3 is C, the DR and DL are taken to the complement of the modulus. This has the effect of subtracting the modulus value from the DR and DL value.

Example:

Modulus is 11.

DR = 7  
DL = 0

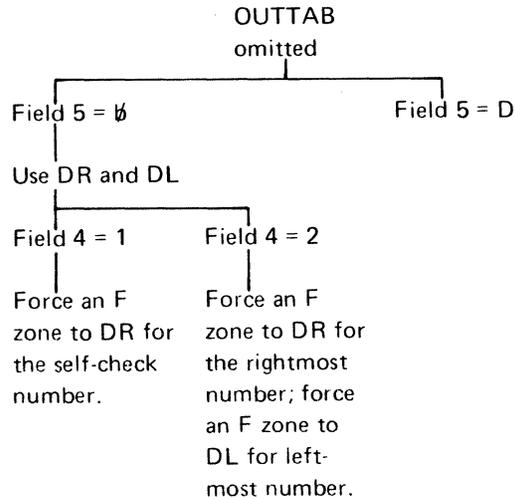
Complement DR to 11.  
DR = 4

11 - 7 = 4

Complement DL to 11.  
DL = 0

11 - 0 = 0  
The complement of 0 will always be 0.

**Note:** If you specify E for Field 2 (special 8 and 3), do not specify C for Field 3 if your modulus is less than 8.



If no output translate table is specified, and Field 5 of the CNTL parameter is blank, both the DR and DL are used to determine the self-check number output.

If Field 4 = 1, only one self-check digit is output for the self-check number. An F zone is forced to the hexadecimal representation of the DR, and this is used for the self-check number.

Example:

|                                  |    |
|----------------------------------|----|
| DR = 7                           |    |
| Hexadecimal representation of DR | 07 |
| Force an F zone to the DR        | F7 |

The self-check number is 7,  
which is displayed and printed as 7

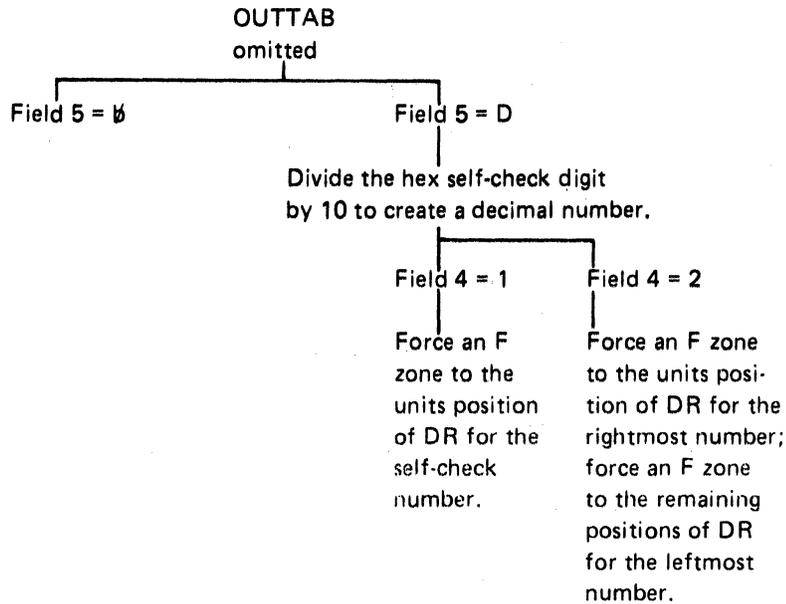
If Field 4 = 2, two digits are output for the self-check number. An F zone is forced to the hexadecimal representation of the DR, and this is used for the rightmost digit; an F zone is forced to the hexadecimal representation of the DL, and this is used for the leftmost digit.

Example:

|                                  |                |
|----------------------------------|----------------|
| DR = 7                           | Hexadecimal 07 |
| DL = 3                           | Hexadecimal 03 |
| Force an F zone to the DR and DL | F7F3           |

The self-check number is F7F3,  
which is displayed and printed as 73.

Usually, if this option is used, the DR and DL have a value of less than 10.



If Field 5 = D, only the DR is used to determine the self-check number output.

If Field 4 = 1, only one digit is output for the self-check number. An F zone is forced to the units position of the DR, and this is used for the self-check number.

Example:

DR = 10

Force an F zone to the units position.      F0

The self-check number is F0, which is displayed and printed as 0.

If Field 4 = 2, two digits are output for the self-check number. An F zone is forced to the units position of the DR, and this is used for the rightmost digit; an F zone is forced to the remaining positions, and this is used for the leftmost digit.

Example:

DR = 123

Force an F zone to the units position      F3

Force an F zone to the remaining positions      FC (12 is C in hex)

The self-check number is F3FC, which is displayed as 3 and printed as 3.

This option is usually used if:

- The modulus is greater than 10, there are 2 check digits, and the DL is 0.
- The modulus is 10 or less, and there is only one check digit.

For this option, DR usually has a maximum value of less than 100.



This page is intentionally left blank

EBCDIC CHARTS FOR PRINTABLE CHARACTERS

| Second Hex Digit | First Hex Digit | 0 | 1 | 2 | 3  | 4 | 5 | 6 | 7 | 8 | 9 | A | B | C | D | E | F |
|------------------|-----------------|---|---|---|----|---|---|---|---|---|---|---|---|---|---|---|---|
| 0                | 0               |   |   |   |    | & | - | ø | Ø | ° | µ | ¢ | ( | ) | \ | / | 0 |
| 1                | 1               |   |   |   |    | é | / | é | a | j | - | £ | A | J |   |   | 1 |
| 2                | 2               |   |   | ä | ë  | Ä | Ë | b | k | s | ¥ | B | K | S |   |   | 2 |
| 3                | 3               |   |   | ä | ë  | Ä | Ë | c | l | t | ℞ | C | L | T |   |   | 3 |
| 4                | 4               |   |   | à | è  | À | È | d | m | u | ƒ | D | M | U |   |   | 4 |
| 5                | 5               |   |   | á | í  | Á | Í | e | n | v | ₤ | E | N | V |   |   | 5 |
| 6                | 6               |   |   | ā | ī  | Ā | Ī | f | o | w | Ⱬ | F | O | W |   |   | 6 |
| 7                | 7               |   |   | ā | ī  | Ā | Ī | g | p | x |   | G | P | X |   |   | 7 |
| 8                | 8               |   |   | ç | ï  | Ç | Ï | h | q | y |   | H | Q | Y |   |   | 8 |
| 9                | 9               |   |   | ñ | β  | Ñ | β | i | r | z |   | I | R | Z |   |   | 9 |
| A                | A               |   |   | [ | ]  | [ | ] | : | « | » | ı | ı | ı | ı | ı | ı | ı |
| B                | B               |   |   | . | \$ | , | # | » | » | » | ı | ı | ı | ı | ı | ı | ı |
| C                | C               |   |   | < | *  | % | @ | d | z | Đ | - | ö | ü | Đ | Ü |   |   |
| D                | D               |   |   | ( | )  | _ | ' | ý | ı | ı | ı | ı | ı | ı | ı | ı | ı |
| E                | E               |   |   | + | ;  | > | = | ƒ | ƒ | ƒ | ' | ó | á | ó | ú |   |   |
| F                | F               |   |   | ! | ^  | ? | " | ± | ¤ |   | = | ö | ý | ö |   |   |   |

LANGUAGE: INTERNATIONAL

| Second Hex Digit | First Hex Digit | 0 | 1 | 2 | 3 | 4 | 5  | 6 | 7 | 8 | 9 | A | B | C | D | E | F |
|------------------|-----------------|---|---|---|---|---|----|---|---|---|---|---|---|---|---|---|---|
| 0                | 0               |   |   |   |   |   | &  | - |   |   |   |   |   | ( | ) | \ | 0 |
| 1                | 0               |   |   |   |   |   | /  |   |   | a | j | - |   | A | J |   | 1 |
| 2                | 0               |   |   |   |   |   |    |   |   | b | k | s |   | B | K | S | 2 |
| 3                | 0               |   |   |   |   |   |    |   |   | c | l | t |   | C | L | T | 3 |
| 4                | 0               |   |   |   |   |   |    |   |   | d | m | u |   | D | M | U | 4 |
| 5                | 0               |   |   |   |   |   |    |   |   | e | n | v |   | E | N | V | 5 |
| 6                | 0               |   |   |   |   |   |    |   |   | f | o | w |   | F | O | W | 6 |
| 7                | 0               |   |   |   |   |   |    |   |   | g | p | x |   | G | P | X | 7 |
| 8                | 0               |   |   |   |   |   |    |   |   | h | q | y |   | H | Q | Y | 8 |
| 9                | 0               |   |   |   |   |   |    |   |   | i | r | z |   | I | R | Z | 9 |
| A                | 0               |   |   |   |   | ¢ | !  | ! | : |   |   |   |   |   |   |   |   |
| B                | 0               |   |   |   |   | . | \$ | , | ‡ |   |   |   |   |   |   |   |   |
| C                | 0               |   |   |   |   | < | *  | % | @ |   |   |   |   |   |   |   |   |
| D                | 0               |   |   |   |   | ( | )  | _ | ' |   |   |   |   |   |   |   |   |
| E                | 0               |   |   |   |   | + | ,  | > | = |   |   |   |   |   |   |   |   |
| F                | 0               |   |   |   |   |   | ~  | ? | " |   |   |   |   |   |   |   |   |

IBM 5256 STANDARD CHARACTER SET

LANGUAGE: US AND CANADA

## KEYBOARD FUNCTIONS: EBCDIC CODES AND BIT NUMBERS

### IBM 5280 Mode Keyboard Functions

The EBCDIC is the code the microprocessor uses to define the key. The bit number is the number used for the TRAP parameter of the .KBCRT control statement.

| EBCDIC | Bit Number | Key  | Description   |
|--------|------------|--|---|
| X'00'  | —          | —  | Invalid scan code generated from translate table or hardware. An error code is presented to the operator.   |
| X'01'  | 0          | CMD  | Command key prefix to select command function.  |
| X'02'  | 1          | CMD  | Shifted command key.  |
| X'03'  | 2          | —  | Keyboard overrun; keyboard has lost two keystrokes due to hardware keystroke buffer overrun.  |
| X'04'  | 3          | —  | Invalid keystroke; the code is generated directly from the scan code translate table or the World Trade translate table.  |
| X'05'  | 4          | RESET  | Reset function; reset error condition, or reset Hex key command, or insert function.  |
| X'06'  | 5          | INS  | Insert function; initiate character insert.   |
| X'07'  | 6          | Del  | Delete function; initiate character delete.   |
| X'08'  | 7          | Alpha  | Alpha shift, with the Alpha key pressed.  |
| X'09'  | 8          |  or Num | Numeric shift, with the  (Shift) key or Num key pressed; uses the lower half of the scan code translate table. |
| X'0A'  | 9          |         | Shift lock, with the  (Shift Lock) key pressed; locks shift to use upper half of scan code translate table.    |
| X'0B'  | 10         | Num Shift  | Katakana numeric shift, with the Num Shift key pressed; uses upper half of scan code translate table.   |
| X'0C'  | 11         | Alpha Shift  | Katakana Alphabetic shift, with the Alpha shift key pressed; uses lower half of scan code translate table.  |
| X'0D'  | 12         | Kata Shift   | Katakana shift, with the Kata Shift key pressed; uses lower half of World Trade translate table.  |

| EBCDIC | Bit Number | Key                | Description   |
|--------|------------|--------------------|---|
| X'0E'  | 13         | Sym Shift          | Katakana upper shift; with Sym Shift key pressed; uses upper half of the World Trade translate table.     |
| X'0F'  | 14         | Lock               | Katakana shift lock; with the Lock key pressed; locks the shift to Kata Shift or to Katakana Alpha shift. |
| X'12'  | 17         | ↑                  | Move cursor up; valid only for format level zero.   |
| X'13'  | 18         | ↓                  | Move cursor down; valid only for format level zero.   |
| X'14'  | 19         | ←┘                 | Moves cursor to the first position on the next line; valid only for format level zero.                    |
| X'15'  | 20         | Field Exit, Field+ | Field exit function.  |
| X'16'  | 21         | Field -            | Field exit minus function.  |
| X'17'  | 22         | Skip               | Skip function.  |
| X'18'  | 23         | Alpha              | Alpha shift, with the Alpha key released.   |
| X'19'  | 24         | ↑<br>or Num        | Numeric shift, with the ↑ (Shift) key or Num key released.  |
| X'1A'  | 25         | ↓                  | Shift lock, with the ↓ (Shift Lock) key released.   |
| X'1B'  | 26         | Num Shift          | Katakana numeric shift, with the Num Shift key released.  |
| X'1C'  | 27         | Alpha Shift        | Katakana alphabetic shift, with the Alpha Shift key released.   |
| X'1D'  | 28         | Kata Shift         | Katakana shift, with the Kata Shift key released.   |
| X'1E'  | 29         | Sym Shift          | Katakana upper shift, with the Sym Shift key released.  |
| X'1F'  | 30         | Lock               | Katakana shift lock, with the Lock key released.  |
| X'20'  | 31         | Dup                | Duplicate function.   |
| X'21'  | 32         | →                  | Field advance function.   |

| EBCDIC   | Bit Number | Key                                      | Description   |
|----------|------------|--|---|
| X'22'    | 33         | ←  | Field backspace function.   |
| X'23'    | 34         | Corr                                     | Field correct function.   |
| X'24'    | 35         | Enter/<br>Rec Adv                        | Record advance function   |
| X'25'    | 36         | Home                                     | Record backspace function.  |
| X'26'    | 37         | →  | Character advance function.   |
| X'27'    | 38         | ←  | Character backspace function.   |
| X'28'    | 39         | Hex<br>command<br>function<br>key        | Hex command function key.   |
| —        | 40         |  | Keystroke error, detected and normally handled by the keyboard/display.           |
| X'29'    | —          | No key is associated with this function. | Clear screen function; blanks all positions on the screen except the status line. |
| X'2A'    | —          | No key is associated with this function. | Clear status line function; blanks all positions on the status line.              |
| X'2B'    | —          | No key is associated with this function. | Keystroke with this EBCDIC is ignored.  |
| X'2C'    | 43         | =  | Field-/dash combination key.  |
| X'2D-32' |            | Reserved                                 |   |

**Note:** Even if the user specifies via the keyboard bit map that the shifted or unshifted CMD key be processed by software, the microcode will continue to track and post CMD key sequences to the software through external status code 2 or 3.

Functions from 33-3F are handled by your program, with external status condition 1 subroutines.

| <b>EBCDIC</b> | <b>Bit Number</b> | <b>Key</b>    | <b>Description</b>  |
|---------------|-------------------|---------------|---|
| X'33'         |                   | Sel<br>Fmt    | Select format function.   |
| X'34'         |                   | Dup<br>Skip   | Switches the auto dup/skip flag.  |
| X'35'         |                   | Auto<br>Enter | Switches the auto enter flag.   |
| X'36'         |                   | Cncl          | Cancel function; defined and processed by your program.   |
| X'37'         |                   | Page<br>Fwd   | Page forward function; to read the next record without writing out the current record, processed by your program. |
| X'38'         |                   | Next<br>Fmt   | Next format function; to allow the operator to exit a repetitive format, processed by your program.               |
| X'39'         |                   | Prnt          | Print function; to initiate output from the printer.  |
| X'3A'         |                   |               | Not used.   |
| X'3B'         |                   | Erase<br>Inpt | Erase input function.   |
| X'3C'         |                   | Corr          | Record correct function; initiated by the shifted Corr key.   |
| X'3D'         |                   | Sys<br>Req    | System request function.  |
| X'3E'         |                   | Attn          | Attention function.   |
| X'3F'         |                   | Help          | Help function; to request a help message.   |

## IBM 3270 Mode Keyboard Functions

The internal code is the code the microprocessor uses to define the key during the IBM 3270 mode.

| Internal Code | Function   | General Description  |
|---------------|--|--|
| 00            | Invalid scan code                                  | Scan code improperly generated by hardware or problem in scan code translate table or world trade translate table. This internal code causes the invalid scan code to be logged to the hard error log and input inhibited to be set. |
| 80            | Ignore keystroke                                   | Keystroke with this code is ignored.   |
| 81            | Command  | Command key prefix to select function.   |
| 82            | Erase EOF  | Clears the cursor position and all positions to the right in the current field to nulls.   |
| 83            | Keyboard overrun                                   | Keyboard has lost 2 keystrokes due to hardware keystroke buffer overrun.   |
| 84            | Error keystroke                                    | Invalid keystroke due to undefined key position.   |
| 85            | Erase input  | Clears all unprotected character locations to nulls, resets MDT's, repositions cursor.   |
| 86            | Field mark   | Causes field mark character code to be inserted into the current field.  |
| 87            | NUM lock key                                       | Fixes or releases the upshift character selection on the data entry keyboard.  |
| 88/98         | Shift key make/<br>break numeric key<br>make/break | Shifts to upper half of scan code translate table, selecting characters on upper half of key.  |
| 89/99         | Alpha shift key<br>make/break                      | Shifts to lower half of scan code translate table, selecting character on lower half of key.   |
| 8A/9A         | Lock make/break                                    | Fixes the upshift character selection.   |
| 8B/9B         | Alpha/numeric<br>lower shift key<br>make/break     | On Katakana keyboard, shifts to lower half of scan code translate table, selecting character on lower left half of key.  |
| 8C/9C         | Alpha/numeric<br>upper shift key<br>make/break     | On Katakana keyboard, shifts to upper half of scan code translate table, selecting character on upper left half of key.  |

| <b>Internal Code</b> | <b>Function</b>                 | <b>General Description</b>  |
|----------------------|---------------------------------|---|
| 8D/9D                | Katakana lower shift make/break | On Katakana keyboard, shifts to lower half of world trade translate table, selecting character on lower right half of key.  |
| 8E/9E                | Katakana upper shift make/break | On Katakana keyboard, shift to upper half of world trade translate table, selecting character on upper right half of key.   |
| 8F/9F                | Katakana shift lock make/break  | Locks Katakana keyboard to Katakana lower or alpha/numeric lower shift.   |
| 90                   | Cursor up                       | Moves cursor up 1 line.   |
| 91                   | Cursor down                     | Moves cursor down 1 line.   |
| 92                   | Cursor right                    | Moves cursor one position to the right.   |
| 93                   | Cursor left                     | Moves cursor one position to the left.  |
| 94                   | New line                        | Moves cursor to the first unprotected character location of the next line.  |
| 95                   | DUP                             | Causes DUP character code to be inserted into the current field with tab to next field.   |
| 96                   | Insert mode                     | Puts keyboard into insert mode. Subsequent data keystrokes are inserted at the cursor position. Characters above and to the right of the cursor are shifted to the right. |
| 97                   | Delete                          | Deletes the character at the cursor position. Characters to the right in the same field within the same row are shifted one position to the left.                         |
| A0                   | Tab key<br>Skip key             | Moves cursor to first character location of next unprotected data field.  |
| A1                   | Backtab key                     | Backspaces cursor to first character location of current field or previous unprotected field.   |

All other function keys are processed by the application programs. See the IBM 3270 Information Display System Component Description Manual, GA27-2749, for a detailed description of the function keys.

**ASSEMBLER ERRORS AND MESSAGES**

*Miscellaneous Errors*

0002 Symbol table full  
0003 Cross-reference table full

*Sequence or Location Assignment Errors*

0004 Location counter inappropriate for register  
0005 Table statements not contiguous  
0006 Invalid sequence of .FMT or .SFMT  
0007 Statement must precede first instruction in SUB  
0009 Invalid parameter combination  
0010 One or more required parameters omitted  
  
0011 Label is multiply defined  
0012 Invalid combination of labels in expression  
0013 Length of LEVL=2 exceeds length of LEVL=1  
0017 Unable to open included data set  
0022 Branch target out of range  
0023 Statement not allowed in SUB assembly  
0024 .START must be first statement in program  
0025 Control statement sequence error  
0026 .KBCRT statement required in program

*Invalid Operand in Instruction*

0032 Instruction description not in table  
0033 Unbalanced parentheses  
0034 Must not use same register twice  
0035 Extra operands in instruction  
0036 Option or modifier not recognized  
0037 Third operand is not Rn, +, 0, or -  
0038 Invalid data set number  
0039

### *Invalid Parameter in Control Statement*

0040 Format parameter not valid  
0041 Invalid shift count  
0042 Displacement not multiple of 8  
0043 Displacement plus length too large  
0044 Pad length exceeds 64  
0045  $\text{MAXM} * (\text{ARGL} + \text{BYPASS}) > \text{table length}$   
0046  $\text{MAXM} \times (\text{ARGL} + \text{BYPASS}) < \text{table length}$   
0047 PB1 and PB2 must be on a 128-byte boundary  
0048 SCS conversion not allowed in keyed file  
0049 SCS conversion not specified, assumed  
  
0050 xxxxxxxx is an undefined symbol  
0051 Must not use both ELAB and ETAB  
0052 Either ELAB or ETAB must be used

### *Screen Format and Edit Format Messages*

0070 Unfinished BYPASS in series  
0071 BYPASS not permitted within a BYPASS  
0072 Must not have RG alone  
0074 Cursor position exceeds screen size  
0075 Must not have NORMAL and another attribute  
0076 BYPASS end with no BYPASS initiated  
0077 MD and MS must be identical  
0078 No .FMTEND on previous edit format  
0079 No .SFMTEND on previous screen format  
0080 No start statement for this series

### *Messages That Reference a Column Number*

1001 Invalid delimiter at column ##  
1002 Unrecognized keyword left of column ##  
1003 Sublist too long, truncated at column ##  
1004 More than 8 operands, truncated at column ##  
1005 Unbalanced parentheses  
1006 Duplicate keyword left of column ##

### *Messages That Reference an Operand Number*

2001 Code conversion error in operand ##  
2002 Invalid value (too large, etc) in operand #  
2003 Operand ## is wrong type  
2004 Operand ## is wrong length  
2005 Operand ## must not be omitted

### *Messages That Contain a Variable Keyword*

3001 xxxxxxxx has a conversion error  
3002 xxxxxxxx is invalid (too large, etc)  
3003 xxxxxxxx is the wrong data type  
3004 xxxxxxxx is the wrong length  
3005 xxxxxxxx must not be omitted

### *Control Statement Error*

4001 Too many prompts, constants or MPUPs  
4002 Too many screen format strings  
4003 Too many edit format strings  
4004 Too many table definitions  
4005 Too many indicators  
  
5002 TYPE is not STOR/BIN/DEC/PRMT/MDUP  
5003 PRMT is not SP/FP/CP  
5004 FLDF(1) is not A/N/W/X/Y/Z/D/H/S/F  
5005 PIC(2) is not A/N/W/X/Y/Z/D/H  
5006 CI(2) is not ON/OFF  
5007 Invalid combination of EDIT specifications  
  
5011 TYPE(1) is not SR/SW/SU/KR/KRN/KU/KUN/I  
5012 OPTION is not MAIN/SUB  
5013 CNTR is not K/B/C/F  
5015 TYPE(1) is not COM/SR/SW/CN  
5016 TYPE(2) is not CB/CM/BT  
5017 CAM is not BSC/SNA  
5018 RECFM is not F/FB/V/VB  
5020 MOD is not S10 or S11 or numeric  
  
5027 Either DEV or DEVID must be used

### **CONVERSION PROGRAM ERRORS**

#### **Error Cause and Recovery**

9400 You have entered invalid data for a data set name, a device address, or an option. Respecify the invalid entry.  
9900 You have issued an EOJ command. Terminate the current prompt.  
9998 An unexpected external status condition has occurred. Terminate the current prompt.

This page is intentionally left blank

Your assembler diskette contains two sample source programs, labeled SYSASMP and SYSACLP. SYSACLP is an ACL program that can be converted to IBM 5280 assembler language by using the ACL to IBM 5280 assembler language conversion aid. SYSACLP was originally published as a sample program in *A Programmer's Introduction to the Application Control Language*, GA21-9195-1. SYSASMP is a source program that performs the same job as the original ACL program but has been written in IBM 5280 assembler language to make better use of the language. The assembler listing for SYSASMP is illustrated in this appendix. You may assemble the sample program with any of the options available, but to obtain a listing like the one in this appendix, select the following assembler listing options:

|                  |     |                 |     |
|------------------|-----|-----------------|-----|
| Cross reference: | Yes | Included lines: | Yes |
| Full data print: | No  | Errors only:    | No  |
| Literal spacing: | No  | Page size:      | 84  |

Following the listing is an example of the output from the sample program.

5280 ASSEMBLER 01.00

SAMPLE2 - ORDER ENTRY SAMPLE PROGRAM,

```

ADDR  OBJECT CODE  LINE  SOURCE CODE
2 *****
3 *
4 * NAME - Order entry sample program
5 *
6 * PURPOSE - Illustrate assembler coding techniques.
7 *
8 * FUNCTION - Prompt for header and detail record data, write records
9 * to diskette, and optionally write to a printer log also.
10 * Also allows updating of previously entered records.
11 * The printer is not used when scrolling or updating
12 * previously entered records.
13 *
14 * OPERATION - The user is prompted for:
15 * 1. Data set name and address.
16 * 2. Printer address (optional).
17 * 3. Header record data fields.
18 * 4. Detail record data fields.
19 * 5. Additional detail records.
20 * Press the SELECT FORMAT key to enter a new header record.
21 * Press the HOME key to backspace into previous records.
22 * Most other function keys (RECORD ADVANCE, SYSTEM REQUEST,
23 * ATTENTION, CURSOR LEFT and RIGHT, CMD EOJ) have their
24 * usual significance. Other function keys and the command
25 * keys are ignored.
26 *
27 * OUTPUT - Header record written to diskette:
28 * columns contents
29 * 1 - 8 customer no. (self check)
30 * 9 - 16 order no.
31 * 17 - 22 date
32 * 23 - 24 salesman no.
33 * 25 - 34 post office
34 * 35 - 41 ship code (table look-up)
35 * 42-128 not used, blank
36 * Detail record written to diskette:
37 * columns contents
38 * 1 - 8 customer no. (from header)
39 * 9 - 16 order no. (from header)
40 * 17 - 24 item no.
41 * 25 - 29 quantity (limit check)
42 * 30 - 36 unit price
43 * 37 - 49 amount = quantity * price
44 * 80 character D - indicates detail
45 * 50-128 not used, blank
46 * Printed log (optional):
47 * The header data and the first detail record are formatted
48 * on the same line. Additional detail records for the same
49 * header are placed below it without repeating header data.
50 *
51 * RESTRICTIONS - Basic common function area (SYSDPRT2) is required.
52 * The output dataset must be previously allocated.
53 *
54 *****

```

```

0000 E2C1D4D7D3C5F240
0080 0010000000000000
00A6
009E
0140
0001
0114
0000
0115
0140 0800
0142
0144
0146
0148 0435
014A 08C2

```

```

56 .START PNAME='SAMPLE2' ENTRY=START MCHK=CFPGMCHK;
57 .KBCRT CRBA=IOBUF ETAB=ERRKB;
58 .DC LABEL=3CNCD LEVL=2 DISP='X'A6'; SCAN CODE & CONVERTED EBCDIC
59 .DC LABEL=MODESW LEVL=2 DISP='X'BE'; KB/CRT MODE SWITCH
60 .DCLIND LABEL=EQDSW,PRNTSW,QUANSW;
61 .EQUATE NUMB=(1,TRANS),(2,PRNT);
62 * THE FOLLOWING EQUATES ARE OPTIONAL, BUT WILL PUT ENTRIES IN THE
63 * CROSS REFERENCE LISTING FOR SYSTEM REGISTERS AND INDICATORS.
64 .EQUATE REG=(BR10,BR10),(BR18,BR10),(BR19,BR19),(BR22,BR22);
65 .EQUATE IND=(I115,I115),(I118,I118),(I125,I125),(I158,I158);
66 .XTRN LABEL=CFPGMCHK,CFGIOERR,CFPERATT,CFLOAD01,CFERCD5M,
67 CFATFBGD,CFDEVOCHK;
68 .DC LABEL=CFWKPTR TYPE=BIN INIT=ADDR(CFSAVE); C.F. WORK AREA PTR
69 .DC LABEL=CFPARM1 TYPE=BIN; C.F. PARAMETER REG
70 .DC LABEL=CFPARM2 TYPE=BIN; C.F. PARAMETER REG
71 .DCLBR LABEL=WKBR1; BINARY WORK REGISTERS
72 .DC LABEL=RCOL TYPE=BIN INIT='X'0435'; ROW & COL FOR CRTMH
73 .DC LABEL=BUF# TYPE=BIN INIT=ADDR(IOBUF); I/O BUFFER PTR

```

5280 ASSEMBLER 01.00

SAMPLE2 - ORDER ENTRY SAMPLE PROGRAM,

```

ADDR  OBJECT CODE  LINE  SOURCE CODE
014C 08E4
0150 0150
0150 8000
0152 0006
0154 0022
0156 0029
0158 3701
0160
0190 F0F0F0F0F0F0F0
01A0 F9F9F9F9F9F9F9
01A5 F9F9F9F9F9F9F9
0200 0000000000000000
0208 0000000008C2FF00
0280 0000000000000000
0288 000000000840FF00
0290 0D50000000000000
0300
0700
0800
0840
0888
08C2
E3D9C1D5E2404040
08C2
08C2
08C4
08D2
08D8
08E4
08D2
08DA
08DF
08E6
0911
0942 0201020102010201
096E 0986000506000005
0976 0986000500060005
097E 0E0C000401000004
0986
0986 E3D9E4C3D24040
098D D9C1C9D3404040
0994 D7D6E2E3404040
0998 C1C9D940404040
09A2 C3D6E4D5E3C5D9
09A9 FF0F9A
09AC 070199
09AF 0F41B50702B8
09B5 0F41960703BC
09B8 0F41920187
09C0 0F41B707048D
09C6 0F41B202034084
09CD 0F83
09CF FF0F9A
74 .DC LABEL=TXTR TYPE=BIN INIT=ADDR(SHPCD); PTR FOR CRTMH
75 .DC LABEL=PRCNTL# TYPE=BIN INIT=ADDR(PRCNTL); PTR TO PRINT CONTROL
76 .DC LABEL=PRCNTL TYPE=BIN INIT='X'8000'; PRINTER SPACE CONTROL
77 .DC LABEL=K6 TYPE=BIN INIT=6;
78 .DC LABEL=K22 TYPE=BIN INIT='X'22';
79 .DC LABEL=K29 TYPE=BIN INIT='X'29';
80 .DC LABEL=K3701 TYPE=BIN INIT='X'3701'; CODE FOR EDD
81 .DCLDR LABEL=WKDR1,QUAN,RPRICE; DECIMAL WORK REGISTERS
82 .DC LABEL=LIMIT TYPE=DEC INIT=1000; QUANTITY LIMIT CHECK
83 .DC LABEL=MSG911 INIT='X'F9F9F9F9F9F9F9'; 9111 - SELF CHECK INCORRECT
84 .DC LABEL=MSG999B INIT='X'F9F9F9F9F9F9F9'; 999B - UNEXPECTED KB EXT STATUS
86 * TRANSACTION DATA SET DEFINITION
87 .DATASET DSN=TRANS DEVID=D1 RECL=128 BSIZ=0 TYPE=SU NAME=DSNAME
88 LBUF=IOBUF PB1=BUFF1 ELAB=DSKERR;
90 * PRINTER DEFINITION
91 .DATASET DSN=PRNT DEV='X'8000' RECL=120 BSIZ=256 TYPE=SW,SCS
92 PSSIZ=66 LINSZ=120 LSTLN=50
93 LBUF=PRBUF PB1=BUFF2 ELAB=GI0ERR;
94 .DC LABEL=BUFF1 LEN=1024 BDY=128;
95 .DC LABEL=BUFF2 LEN=256 BDY=128;
96 .DC LABEL=CFSAVE LEN=64;
97 .DC LABEL=PRBUF LEN=120;
98 .DC LABEL=DSNAME LEN=10;
100 * LOGICAL BUFFER FOR DISKETTE & KEYBOARD
101 .DC LABEL=IOBUF LEN=128;
102 .DC LEVL=2 INIT='TRANS XXXX8000'; DATASET OPEN PARAMETERS
103 .DC LEVL=2 LABEL=CUSTOM LEN=8 DISP=0; CUSTOMER NO.
104 .DC LEVL=2 LABEL=ORDNO LEN=8; ORDER NO.
105 .DC LEVL=2 LABEL=DATE LEN=6; DATE
106 .DC LEVL=2 LABEL=QUAN LEN=2; SALESMAN NO.
107 .DC LEVL=2 LABEL=POSTD LEN=10; POST OFFICE
108 .DC LEVL=2 LABEL=SHPCD LEN=7; SHIP CODE
109 .DC LEVL=2 LABEL=ITMNO LEN=8 DISP=16; ITEM NO.
110 .DC LEVL=2 LABEL=QUAN LEN=5; QUANTITY
111 .DC LEVL=2 LABEL=PRICE LEN=7; PRICE
112 .DC LEVL=2 LABEL=AMT LEN=13; QUAN * PRICE
113 .DC LEVL=2 LABEL=CNTRL LEN=1 DISP=79; CONTROL CHAR
115 .SELFCHK NOB=S10; STANDARD SELF CHECK
117 * TABLE OF TEXT FOR SHIP CODE
118 .TABLE LABEL=SHIPTXT DCLBL=SHIPCODE ARG1=7 ENTRIES=5;
119 .TABLE LABEL=SHIPTD DCLBL=SHIPCODE ARG1=1 BYPAS=6 ENTRIES=5;
120 .TABLE LABEL=RGRTM DCLBL=BUFAD ENTRIES=4 ARG1=2; TEXT FOR SHIP CODE
121 .DC LABEL=SHIPCODE LEN=35;
122 .DC LEVL=2 LEN=7 INIT='TRUCK';
123 .DC LEVL=2 LEN=7 INIT='RAIL';
124 .DC LEVL=2 LEN=7 INIT='POST';
125 .DC LEVL=2 LEN=7 INIT='AIR';
126 .DC LEVL=2 LEN=7 INIT='COUNTER';
128 * SCREEN FORMATS FOR DATASET OPEN PROMPTS
129 .SFMTST LABEL=PRTRANS CNTRL=CS,MV; PROMPT FOR TRANSACTION FILE
130 .SFMTPT PRINT=SP,TITLE;
131 .SFMTPT PRINT=SP,TRANS1 CSPS=NL;
132 .SFMTPT PRINT=SP,TRANS2 CSPS=NL; DATASET NAME
133 .SFMTFLD FLDF=M,4,B CSPS=3;
134 .SFMTPT PRINT=SP,TRANS3 CSPS=NL;
135 .SFMTFLD FLDF=M,4,B CSPS=3; DEVICE ADDRESS
136 .SFMTEND;
138 .SFMTST LABEL=PRPRINT CNTRL=CS,MV; PROMPT FOR PRINTER ADDRESS

```

| ADDR | OBJECT CODE       | LINE | SOURCE CODE   |
|------|-------------------|------|---|
| 09D2 | 070199            | 139  | .SFMTPT PRMT=SP, TITLE;   |
| 09D5 | 0F41B507059B      | 140  | .SFMTPT PRMT=SP, PRINT1 CSPS=NL;                                      |
| 09D8 | 0F218B0F41820203  | 141  | .SFMTFLD FLDF=N, 4, BC CSPS=+3 BFPS=+12;                              |
| 09E5 | 0F41A070709A      | 142  | .SFMTPT PRMT=SP, PRINT2 CSPS=NL;                                      |
| 09E8 | 0FB8              | 143  | .SFMTEND;   |
| 09ED | E2C1D4D73C540D6   | 145  | .DC TYPE=PRMT LABEL=TITLE INIT='SAMPLE ORDER ENTRY PROGRAM';          |
| 0A07 | C5D5E3C2D940E3C8  | 146  | .DC TYPE=PRMT LABEL=TRANS;  |
| 0A0F | C340CC6A03D3D4E6  | 147  | INIT='ENTER THE FOLLOWING INFORMATION FOR TRANSACTION DATA SET:';     |
| 0A40 | C4CE3C140E2C5E3   | 148  | .DC TYPE=PRMT LABEL=TRANS2 INIT='DATA SET NAME';                      |
| 0A4D | C4C5E3C9C3C540C1  | 149  | .DC TYPE=PRMT LABEL=TRANS3 INIT='DEVICE ADDRESS';                     |
| 0A58 | C5D5E3C2D940D7D9  | 150  | .DC TYPE=PRMT LABEL=PRINT1 INIT='ENTER PRINTER DEVICE ADDRESS';       |
| 0A77 | D3C5C1E5C540C2D3  | 151  | .DC TYPE=PRMT LABEL=PRINT2 INIT='LEAVE BLANK IF NOT PRINTING';        |
| 0A92 | FF0F8A            | 153  | * SCREEN FORMATS FOR ORDER ENTRY                                      |
| 0A95 | 0F41CF070787      | 155  | .SFMTPT PRMT=SP, HEADR1 CSPS=NL; PROMPT FOR HEADER DATA               |
| 0A9B | 0F41817B87        | 156  | .SFMTFLD FLDF=D, 8, RZ CSPS=2 CNTL=RG; CUSTOMER                       |
| 0AA0 | 0F4183070884      | 157  | .SFMTPT PRMT=SP, HEADR2 CSPS=4; ORDER                                 |
| 0AA6 | 0F4183D0784       | 158  | .SFMTFLD FLDF=D, 8, RZ, DR CSPS=2; DATE                               |
| 0AAC | 0F418307083       | 159  | .SFMTPT PRMT=SP, HEADR3 CSPS=4; SALESMAN                              |
| 0AB2 | 0F41813B85        | 160  | .SFMTFLD FLDF=D, 6, RZ CSPS=2; SALESMAN                               |
| 0AB7 | 0F419A070A87      | 161  | .SFMTPT PRMT=SP, HEADR4 CSPS=NL; SALESMAN                             |
| 0ABD | 0F41813B81        | 162  | .SFMTFLD FLDF=D, 2, RZ CSPS=2; P. O.                                  |
| 0AC2 | 0F4189070884      | 163  | .SFMTPT PRMT=SP, HEADR5 CSPS=10; P. O.                                |
| 0ACB | 0F41813B81        | 164  | .SFMTFLD FLDF=A, 10, FE CSPS=2; SHIP CODE                             |
| 0ACD | 0F4181070C8B      | 165  | .SFMTPT PRMT=SP, HEADR6 CSPS=2; SHIP CODE                             |
| 0AD3 | 0F418101004484    | 166  | .SFMTFLD FLDF=A, 1, BC, DR CSPS=2; SHIP CODE                          |
| 0ADA | 0FB3              | 167  | .SFMTEND;   |
| 0ADC | FF0F82            | 169  | .SFMTST LABEL=PRSHIP; PROMPT FOR SHIP CODE ERROR                      |
| 0ADF | 060D95            | 170  | .SFMTPT PRMT=FF, HEADR7; SHIP CODE ERROR                              |
| 0AE2 | 0F21A10F417F01D3  | 171  | .SFMTFLD FLDF=A, 1, BC, DR CSPS=212 BFPS=34;                          |
| 0AE6 | 0600A7            | 172  | .SFMTPT PRMT=CP, 40; CLEAR FIXED PROMPT LINE                          |
| 0AF1 | 0FB3              | 173  | .SFMTEND;   |
| 0AF3 | FF0F82            | 175  | .SFMTST LABEL=PRDETL; PROMPT FOR DETAIL RECORD DATA                   |
| 0AF6 | 0F6480            | 176  | .SFMTCTL CI=EODSW, OFF; BYPASS IF UPDATING                            |
| 0AF9 | 0F417F01EF07149B  | 177  | .SFMTPT PRMT=SP, BLANKS CSPS=240; CLEAR LINE 5 - PREVIOUS DETAIL LINE |
| 0B01 | 07149B            | 178  | .SFMTPT PRMT=SP, BLANKS;  |
| 0B04 | 07149B            | 179  | .SFMTPT PRMT=SP, BLANKS;  |
| 0B07 | 0F51CA0F2480      | 180  | .SFMTCTL CSPS=-75 CNTL=CE; END BYPASS                                 |
| 0B0D | 070E83            | 181  | .SFMTPT PRMT=SP, DETL1; ITEM  |
| 0B10 | 0F218F0F41813B07  | 182  | .SFMTFLD FLDF=D, 8, RZ, DR CSPS=2 BFPS=16;                            |
| 0B19 | 0F4183070787      | 183  | .SFMTPT PRMT=SP, DETL2 CSPS=4; QUANTITY                               |
| 0B1F | 0F41817B0484      | 184  | .SFMTFLD FLDF=D, 5, RZ, DR CSPS=2 CNTL=RG; BYPASS IF QUANTITY IS OK   |
| 0B25 | 0F6482            | 185  | .SFMTCTL CI=QUANSW, OFF;  |
| 0B28 | 0F517F02900EAA07  | 186  | .SFMTPT CSPS=-273 DSPLY=HI, BL  |
| 0B30 | 10970EA0          | 187  | PRMT=SP, DETL3; QUANTITY ERROR MESSAGE                                |
| 0B3A | 071490            | 188  | .SFMTPT PRMT=SP, DETL4; QUANTITY ERROR OPTIONS                        |
| 0B37 | 0F21810F41850100  | 189  | .SFMTFLD FLDF=A, 1, BC, DR CSPS=+6 BFPS=+50;                          |
| 0B41 | 4680              | 190  | .SFMTPT PRMT=CP CNTL=RG; CLEAR FIXED PROMPT LINE                      |
| 0B43 | 0F31B20F417F01DE  | 191  | .SFMTCTL CSPS=+223 BFPS=-51 CNTL=CE; END BYPASS, RESTORE CSPS & BFPS  |
| 0B4E | 0F4182071284      | 192  | .SFMTPT PRMT=SP, DETL5 CSPS=3; PRICE                                  |
| 0B54 | 0F41813B0684      | 193  | .SFMTFLD FLDF=D, 7, DR, RZ CSPS=2;                                    |
| 0B5A | 0F4183071385      | 194  | .SFMTPT PRMT=SP, DETL6 CSPS=4; AMOUNT                                 |
| 0B60 | 0F41814B0C83      | 195  | .SFMTFLD FLDF=D, 13, BY CSPS=2 CNTL=RG;                               |
| 0B66 | 0FB3              | 196  | .SFMTEND;   |
| 0B68 | C3EAE2E3D6D4C5D9  | 198  | .DC LABEL=HEADR1 TYPE=PRMT INIT='CUSTOMER';                           |
| 0B70 | D6D9C4C5D9        | 199  | .DC LABEL=HEADR2 TYPE=PRMT INIT='ORDER';                              |
| 0B75 | C4C1E3C5          | 200  | .DC LABEL=HEADR3 TYPE=PRMT INIT='DATE';                               |
| 0B79 | E2C1D3C5E2D4C1D5  | 201  | .DC LABEL=HEADR4 TYPE=PRMT INIT='SALESMAN';                           |
| 0B81 | D74840D48         | 202  | .DC LABEL=HEADR5 TYPE=PRMT INIT='P. O.';                              |
| 0B86 | E2C8C9D740C3D6C4  | 203  | .DC LABEL=HEADR6 TYPE=PRMT INIT='SHIP CODE';                          |
| 0B8F | E2C8C9D740C3D6C4  | 204  | .DC LABEL=HEADR7 TYPE=PRMT INIT='SHIP CODE ERROR, REKEY';             |
| 0BA5 | C9E3C5D4          | 205  | .DC LABEL=DETL1 TYPE=PRMT INIT='ITEM';                                |
| 0BA9 | D8E4C1D5E3C9E3E8  | 206  | .DC LABEL=DETL2 TYPE=PRMT INIT='QUANTITY';                            |
| 0BB1 | C5E7C8C5C4E240    | 207  | .DC LABEL=DETL3 TYPE=PRMT INIT='EXCEEDS QUANTITY LIMIT. ';            |
| 0BC9 | C17E1C1C3C3C5D7E3 | 208  | .DC LABEL=DETL4 TYPE=PRMT INIT='ACCEPT, R=REKEY';                     |
| 0BD4 | D7D9C9C3C5        | 209  | .DC LABEL=DETL5 TYPE=PRMT INIT='PRICE';                               |
| 0BDF | C1D4D6E4D5E3      | 210  | .DC LABEL=DETL6 TYPE=PRMT INIT='AMOUNT';                              |
| 0BE5 | 4040404040404040  | 211  | .DC LABEL=BLANKS TYPE=PRMT LEN=25 INIT=' ';                           |

| ADDR | OBJECT CODE      | LINE | SOURCE CODE   |
|------|------------------|------|---|
| 0BFE | FFFF40           | 213  | * EDIT FORMATS FOR PRINTER  |
| 0C01 | 80420060070708D2 | 214  | .FMST LABEL=DILFMT;   |
| 0C02 | 0764040400A0     | 215  | .FMFTLD DCLBL=ITMNO TYPE=DEC LEN=8 COL=60;                        |
| 0C10 | 0C64090608BF2    | 216  | .FMFTLD DCLBL=QUAN TYPE=DEC LEN=5 COL=70 EDIT=BF;                 |
| 0C17 | 3468110C08E6202  | 217  | .FMFTLD DCLBL=PRICE TYPE=DEC LEN=10 COL=77 EDIT=FL, CP, DP, 2;    |
|      |                  | 218  | .FMFTEND DCLBL=AMT TYPE=DEC LEN=18 COL=89 EDIT=FX, AF, CP, DP, 2; |
| 0C1F | FFFF40           | 220  | * EDIT FORMATS FOR PRINTER  |
| 0C22 | 0760070708C2     | 221  | .FMST LABEL=HDRFMT;   |
| 0C28 | 0660070708CA     | 222  | .FMFTLD DCLBL=CUSTM TYPE=DEC LEN=8 COL=1;                         |
| 0C2E | 066807508D20008  | 223  | .FMFTLD DCLBL=ORDNO TYPE=DEC LEN=8 COL=11;                        |
| 0C36 | 0460010108DB     | 224  | .FMFTLD DCLBL=DATE TYPE=DEC LEN=8 COL=11; EDIT=C;                 |
| 0C3C | 0C60090708DA     | 225  | .FMFTLD DCLBL=MANN0 TYPE=DEC LEN=2 COL=31;                        |
| 0C42 | 2960060608E4     | 226  | .FMFTLD DCLBL=POST0 TYPE=DEC LEN=10 COL=35;                       |
|      |                  | 227  | .FMFTEND DCLBL=SHIPCD TYPE=DEC LEN=7 COL=47;                      |
| 0C48 | 2B41FF46         | 229  | START: SRAT (TRANS, WKBR1); GET DEV ADDR OF 'D1'                  |
| 0C4C | 04760C54         | 230  | IFIR I118 IS ON GOTO OPNTRN; BR IF SRAT FAILED                    |
| 0C50 | 494608CA         | 231  | BIMHEX (IOBUF(4)+B, WKBR1); ELSE USE AS DEFAULT FOR PRMT          |
| 0C54 | 0B00012D         | 232  | OPNTRN: CALL CFATFBDD; ASSURE KB IS ATTACHED                      |
| 0C58 | CF800000         | 233  | ENTR (PRTRANS); PROMPT FOR TRANS. FILE PARMS                      |
| 0C5C | 99440001         | 234  | CFPARM2 = ADDR; DSN FOR CFDEVCHK                                  |
| 0C60 | 994208CA         | 235  | CFPARM1 = ADDR(IOBUF+8); DEV ADDR PTR                             |
| 0C64 | 0B000139         | 236  | CALL CFDEVCHK; VALIDATE AND STORE DEVICE ADDR                     |
| 0C68 | 00000C54         | 237  | GOTO OPNTRN; BR TO REPRMPT IF INVALID                             |
| 0C6C | 876108C2         | 238  | WKDR1 = IOBUF(8); MOVE DATASET NAME                               |
| 0C70 | 87618BB8         | 239  | DSNAME(B) = WKDR1;  |
| 0C74 | 2241FF00         | 240  | OPEN (TRANS); OPEN TRANSACTION FILE                               |
| 0C78 | 25B10C55         | 241  | IFDS1 B, TRANS NOT ON GOTO OPNTRN; BR IF OPEN FAILED              |
| 0C7C | 24C1FF00         | 242  | PSDN (TRANS, EOD); PREPARE TO ADD TO EXISTING DATA                |
| 0C80 | 4000FFFF         | 243  | SDN (EODSW); INDICATE ADD   |
| 0C84 | 0B00012D         | 245  | OPNPR: CALL CFATFBDD; ASSURE KB IS ATTACHED                       |
| 0C88 | 87600100         | 246  | ENTR (PRPRINT); PROMPT FOR PRINTER PARMS                          |
| 0C8C | 4C400BC6         | 247  | IFC 12, IOBUF NOT C' ' SKIP;                                      |
| 0C90 | 00000C80         | 248  | GOTO BLNK; BR IF PRINTER NOT REQ'D                                |
| 0C94 | 99440002         | 249  | CFPARM2 = PRMT; DSN FOR CFDEVCHK                                  |
| 0C98 | 994208CE         | 250  | CFPARM1 = ADDR(IOBUF+12); DEV ADDR PTR                            |
| 0C9C | 0B000139         | 251  | CALL CFDEVCHK; VALIDATE AND STORE DEVICE ADDR                     |
| 0CA0 | 00000C80         | 252  | GOTO OPNPR; BR TO REPRMPT IF INVALID                              |
| 0CA4 | 2242FF00         | 253  | OPEN (PRNT); OPEN PRINTER   |
| 0CAC | 25B10C85         | 254  | IFDS1 B, TRANS NOT ON GOTO OPNPR; BR IF OPEN FAILED               |
| 0CC8 | 4001FFFF         | 255  | SDN (PRNTSW); INDICATE PRINTER PRESENT                            |
| 0CBO | B9404A00         | 257  | * PROMPT FOR HEADER DATA  |
| 0CB4 | B04E4A00         | 258  | BLNK: O(BUFE) = C' '; SEED A BLANK                                |
| 0CB8 | 0B00012D         | 259  | DUP (O, BUFE, 79); BLANK THE BUFFER                               |
| 0CB8 | 0B00012D         | 260  | CUST: CALL CFATFBDD; ASSURE KB ATTACHED                           |
| 0CBC | CF800200         | 261  | ENTR (PRHDR); PROMPT FOR CUST NO. ETC.                            |
| 0CC0 | 806108E4         | 262  | SHIP: WKDR1 = SHPCD(1); GET SHIP CODE                             |
| 0CC4 | 53014661         | 263  | WKBR1 = TBFX(SHIPID, WKDR1); LOOK UP IN TABLE                     |
| 0CC8 | 04700C9D         | 264  | IFIR I125 NOT ON GOTO SHIPOK; BR IF CODE FOUND OK                 |
| 0CC8 | 0B00012D         | 265  | CALL CFATFBDD; ASSURE KB ATTACHED                                 |
| 0CC0 | CF800300         | 266  | ENTR (PRSHIP); ELSE GIVE ERROR PROMPT                             |
| 0CD4 | 00000C80         | 267  | GOTO SHIP; RECHECK SHIP CODE                                      |
| 0CDB | 52004661         | 269  | SHIPOK: WKDR1 = TBFX(SHIPTXT, WKBR1); GET TEXT FOR SHIP CODE      |
| 0CDE | 86618BE4         | 270  | SHPCD = WKDR1; MOVE IN RECORD BUFFER                              |
| 0CE0 | CA5244A9         | 271  | CRTHM (RMCDL, TXTB, K6); PUT TO DISPLAY SCREEN                    |
| 0CE4 | 03140B34         | 273  | * WRITE HEADER RECORD   |
| 0CE8 | 07010D05         | 274  | IF BR10 IS 0 GOTO READ; BR IF RECORD WAS NOT MODIFIED             |
| 0CEC | 99509003         | 275  | IFIR PRNTSW NOT ON GOTO WRTHD; BR IF NOT PRINTING                 |
| 0CF0 | 04073CF0         | 276  | PRCNTL = X'9003'; SET FOR TRIPLE SPACE                            |
| 0CF4 | 99509003         | 277  | IFIR I115 NOT ON GOTO PRTHD; SKIP IF NOT AT END OF PAGE           |
| 0CF8 | 30C2F4E          | 278  | PRCNTL = X'5003'; SET FOR PAGE SKIP                               |
| 0CF8 | 30C2F4E          | 279  | PRTHD: WRT (PRMT, PRCNTL); SPACE BEFORE PRINTING                  |
| 0CFC | 99508000         | 280  | PRCNTL = X'8000'; SET FOR SPACE SUPPRESS                          |
| 0D00 | 30D2024E         | 281  | WRT (PRMT, HDRFMT, PRECNTL, B); WRITE TO PRINTER, SPACE SUPPRESS  |
| 0D04 | 3041FF08         | 282  | WRTHD: WRT (TRANS, O); WRITE TO DISKETTE                          |

5280 ASSEMBLER 01.00

SAMPLE2 - ORDER ENTRY SAMPLE PROGRAM;

| ADDR | OBJECT CODE      | LINE | SOURCE CODE   |
|------|------------------|------|---|
| OD08 | 07000D35         | 283  | IFI EODSW NOT ON GOTO READ; BR IF UPDATING OLD RECORDS        |
|      |                  | 285  | * PROMPT FOR DETAIL RECORD DATA                               |
| OD0C | B9404A10         | 286  | DETL: 16(BUFG) = C' '; SEED A BLANK                           |
| OD10 | B01F4A10         | 287  | DUP (16,BUFG,32); BLANK THE BUFFER                            |
| OD14 | 0800012D         | 288  | ITEM: CALL CFATFBGD; ASSURE KB ATTACHED                       |
| OD18 | CF800400         | 289  | ENTR (PRDETL); PROMPT FOR ITEM NO. ETC.                       |
|      |                  | 291  | * WRITE DETAIL RECORD   |
| OD1C | 03140D34         | 292  | IF BR10 IS 0 GOTO READ; BR IF RECORD WAS NOT MODIFIED         |
| OD20 | 07010D29         | 293  | IFI PRNTSW NOT ON GOTO WRTDT; BR IF NOT PRINTING              |
| OD24 | 30520108         | 294  | WRT (PRNT,DTLFMT,,B); WRITE TO PRINTER                        |
| OD28 | 44C40911         | 295  | WRTDT: CNTL = C'D'; CONTROL CODE FOR DATA DIR FMT             |
| OD2C | 3041F08          | 296  | WRT (TRANS,,0); WRITE TO DISKETTE                             |
| OD30 | 07000D0C         | 297  | IFI EODSW IS ON GOTO DETL; BR TO DO NEXT ITEM                 |
|      |                  | 299  | * READ FROM TRANSACTION FILE, DATA DIRECTED FORMAT            |
| OD34 | 20410009         | 300  | READ: READ (TRANS,,*); READ NEXT RECORD                       |
| OD38 | 07000D0C         | 301  | IFI EODSW IS ON GOTO DETL; BR IF REACHED EOD                  |
| OD3C | 4EC40911         | 302  | SELECT: IFC CNTL IS C'D' SKIP; SKIP IF DETAIL RECORD          |
| OD40 | 00000CB8         | 303  | GOTO CUST; ELSE UPDATE HEADER RECORD                          |
| OD44 | 00C00D14         | 304  | GOTO ITEM; UPDATE DETAIL RECORD                               |
|      |                  | 306  | * DISKETTE AND PRINTER EXTERNAL STATUS                        |
| OD48 | B8462C04         | 307  | DSKERR: WKBRI = 4(2,BK22); GET ERROR CODE FROM IOB            |
| OD4C | 6E465803         | 308  | IF WKBRI EQ K3701 GOTO EOD; BR IF AT END OF DATA              |
| OD50 | 08000119         | 309  | GIOERR: CALL CFGIOERR; ELSE POST TO STATUS LINE               |
| OD54 | 00000DA0         | 310  | GOTO EOJ; BR IF END OF JOB RESPONSE                           |
| OD58 | 0C000000         | 311  | RETURN; ELSE GO BACK & TRY AGAIN                              |
| OD5C | 4000FFFF         | 312  | SON (EODSW); INDICATE AT EOD                                  |
| OD60 | 984700BE         | 313  | WKBRI = MODESW; GET KB/ERT MODE BYTE                          |
| OD64 | 9F4600C0         | 314  | WKBRI X= X'CO'; CHANGE UPDATE TO ENTER MODE                   |
| OD68 | A24700BE         | 315  | MODESW = WKBRI; PUT IT BACK                                   |
| OD6C | 25820B75         | 316  | IFDSI 8,PRNT NOT ON GOTO DKRTN; SKIP IF PRINTER NOT AVAILABLE |
| OD70 | 4004F0F0         | 317  | SON (PRNTSW); ELSE TURN FRONT SWITCH BACK ON                  |
| OD74 | 0C000000         | 318  | DKRTN: RETURN; GO DO NEXT ITEM                                |
|      |                  | 320  | * KEYBOARD EXTERNAL STATUS                                    |
| OD78 | 0D940DBC0DD00D94 | 321  | .LABTAB LABEL=ERRKB ENTRY=DEXS,COMK,COMS,FWRG,BKRG;           |
| OD80 | 0E000EA80EA80E80 | 322  | RCAD,RCBK,KERR,NEXS,INOP,MAGR,WEXS,FCSE;                      |
| OD94 |                  | 323  | DEXS: ; 0 - DOUBLE EXTERNAL STATUS                            |
| OD94 |                  | 324  | COMS: ; 3 - SHIFTED COMMAND KEY                               |
| OD94 |                  | 325  | KERR: ; 8 - KEY STROKE ERROR                                  |
| OD94 |                  | 326  | NEXS: ; 9 - I/O MEMORY PARITY ERROR                           |
| OD94 |                  | 327  | INOP: ; 10 - INVALID OPERATION                                |
| OD94 |                  | 328  | MAGR: ; 11 - MAG STRIPE READER                                |
| OD94 |                  | 329  | WEXS: ; 12 - I/O MEMORY PARITY ERROR                          |
| OD94 |                  | 330  | FCSE: ; 13 - INVALID SPMT SERIES                              |
| OD94 | 994201A5         | 331  | CFPARM1 = ADDR(MSG9998); GET MESSAGE PTR                      |
| OD98 | 08000125         | 332  | CALL CFERCD5M; POST ERROR MSGE TO STATUS LINE                 |
| OD9C | 00000DA0         | 333  | GOTO EOJ; BR IF END OF JOB RESPONSE                           |
|      |                  | 335  | * END OF JOB - NORMAL OR ERROR                                |
| ODA0 | 23510000         | 336  | EOJ: CLOZ (TRANS); CLOSE TRANSACTION DATA SET                 |
| ODA4 | 23520000         | 337  | CLOZ (PRNT); PRINTER ALSO                                     |
| ODAB | C7050000         | 338  | CNENTR; CANCEL POSSIBLE ENTR STMT                             |
| ODAC | 074600B5         | 339  | IFI I110 NOT ON GOTO EOJ1; BR IF IN FOREGROUND PARTITION      |
| ODB0 | 2F000000         | 340  | EXIT; RELEASE BACKGROUND PARTITION                            |
| ODB4 | 08000141         | 341  | EOJ1: CALL CFLOAD01; SHOW LOAD PROMPT                         |
| ODB8 | 00000DB4         | 342  | GOTO EOJ1; REPEAT IF NECESSARY                                |
|      |                  | 344  | FUNC: ; 1 - FUNCTION KEY                                      |
| ODBC | 984600A6         | 345  | WKBRI = SCNCD(2); TEST CONVERTED EBCDIC                       |
| ODC0 | 6A463E09         | 346  | IFH WKBRI EQ X'3E' GOTO ATTN; BR IF ATTENTION KEY             |
| ODC4 | 6A463D0A         | 347  | IFH WKBRI EQ X'3D' GOTO SYSRQ; BR IF SYSTEM REQUEST KEY       |
| ODC8 | 6A463304         | 348  | IFH WKBRI EQ X'33' GOTO SELFMT; BR IF SELECT FORMAT KEY       |
| ODCC | 00000EAB         | 349  | GOTO KRTRN; BR TO IGNORE KEY                                  |
|      |                  | 351  | COMK: ; 2 - UNSHIFTED COMMAND KEY                             |
| ODD0 | 984700A6         | 352  | WKBRI = SCNCD(1); TEST SCAN CODE                              |
| ODD4 | 6A4637F2         | 353  | IFH WKBRI EQ X'37' GOTO EOJ; BR IF CHD EOJ KEY                |
| ODD8 | 00000EAB         | 354  | GOTO KRTRN; BR TO IGNORE KEY                                  |

5280 ASSEMBLER 01.00

SAMPLE2 - ORDER ENTRY SAMPLE PROGRAM;

| ADDR | OBJECT CODE      | LINE | SOURCE CODE  |
|------|------------------|------|--|
| ODDC | 07000EA9         | 356  | SELFMT: IFI EODSW NOT ON GOTO KRTRN; BR TO IGNORE IF UPDATING FILE |
| ODE0 | C7050000         | 357  | CNENTR; CANCEL ENTER   |
| ODE4 | 0C010C80         | 358  | ENABLE (BLNK,POP); BR TO START NEW HEADER                          |
| ODE8 | 08000131         | 360  | ATTN: CALL CFPERATT; PERMIT ANOTHER PARTITION ATTACH               |
| ODEC | 00000EAB         | 361  | GOTO KRTRN;  |
| OF00 | C7050000         | 363  | SYSRQ: CNENTR; CANCEL POSSIBLE ENTER                               |
| OF04 | 0800012D         | 364  | CALL CFATFBGD; ASSURE KB ATTACHED                                  |
| OF08 | 08000141         | 365  | CALL CFLOAD01; PROMPT FOR LOADING BACKGROUND                       |
| OF0C | 00000EAB         | 366  | GOTO KRTRN;  |
| OE00 |                  | 368  | FWRG: ; 4 - FORWARD PASS OVER RG                                   |
| OE00 | 81610126         | 370  | * WKBRI = BR19+0; BR19 CONTAINS BUFFER ADDRESS OF CURRENT FIELD    |
| OE04 | 53024661         | 371  | WKBRI = TBFX(RGRTRN,WKBRI); FIND DATA ADDRESS IN TABLE             |
| OE08 | 08470E14         | 372  | GOTAB WKBRI,RGRTRN; GO TO CORRESPONDING ROUTINE                    |
| OE0C | 08C20BDA091108E6 | 373  | .LABTAB LABEL=BUFAD ENTRY= CUSTM, QUAN, CNTL, AMT;                 |
| OE14 | 0D940E200E3C0E4C | 374  | .LABTAB LABEL=RGRTRN ENTRY= FCSE,CUSTROUT,QUANROUT,QUANCK,PRCROUT; |
| OE20 | 876108C2         | 376  | CUSTROUT:WKBRI = CUSTM; TEST CUSTOMER NO.                          |
| OE24 | 0E610EAB         | 377  | IF WKBRI IS CK GOTO KRTRN; BR IF SELF CHECK IS OK                  |
| OE28 | 994201A0         | 378  | CFPARM1 = ADDR(MSG9111); GET MESSAGE PTR                           |
| OE2C | 08000125         | 379  | CALL CFERCD5M; POST ERROR MSGE TO STATUS LINE                      |
| OE30 | 00000DA0         | 380  | GOTO EOJ; BR IF END OF JOB RESPONSE                                |
| OE34 | C0000001         | 381  | RESUME (B); BACKSPACE TO CUSTOMER NO.                              |
| OE38 | 00000EAB         | 382  | GOTO KRTRN; LET OPERATOR TRY AGAIN                                 |
| OE3C | 847108DA         | 384  | QUANROUT:RQUAN = QUAN; GET QUANTITY IN DEC REG                     |
| OE40 | 67917108         | 385  | IFD RQUAN LE LIMIT GOTO QUANOK; BR IF BELOW MAXIMUM                |
| OE44 | 4002FFFF         | 386  | SON (QUANSW); ENABLE ERROR PROMPT                                  |
| OE48 | 00000EAB         | 387  | GOTO KRTRN; ASK OPERATOR TO VERIFY                                 |
| OE4C | 4ED90911         | 389  | QUANCK: IFC CNTL IS C'R' SKIP; SKIP IF 'RETRY'                     |
| OE50 | 00000E64         | 390  | GOTO QUANOK; BR IF REPLY 'ACCEPT', CONTINUE                        |
| OE54 | 0C010E59         | 391  | ENABLE; RE-ENABLE KEYBOARD INTERRUPTS                              |
| OE58 | C0000001         | 392  | RESUME (B); FIELD BACKSPACE TO QUANTITY                            |
| OE5C | C7115A00         | 393  | KEYOP (X'11',K22);   |
| OE60 | 00000EAB         | 394  | GOTO KRTRN; RECHECK QUANTITY                                       |
| OE64 | 4102FFFF         | 396  | QUANOK: SOFF (QUANSW); BYPASS ERROR PROMPT                         |
| OE68 | 00000EAB         | 397  | GOTO KRTRN; PROMPT FOR PRICE                                       |
| OE6C | 868108DF         | 399  | PRCROUT:RPRICE = PRICE; GET PRICE IN DEC REG                       |
| OE70 | 18617181         | 400  | WKBRI = RQUAN * RPRICE; MULTIPLY PRICE * QUANTITY                  |
| OE74 | 8C6188E6         | 401  | AMT = WKBRI; STORE IN DATA BUFFER                                  |
| OE78 | C5000000         | 402  | REFPLD; REPLACE IN DISPLAY   |
| OE7C | 00000EAB         | 403  | GOTO KRTRN; LET OPERATOR TRY AGAIN                                 |
| OE80 | 4EC40911         | 405  | RCBK: ; 7 - RECORD BACKSPACE                                       |
| OE84 | C7115A00         | 406  | IFC CNTL IS C'D' SKIP; SKIP IF DETAIL RECORD                       |
| OE88 | 2041FF07         | 407  | KEYOP (X'11',K29); ELSE CLEAR HEADER FROM SCREEN                   |
| OE8C | 410001FF         | 408  | READ (TRANS,-); READ PREVIOUS RECORD                               |
| OE90 | 984700BE         | 409  | SOFF (EODSW,FRNTSW); SET UPDATE MODE, STOP PRINTING                |
| OE94 | 9846000F         | 410  | WKBRI = MODESW; GET KB/CRT MODE BYTE                               |
| OE98 | 99460040         | 411  | WKBRI &= X'0F'; CLEAR MODE BITS, KEEP OTHERS                       |
| OE9C | A24700BE         | 412  | MODESW = WKBRI; SET TO UPDATE MODE                                 |
| OEA0 | 93240002         | 413  | BR18 = 2; PUT IT BACK  |
| OEA4 | 00000D3C         | 414  | GOTO SELECT; POP SUBRTN STACK - NO RETURN                          |
| OEAB |                  | 417  | BKRG: ; 5 - BACKWARD PASS OVER RG                                  |
| OEAB |                  | 418  | RCAD: ; 6 - RECORD ADVANCE   |
| OEAB | 079E0E80         | 419  | KRTN: IFI I158 IS ON GOTO KRTRN1; BR IF ATTACHED                   |
| OEAC | 0C010000         | 420  | RETEXT; RETURN, ENABLE   |
| OEBO | CD010000         | 421  | KRTN1: RESHXT; RETURN, ENABLE, RESUME                              |
| 0040 | 000008000C00200  | 423  | .END;  |
| OC48 |                  |      | FIRST INSTRUCTION ADDRESS  |
| OE4E |                  |      | MAXIMUM LOCATION COUNTER   |
| OE4E |                  |      | SUBROUTINE STACK ADDRESS   |
| 005E |                  |      | BOUNDARY ALIGNMENT BYTE LOSS                                       |

5280 ASSEMBLER 01.00 SAMPLE2 - ORDER ENTRY SAMPLE PROGRAM.

| DEFINED | SYMBOL      | COMP | LENG | VALUE | TYPE | REFERENCES  |
|---------|-------------|------|------|-------|------|---|
| 00112   | ANT         | 0000 | 0000 | 08E6  | DATA | 00218 00373 00401   |
| 00360   | ATIN        | 0000 | 0004 | 0DEB  | INST | 00346   |
| 00417   | BNRG        | 0000 | 0004 | 0EAB  | INST | 00321   |
| 00211   | BLANKS      | 0014 | 0018 | 0BE5  | PRMT | 00177 00178 00179   |
| 00258   | BLNK        | 0000 | 0004 | 0C80  | INST | 00248 00358   |
| 00064   | BR10        | 0014 | 0002 | 0114  | BIN  | 00274 00292   |
| 00064   | BR18        | 0024 | 0002 | 0124  | BIN  | 00414   |
| 00064   | BR19        | 0026 | 0002 | 0126  | BIN  | 00370   |
| 00064   | BR22        | 002C | 0002 | 012C  | BIN  | 00307   |
| 00073   | BUF0        | 0046 | 0002 | 014A  | BIN  | 00258 00259 00286 00287   |
| 00373   | BUFAD       | 0000 | 0008 | 0E0C  | DATA | 00120   |
| 00094   | BUFF1       | 0000 | 0400 | 0300  | DATA | 00387   |
| 00095   | BUFF2       | 0000 | 0100 | 0700  | DATA | 00691   |
| 00066   | CFATFEGD *  | 0000 | 0000 | 0120  | INST | 00232 00245 00260 00265 00288 00364                               |
| 00066   | CFDEVCHEK * | 0000 | 0000 | 0139  | INST | 00256 00251   |
| 00066   | CFEREDSM *  | 0000 | 0000 | 0125  | INST | 00335 00374   |
| 00066   | CFGIDERR *  | 0000 | 0000 | 0119  | INST | 00309   |
| 00066   | CFLOAD01 *  | 0000 | 0000 | 0143  | INST | 00341 00365   |
| 00069   | CFPARM1     | 0042 | 0002 | 0182  | BIN  | 00225 00250 00331 00378   |
| 00070   | CFPARM2     | 0044 | 0002 | 0144  | BIN  | 00234 00249   |
| 00066   | CFPERATT *  | 0000 | 0000 | 0131  | INST | 00360   |
| 00066   | CFPRECHK *  | 0000 | 0000 | 0115  | INST | 00056   |
| 00076   | CFSAVE      | 0000 | 0040 | 0800  | DATA | 00048   |
| 00068   | CFWNPTR     | 0040 | 0002 | 0140  | BIN  | 00295   |
| 00113   | CNTL        | 0000 | 0001 | 0911  | DATA | 00302 00302 00373 00389 00406                                     |
| 00351   | CONK        | 0000 | 0004 | 0800  | INST | 00374   |
| 00324   | COM2        | 0000 | 0004 | 0D94  | INST | 00321   |
| 00260   | CUST        | 0000 | 0004 | 0C8B  | INST | 00303   |
| 00103   | CUSTTR      | 0000 | 0008 | 08C2  | DATA | 00222 00373 00376   |
| 00376   | CUSTKOUJ    | 0000 | 0004 | 0E39  | INST | 00374   |
| 00105   | DATE        | 0000 | 0006 | 08B2  | DATA | 00224   |
| 00286   | DETL        | 0000 | 0004 | 0D0C  | INST | 00297 00301   |
| 00205   | DETL1       | 0000 | 0004 | 08A5  | PRMT | 00134   |
| 00205   | DETL2       | 000F | 0008 | 08A9  | PRMT | 00183   |
| 00207   | DETL3       | 0010 | 0018 | 08B1  | PRMT | 00186   |
| 00208   | DETL4       | 0011 | 0011 | 08C9  | PRMT | 00188   |
| 00209   | DETL5       | 0012 | 0005 | 08BA  | PRMT | 00192   |
| 00210   | DETL6       | 0013 | 0006 | 08DF  | PRMT | 00194   |
| 00323   | DEXS        | 0000 | 0004 | 0D94  | INST | 00321   |
| 00318   | DKRTM       | 0000 | 0004 | 0D74  | INST | 00316   |
| 00307   | DSXERR      | 0000 | 0004 | 0D48  | INST | 00087   |
| 00098   | DSNAME      | 0000 | 000A | 08B8  | DATA | 00087 00239   |
| 00214   | DTLFMT      | 0001 | 0003 | 08FE  | FMT  | 00294   |
| 00312   | EDD         | 0000 | 0004 | 0D5C  | INST | 00308   |
| 00060   | EQDSW       | 0000 | 0000 | 0000  | IND  | 00176 00243 00283 00297 00301 00312 00356 00409                   |
| 00336   | EDJ         | 0000 | 0004 | 0DA0  | INST | 00310 00333 00353 00380   |
| 00341   | EDJ1        | 0000 | 0004 | 0DB4  | INST | 00339 00342   |
| 00321   | ERRRB       | 0000 | 001C | 0D7B  | DATA | 00057   |
| 00330   | FCSE        | 0000 | 0004 | 0D94  | INST | 00321 00374   |
| 00344   | FUNC        | 0000 | 0004 | 0D8C  | INST | 00321   |
| 00368   | FURD        | 0000 | 0004 | 0E00  | INST | 00321   |
| 00309   | GIDERR      | 0000 | 0004 | 0D50  | INST | 00091   |
| 00221   | HDRFMT      | 0002 | 0003 | 0C1F  | FMT  | 00281   |
| 00198   | HEADR1      | 0007 | 0008 | 0868  | PRMT | 00155   |
| 00199   | HEADR2      | 0008 | 0005 | 0870  | PRMT | 00157   |
| 00200   | HEADR3      | 0009 | 0004 | 0875  | PRMT | 00159   |
| 00201   | HEADR4      | 000A | 0008 | 0879  | PRMT | 00161   |
| 00202   | HEADR5      | 000B | 0005 | 0881  | PRMT | 00163   |
| 00203   | HEADR6      | 000C | 0009 | 0886  | PRMT | 00165   |
| 00204   | HEADR7      | 000D | 0016 | 088F  | PRMT | 00170   |
| 00327   | INDP        | 0000 | 0004 | 0D94  | INST | 00321   |
| 00101   | IDRUF       | 0000 | 0080 | 08C2  | DATA | 00057 00073 00087 00231 00235 00238 00247 00250                   |
| 00288   | ITEN        | 0000 | 0004 | 0D14  | INST | 00304   |
| 00109   | ITMNG       | 0000 | 0008 | 08B2  | DATA | 00215   |
| 00065   | I115        | 0073 | 0000 | 0000  | IND  | 00277   |
| 00045   | I118        | 0074 | 0000 | 0000  | IND  | 00230   |
| 00065   | I125        | 007D | 0000 | 0000  | IND  | 00264   |
| 00065   | I138        | 009E | 0000 | 0000  | IND  | 00419   |
| 00325   | KERR        | 0000 | 0004 | 0D94  | INST | 00321   |
| 00419   | KRTN        | 0000 | 0004 | 0EAB  | INST | 00349 00354 00356 00361 00366 00377 00382 00387 00394 00397 00403 |
| 00421   | KRTN1       | 0000 | 0004 | 0EAB  | INST | 00419   |
| 00078   | K22         | 0054 | 0002 | 0154  | BIN  | 00392   |
| 00079   | K29         | 0056 | 0002 | 0156  | BIN  | 00407   |
| 00080   | K3701       | 0058 | 0002 | 0158  | BIN  | 00308   |
| 00077   | K6          | 0052 | 0002 | 0152  | BIN  | 00271   |
| 00082   | LIMIT       | 0091 | 0010 | 0190  | DEC  | 00389   |
| 00328   | MAGR        | 0000 | 0004 | 0D94  | INST | 00321   |

5280 ASSEMBLER 01.00 SAMPLE2 - ORDER ENTRY SAMPLE PROGRAM.

| DEFINED | SYMBOL   | COMP | LENG | VALUE | TYPE | REFERENCES  |
|---------|----------|------|------|-------|------|---|
| 00106   | HANND    | 0000 | 0002 | 08DB  | DATA | 00225   |
| 00059   | MODESW   | 0000 | 0001 | 00BE  | DATA | 00313 00315 00410 00413   |
| 00083   | MS09111  | 0000 | 0005 | 01A0  | DATA | 00378   |
| 00084   | MS0999B  | 0000 | 0005 | 01A5  | DATA | 00331   |
| 00326   | NEXS     | 0000 | 0004 | 0D94  | INST | 00321   |
| 00245   | OPNFR    | 0000 | 0004 | 0C84  | INST | 00252 00254   |
| 00232   | OPNTRN   | 0000 | 0004 | 0C54  | INST | 00230 00231 00241   |
| 00104   | ORND     | 0000 | 0008 | 08CA  | DATA | 00223   |
| 00107   | POSTO    | 0000 | 000A | 08DA  | DATA | 00226   |
| 00097   | PRBUF    | 0000 | 0078 | 0840  | DATA | 00091   |
| 00399   | PREEROUT | 0000 | 0004 | 0E6C  | INST | 00374   |
| 00074   | PRCNTL   | 0050 | 0002 | 0150  | BIN  | 00075 00276 00278 00280   |
| 00075   | PRCNTLE  | 004E | 0002 | 014E  | BIN  | 00279 00281   |
| 00175   | PRBELT   | 0004 | 0003 | 0AF3  | SFMT | 00289   |
| 00154   | PRBRD    | 0002 | 0003 | 0A92  | SFMT | 00281   |
| 00111   | PRICE    | 0000 | 0007 | 08DF  | DATA | 00217 00399   |
| 00150   | PRINT1   | 0005 | 001C | 0A58  | PRMT | 00140   |
| 00151   | PRINT2   | 0006 | 001B | 0A77  | PRMT | 00142   |
| 00061   | PRNT     | 0000 | 0002 | 0002  | SFT  | 00091 00249 00253 00279 00281 00294 00316 00337                               |
| 00060   | FRNTSW   | 0001 | 0000 | 0000  | IND  | 00255 00275 00293 00317 00409   |
| 00138   | PRPRINT  | 0001 | 0003 | 09CF  | SFMT | 00246   |
| 00169   | PRSHIP   | 0003 | 0003 | 0A8C  | SFMT | 00266   |
| 00279   | FRTHD    | 0000 | 0004 | 0CFB  | INST | 00277   |
| 00129   | PRTRANS  | 0000 | 0003 | 09A9  | SFMT | 00233   |
| 00110   | QUAN     | 0000 | 0005 | 08DA  | DATA | 00216 00373 00384   |
| 00389   | QUANCK   | 0000 | 0004 | 0E4C  | INST | 00374   |
| 00396   | QUANCK   | 0000 | 0004 | 0E44  | INST | 00385 00390   |
| 00384   | QUANROUT | 0000 | 0004 | 0E3C  | INST | 00374   |
| 00060   | QUANSW   | 0002 | 0000 | 0000  | IND  | 00185 00386 00396   |
| 00418   | RCAD     | 0000 | 0004 | 0EAB  | INST | 00321   |
| 00405   | RCBK     | 0000 | 0004 | 0E80  | INST | 00321   |
| 00300   | READ     | 0000 | 0004 | 0D34  | INST | 00274 00283 00292   |
| 00374   | RGROUT   | 0000 | 000A | 0E14  | DATA | 00372   |
| 00120   | RGRTN    | 0002 | 0008 | 097E  | TBL  | 00371   |
| 00081   | RFRICE   | 0001 | 0010 | 0180  | DEC  | 00399 00400   |
| 00081   | RQUAN    | 0071 | 0010 | 0170  | DEC  | 00394 00395 00400   |
| 00072   | RWCOL    | 0048 | 0002 | 0148  | BIN  | 00271   |
| 00058   | SCNCD    | 0000 | 0001 | 00A6  | DATA | 00345 00352   |
| 00302   | SELECT   | 0000 | 0004 | 0D3C  | INST | 00415   |
| 00356   | SELFMT   | 0000 | 0004 | 0D0C  | INST | 00348   |
| 00262   | SHIP     | 0000 | 0004 | 0C00  | INST | 00267   |
| 00121   | SHIPCODE | 0000 | 0023 | 0986  | DATA | 00118 00119   |
| 00119   | SHIFID   | 0001 | 0008 | 0976  | TBL  | 00263   |
| 00269   | SHIPOK   | 0000 | 0004 | 0C8B  | INST | 00264   |
| 00118   | SHIPTXT  | 0000 | 0008 | 096E  | TBL  | 00269   |
| 00108   | SHIPCD   | 0000 | 0007 | 08E4  | DATA | 00074 00227 00262 00270   |
| 00229   | START    | 0000 | 0004 | 0C4B  | INST | 00556   |
| 00263   | ZYSRQ    | 0000 | 0004 | 0DFO  | INST | 00347   |
| 00145   | TITLE    | 0001 | 001A | 09ED  | PRMT | 00130 00139   |
| 00061   | TRANS    | 0000 | 0002 | 0001  | SFT  | 00087 00229 00234 00240 00241 00242 00254 00282 00296 00300 00336 00408       |
| 00146   | TRANS1   | 0002 | 0039 | 0A07  | PRMT | 00131   |
| 00148   | TRANS2   | 0003 | 000D | 0A40  | PRMT | 00132   |
| 00149   | TRANS3   | 0004 | 000E | 0A4D  | PRMT | 00134   |
| 00074   | TX18     | 004C | 0002 | 014C  | BIN  | 00271   |
| 00329   | WEXS     | 0000 | 0004 | 0D94  | INST | 00321   |
| 00071   | WBR1     | 0046 | 0002 | 0146  | BIN  | 00229 00231 00263 00269 00307 00308 00313 00314 00315 00345 00346 00347 00348 |
| 00091   | WDR1     | 0061 | 0010 | 0160  | DEC  | 00352 00353 00371 00372 00410 00411 00412 00413                               |
| 00295   | WTD1     | 0000 | 0004 | 0D28  | INST | 00293   |
| 00282   | WRTHD    | 0000 | 0004 | 0D04  | INST | 00275   |

The following is a sample of output:

|          |          |          |    |           |       |           |     |         |              |
|----------|----------|----------|----|-----------|-------|-----------|-----|---------|--------------|
| 00000026 | 12345678 | 12/25/79 | 45 | AUSTIN    | TRUCK | 00054321  | 15  | \$1.25  | *****18.75   |
|          |          |          |    |           |       | 000667893 | 75  | \$3.24  | *****243.00  |
|          |          |          |    |           |       | 00053241  | 115 | \$10.75 | *****1236.25 |
| 00001834 | 00085326 | 12/25/79 | 02 | ROCHESTER | AIR   | 00030256  | 63  | \$1.98  | *****124.74  |
|          |          |          |    |           |       | 00073325  | 14  | \$9.53  | *****133.42  |

**This page is intentionally left blank**

**access method:** A technique for moving data between main storage and input/output device.

**active data set:** A data set being used by a program.

**address:** A name, label, or number that identifies a register, location in storage, or any other data source.

**alphabetic characters:** Letters and other symbols, excluding digits, used in a language.

**alphabetic field:** One or more alphabetic characters of related information in a record.

**alphabetic shift:** A control (attribute or key) for selecting the alphabetic character set in an alphameric keyboard.

**alphameric characters:** Same as alphabetic characters, with the addition of digits 0 through 9.

**alphameric field:** One or more alphameric characters of related information in a record.

**arithmetic expression:** An expression that contains arithmetic operations and that can be reduced to a single numeric value. An arithmetic expression is evaluated from left to right with multiplication and division preceding addition and subtraction.

**alternate record advance:** A function that causes the system to stop processing the current record and ignore any specifications between the cursor position and the end of the record when the Enter or Record Advance key is pressed.

**application program:** A program that processes user data to perform a particular data processing task; for example, inventory control or payroll.

**ASCII:** (ANSI definition) American National Code for Information Interchange. The standard code, using a coded character set consisting of 7-bit coded characters (8 bits including parity check), used for information interchange among data processing systems, data communication systems, and associated equipment. The ASCII set consists of control characters and graphic characters.

**assembler:** A computer program that prepares an object program from a source program written in a symbolic source language.

**assembler language:** A source language that includes symbolic machine language statements in which there is a one-to-one correspondence with the instruction formats and data formats of the computer.

**attribute:** A characteristic. For example, attributes of a data set include record length, label, and creation date. Attributes of a displayed field could include high intensity, reverse image, and column separators.

**auto dup:** Automatic duplication. 1. The process of automatically copying the contents of a field in a previous record or a storage area into the corresponding positions of the current record. 2. The process of automatically verifying the contents of a field in the current record with the contents of the corresponding positions of a previous record or a storage area.

**auto record advance:** Automatic record advance. A movement forward to the next sequential record without manual intervention when current record is completely entered and the auto rec adv switch is on.

**auto skip:** Automatic skip. In enter mode, if the auto skip/dup switch is on, the process of automatically filling an auto skip field with blanks and advancing to the next field. In verify mode, the process of verifying that all the positions in the field are blank.

**auto verify:** Automatic verify. In verify mode, auto dup fields are checked against the same fields in the previous record. See *auto dup*, 2.

**auxiliary duplication:** The process of copying or verifying data from a named storage location into a field.

**awaiting field exit:** The state of the keyboard when the operator has entered the last position of a field that is defined as a field exit required field.

**awaiting record advance:** The state of the keyboard when the operator has entered the last position of a record with a key other than the Record advance key, and the Auto-enter function is not enabled.

**background job:** A job that is run in a partition which does not have immediate access to a keyboard/display unit.

**base displacement addressing:** An addressing method that involves setting up a base address from which other addresses can be calculated.

**basic data exchange:** A diskette data exchange that uses 128-byte sectors and allows only one record per sector. The logical record length must be  $\leq 128$  bytes, and is unblocked and unspanned. The basic data exchange formats allow you to exchange data between 5280 and other systems that use the basic data exchange format.

**binary:** Base 2 arithmetic.

**binary register:** A two-byte register in partition storage which contains binary notation and is used for binary arithmetic/logical operations.

**binary search:** At each step of the search the set of items is partitioned into two equal parts so that the search starts at the middle.

**blank check:** A check of a field to ensure that there are no blank characters (hex 40) in the field.

**blank fill:** To fill a field with blank characters (hex 40).

**blocking:** Combining two or more records into one block.

**boundary alignment:** The positioning of data areas such as registers or blocks, on an appropriate boundary for that type of data.

**branch instruction:** An instruction that changes the sequence in which the instructions in a computer program are executed. Execution of instructions continues at the address specified in the branch instruction.

**BSC:** Binary synchronous communications.

**buffer:** An area of storage that is temporarily reserved for use in performing an input/output operation, into which data is read or from which data is written.

**CAM:** Communications access method.

**character constant:** Any combination of characters, including blanks, enclosed in apostrophes.

**collating sequence:** The order each character holds in relation to other characters according to the bit structure.

**column separators:** A display screen attribute that shows vertical lines preceding each position of a field on a display. These lines do not occupy positions on the display. For example  $\perp A \perp B \perp C$ .

**command function keys:** The 14 keys on the top row of the data station keyboard that are used with the command key to request functions.

**comments:** Words or statements in a program that serve as documentation rather than instructions to an assembler or compiler.

**common area:** The first part of main storage that contains the system control area, common functions, global tables (such as ASCII and error recording), and so on. Depending upon the common function option selected, this area can be 6 K, 14 K, or 16 K. This area is not available for user programs.

**common functions:** A set of IBM-supplied programs in the common area that is used by programs executing in any partition.

**communications access method (CAM):** An IBM 5280 program that provides the necessary link between a communications program and the communication line. It performs functions such as data formatting and link protocol.

**concurrent:** (ANSI Definition) Pertaining to the occurrence of two or more activities within a given interval of time.

**configuration:** The group of machines, devices, features, and programs that make up a data processing system.

**constant:** A data item that does not change during the execution of a program. This item represents itself and is actually used in processing rather than being a field name representing the data. For example, 'cost' is a name representing a field containing data that changes. The constant 100 is actual data used that does not change.

**control block:** A storage area used by a program to hold control information.

**controller:** A device that controls operation of one or more input/output devices; for example, a work station controller.

**copy:** To read data from a source, leaving the source data unchanged, and to write the same data elsewhere in a physical form that may differ from that of the source.

**counter:** A register or storage location used to accumulate the number of occurrences of an event.

**cursor:** A movable horizontal line (underscore) on a display screen, used to indicate where the next character entered by the operator will appear. It blinks when no additional entry is allowed and the system is awaiting the Enter key.

**cylinder:** The tracks that can be accessed without repositioning the diskette drive access mechanism.

**data-directed format selection:** Format selection is determined by the data contained in the record.

**data required:** A field attribute that indicates an operator must enter at least one non-blank character into the displayed field.

**data set:** An organized collection of related data records treated as a unit and existing on a diskette.

**data set label:** A 128-byte area on the diskette index cylinder that describes a data set.

**data set name:** The name associated with a data set. The first character must be alphabetic, and the remaining characters can be any combination of alphabetic or numeric characters. Blanks cannot appear between characters in a name.

**data stream:** Data transferred by stream-oriented transmission, as a continuous stream of data elements in character form.

**data table:** A table defined by the .TABLE control statement.

**decimal register:** A 16-byte register wherein data is stored in EBCDIC or signed decimal numbers and is used for arithmetic/logical operations.

**default value:** A value automatically chosen by the system when a value is not specified by the user.

**DE/RPG:** Data Entry with RPG subroutines. An IBM 5280 program product that provides a means for writing programs that provide the function required for a specific job.

**device address:** Four hex characters used to identify an IBM 5280 I/O device such as a diskette drive or printer.

**diacritic:** A modifying mark that changes the phonetic value of a character. When you enter a diacritic from the keyboard, the cursor does not advance until another character is entered to combine with the diacritic.

**diacritic table:** A table that defines diacritic characters and valid diacritic-character composites for graphic display.

**direct access:** The ability to obtain data from a storage device directly by key or relative record. Contrast with *sequential access*.

**direct access method:** An access method for processing files by specifying the address (record number) or key value of each record to be accessed.

**direct addressing:** A method of addressing in which the addressed storage location contains the desired data. See also *indirect addressing*.

**direct by key access method:** An access method for processing index data files by specifying the key associated with each record to be accessed. The current key specified need not have any relative sequence with the last key or next key to be specified.

**displacement:** The number of bytes from the beginning of a partition or block to the beginning of a particular data area.

**display attributes:** The characteristics assigned to a field record that control the way the data is displayed.

**double register:** Two decimal or binary registers used together as one data area. In a program, the leftmost register is referenced, followed by the length in parentheses (4 for binary, 32 for decimal).

**dup:** Abbreviation for duplicate.

**EBCDIC (extended binary-coded decimal interchange code):** A character set containing 256 eight-bit characters.

**edit format:** A description of a record that is read from a diskette, written to a diskette or printer, or moved from one storage location to another. An edit format is set up by a FMT series of control statements, and defines the fields, punctuation, data types, and other editing requirements of the record.

**ELAB/ETAB:** Parameter in the COMM and DATASET statements which specifies the name of a routine (ELAB) or table (ETAB) to be used to handle error conditions.

**enter mode:** The mode in which the operator initially enters data through a display station. Some editing and interaction may occur. See also *verify mode*; *update mode*.

**E-type data exchange:** A diskette data exchange format that uses blocked and spanned, blocked and unspanned, or unblocked and unspanned records. Block size can be up to 16,256 bytes.

**extent:** A continuous space on a diskette that is occupied by or reserved for a particular data set.

**extra line:** Row 1 of the screen refresh buffer, which can be displayed on the top row of the screen in place of the status line.

**field:** One or more bytes of related information in a record.

**field attribute:** See *attribute*.

**foreground job:** The keyboard/display unit is immediately available to the partition where the job is being executed.

**format level:** The identification associated with a format.

**format 0 (zero):** A format for display stations that allows entering information on an unformatted display.

**global load:** A load operation that uses the standard load prompt. A global load is initiated by the system when the load parameters are not specified for a LOAD instruction in an assembler program, and when an error occurs when using the Standard Load Processor.

**global table:** A table in the common area. The first two global tables are the error recording tables. If the ASCII translate table is selected during system configuration, the ASCII translate table is another global table.

**hex:** Hexadecimal. A number system using 16 symbols: 0-9, A-F each representing 4 bits (one-half byte).

**H-type data exchange:** A diskette data exchange format that uses 256-byte sectors. It allows only one record per sector. The logical record length must be 256 bytes; it is unblocked and unspanned. The H-type exchange allows you to exchange data between 5280 and other systems that use the H-type data exchange format.

**index data set:** A data set in which the keys from another data set and their record position within that data set are recorded. When index data sets are used, the following access methods can be used: sequential; direct by relative record number; and direct by key value.

**index register:** A register whose contents can be added to or subtracted from the operand address before or during execution of a computer instruction.

**indexed address:** An address that is modified by the content of an index register before or during the execution of an instruction.

**indexed instruction:** An instruction that requires address modification before the data byte is fetched from storage.

**indirect addressing:** A method of addressing in which the addressed storage location contains the address of the desired data. See also *direct addressing*.

**initial program load (IPL):** A sequence of events that loads the system programs and prepares the system for execution of jobs.

**input data set:** A set of records a program uses as source information.

**input/output control block (IOB):** A data area that may be used to pass the required information from the calling program to the input/output supervisor for data operations.

**input record:** A data record that is transferred to computer storage for processing.

**insert field:** A field not present in the enter record, but which will be inserted by the system and will be present in the output record.

**instruction:** A statement that specifies an operation to be performed by the computer and the locations in storage of all data involved in that operation.

**IOB:** Input/output control block.

**IOB pointer:** A 4-byte block in the system control area that contains the address of a device IOB and other information (such as, if the device is installed).

**IPL:** Initial program load.

**I-type data exchange:** A diskette data exchange format that uses 128, 256, 512, or 1024 byte sectors. All records in a data set must be the same length. All records in the data set are blocked and spanned. The I-type exchange allows you to exchange data between the 5280 and other systems that use the I-type data exchange.

**keyboard/display storage:** An area of control storage separate from amin storage, which provides control information and refresh areas for processing keystrokes and for displaying characters on the screen.

**label table:** A table of addresses set up by the .LABTAB control statement, and used for indexed branches and indexed subroutine calls.

**logical record:** A record independent of its physical environment. Portions of the same logical record may be located in different physical records, or several logical records or parts of logical records may be located in one physical record, depending on the exchange type being used.

**Magnetic Stripe Reader feature:** Allows use of the 5280 system only after a valid badge (operator ID) is read by an attached magnetic stripe reader.

**main storage:** 1. General purpose storage of a computer. 2. Storage that can be addressed by programs, from which instructions can be executed, and from which data can be loaded directly into registers.

**main storage duplication field:** See *auxiliary duplication*.

**main storage store field:** A field that is automatically stored from the current record buffer into a main storage location.

**mandatory enter:** A field attribute that indicates an operator must enter at least one character into the displayed field.

**mandatory fill:** A field attribute that indicates an operator must enter all or none of the displayed field.

**mask:** A pattern of characters that is used to control the retention or elimination of another group of characters.

**MPU:** The main microprocessor.

**multinational character set:** The 188-character (or 184 character) display and printer character set available with the 5280.

**multivolume data set:** A data set that extends beyond the boundaries of a single data set. It can be extended on the same diskette or on another diskette.

**nondisplay:** A field attribute that prevents display of data. It can be used for fields containing confidential information.

**null character:** The hexadecimal character 00.

**numeric fields:** A field that contains one or more numeric characters. Valid numeric characters are the digits 0-9 and + (plus sign), - (minus sign), . (decimal point), blank, and , (comma).

**numeric shift:** A control (attribute or key) for selecting the numeric character set in an alphameric keyboard.

**object code:** The four-byte instructions from the compiler or assembler that are machine executable. The first byte of the object code contains the operation code.

**object program:** A set of instructions in machine language (object code). The object program is produced by the assembler from the source program.

**offset:** The distance from the beginning of a register or record to the beginning of a particular field.

**output data set:** A data set containing the data that results from processing.

**packed data field:** One byte is used to store two numeric digits. Bits 0 through 3 for one digit and bits 4 through 7 for the other.

**packed decimal format:** Each byte within a field represents two numeric digits except the rightmost byte, which contains one digit in bits 0 through 3 and the sign in bits 4 through 7. For all other bytes, bits 0 through 3 represent one digit; bits 4 through 7 represent one digit. For example, the decimal value +123 is represented as 0001 0010 0011 1111. Contrast with zoned decimal format.

**pad:** To fill unused positions in a field with dummy data, usually zeros or blanks.

**partition:** An area of 5280 storage in which a program can execute.

**partition pointer:** Contains the address of the beginning of a partition. The partition pointer also contains flags to indicate the status of the partition (such as whether the partition is a foreground or background partition).

**physical record:** A record whose characteristics depend on the manner or form in which it is stored, retrieved, or moved. A physical record may consist of all or part of a logical record.

**program listing:** A computer printout that gives information about the source program, such as source statements, diagnostic messages, indicators used, storage addresses of fields and constants used.

**program product:** An IBM-written, licensed program for which a monthly charge is made. A program product performs functions related to processing user data.

**reformatting:** The rearrangement of an addition or elimination of fields in a record.

**refresh:** The continuous redisplaying of data on the display screen to prevent the data from fading out.

**refresh buffers:** Areas in keyboard/display storage that are used to refresh each row of display characters on the screen. The refresh area for the status line is in an area separate from the refresh area for the other rows on the screen.

**relative addressing:** A means of addressing instructions and data areas by designating their location in relation to the location counter or to some symbolic symbol. Relative addresses of areas within a partition are relative to the beginning of the partition.

**relative record number:** A number that specifies the location of a record in relation to the beginning of the data set.

**resource allocation table:** A table in storage that is used to assign a logical device ID (a name) to a physical device.

**return-to-program exit:** See *RG exit*.

**RG exit:** A user exit that interrupts the processing of a screen format to give control to a user's routine.

**SCP:** See *system control program*.

**screen format:** A description of a record that is entered via the keyboard/display. A screen format is set up by a SFMT series of control statements, and defines the fields, prompts, control specifications, and display attributes of the record.

**screen format control string:** The object code that is generated by a series of SFMT control statements.

**SCS conversion data set:** A data set that has SCS conversion specified in the .DATASET control statement that defined the data set. The system automatically inserts SCS control characters into an SCS conversion data set to create an SCS data set.

**SCS data set:** A data set that contains SCS control characters. Contrast with *SCS conversion data set* which contains no SCS control characters.

**SDLC:** Synchronous data link control.

**SDT:** Self-defining term.

**self-check field:** A field, such as an account number, consisting of a base number and a self-check digit. For data entry applications, the self-check digit entered by the operator is compared to the self-check digit computed by the system. If the operator makes a mistake when entering (keying) a self-check field, an error message is displayed.

**self-defining term:** An absolute decimal, binary, hexadecimal, or character data value or a label that has been equated to such a value. Labels that imply relative addressing are not self-defining terms.

**sequential access method:** An access method in which records are accessed in the order in which they occur in the file. Contrast with *direct access method*.

**sequential by key:** A method of data set processing that accesses records in the order in which a keyed or indexed data set is arranged.

**SNA:** Systems network architecture.

**source program:** A set of instructions that represents a particular job as defined by the programmer. These instructions are written in a programming language, such as GSLE.

**spanned record:** 1. A record that crosses a block boundary. 2. A record that is stored in more than one block.

**stack pointer:** The binary register (BR18) used for subroutine calls and returns. During a subroutine call, the stack pointer contains the address of the next available entry in the subroutine stack; during a subroutine return, it contains the address of the last entry in the subroutine stack.

**standard load prompt:** The screen format stored in the common area that is used to prompt for load parameters during a global load or by the Standard Load Processor.

**status line:** Usually, the first line on a display screen. This line provides operational information.

**stripped zone:** See *packed data field*.

**subroutine stack:** A table of return addresses used for subroutine returns.

**Synchronous data link control (SDLC):** A discipline for managing synchronous, transparent, serial-by-bit information transfer over a communications line.

**syntax:** (ANSI definition) 1. The structure of expressions in a language. 2. The rules governing the structure of a language.

**system configuration:** A process that specifies the various components and devices that form a particular operating system. System configuration combines user-specified options and parameters with IBM programs to produce a system having the desired form and capacity.

**system control programs:** IBM-supplied programs that are on a diskette. These programs are included with each system and allows the operator to configure the system, IPL the system, and recover from power failures.

**system control area:** 256 bytes starting at address X'00'. This area contains information such as the address of each partition, device IOB pointers, system flags, machine storage size, and so on.

**system table:** A table set up and used by the system to store the addresses of screen formats, edit formats, prompts, data tables, and duplicate or store fields.

**system network architecture (SNA):** A total description of logical structure, formats, protocols, and operation sequences for transmitting information throughout a communications network.

**timeout:** A time interval during which a station waits for a certain operation to occur. Some timeouts are automatic hardware functions and some are program functions.

**update mode:** The mode in which the operator selects certain records for review and correction. See also *enter mode*; *verify mode*.

**verify:** To determine whether a transcription of data or other operation has been accomplished accurately.

**verify bypass field:** A field that was entered, but does not need to be verified.

**verify mode:** The mode in which the operator rekeys data from a source document that has already been keyed in order to check that the data has been entered correctly. See also *enter mode*; *update mode*.

**zero fill:** To fill with the numeric value zero.

**zero suppress:** The elimination of preceding zeros in a number. For example, 0057 becomes 57 when zero suppressed.

**zoned decimal format:** Representation of a decimal value by 1 byte per digit. Bits 0 through 3 of the rightmost byte represent the sign; bits 0 through 3 of all other bytes represent the zone portion; bits 4 through 7 of all bytes represent the numeric portion. For example, the decimal value +123 is represented as 1111 0001 1111 0010 1111 0011. Contrast with packed decimal format.

**zoned field:** A field that contains data in the zoned decimal format.

This page is intentionally left blank

- + 143
- + = 136
- & = 138
- \* 144
- \* = 137
- 143
- = 137
- / 144
- / = 137
  
- absolutely automatic field 109
- accept key entry (ENTR) 200
- accept unformatted keystrokes (KACCPT) 207
- access methods 185
- ACL conversion 317
- add, binary 136
- add, decimal 143
- ADDR function 60, 133
- addressing methods
  - base displacement addressing 12, 125
  - direct addressing 11, 125
- AFIL (.KBCRT parameter) 54
- algorithm, self-check 43
- ALLOC 175
- allocate 175
- allow detach routine (CFDETFGD) 290
- alphabetic fill character 54
- alphabetic only field 108
- alphabetic shift field 107
- alternate record advance 29
- alternate register 119
- ALTREG (.SELFCHK parameter) 119
- AND instruction 165
- AND, binary 138
- application control language 317
- ARGL (.TABLE parameter) 79
- arithmetic expressions 133
- arithmetic instructions, binary 135
- arithmetic instructions, decimal 141
- ASCII processor 289
- ASCII translate table 268
- assembler diskette 271
- assembler error messages 377
- assembly listing 113, 278
- assembly time arithmetic 132
- attach a partition 208
- attach partition routine (CFATFBGD) 289
- auto dup field 109
- auto dup skip field 29
- auto skip field 109
- automatic functions 28, 109
- automatic logical buffering 36
- awaiting record advance 29
  
- background partition 208
- badge reader characters 210
- base displacement addressing 12, 125
- base displacement skip on constant compare 164
- BDY (.DC parameter) 59
- beginning of extent 174
- BFPS (.SFMT series parameter) 95
- BFPS (.SFMTCNS parameter) 102
- BFPS (.SFMTCTL parameter) 99
- BFPS (.SFMTFLD parameter) 105
- binary arithmetic 135
- binary register relational compare 159
- binary registers 9
  - declare a register 65
  - double binary registers 9
  - load 229
  - store 234
  - system registers 9, 284
- binary search, data set 190
- binary search, table 258
- BINDEC 238
- BINHEX 238
- bit numbers for keyboard functions 371
- BKCK (.KBCRT parameter) 54
- blank a decimal register 146
- blank check character 54
- blank check field 110
- blanks 14
  - in a control statement 51
  - in an instruction 123
- block size 71, 76, 176
- boundaries 13
- boundary alignment 13, 59
- braces 15
- brackets 15
- branch instructions 149
- branch on relational compare instructions 158
- branch on test instructions 154
- branch through table 150
- BSIZ 176
- BSIZ (.COMM parameter) 71
- BSIZ (.DATASET parameter) 76
- buffer position pointer 92, 95
- buffers 36
- buzz 106
- BUZZ 205
- BYPAS (.TABLE parameter) 80
- bypass 80
- bypass field 33, 95, 110
- bypass in verify mode field 110
  
- CALL 151
- call through table 153

CALLTB 153  
 CAM (.COMM parameter) 70  
 cancel current ENTR 206  
 category digits 24  
 CCHAR (.FMTST parameter) 85  
 CCOL (.FMTST parameter) 85  
 CFASCII 289  
 CFATFBGD 289  
 CFCOMDSP 291  
 CFDETFGD 290  
 CFDEVCHK 291  
 CFDUMPTR 292  
 CFEOJRTN 294  
 CFERCDSM 295  
 CFERCDSP 294  
 CFGIOERR 296  
 CFKEYRT 296  
 CFLOAD01 301  
 CFMSGDSP 297  
 CFPERATT 298  
 CFPGMCHK 299  
 CFSECVOL 300  
 change cursor address 214  
 change data type 134  
 change row attribute 213  
 check/move device address routine (CFDEVCHK) 291  
 CI (.SFMTCTL parameter) 98  
 CLC 262  
 clear screen 94, 106  
 CLICK 205  
 close a data set  
     communications 172  
     diskette 178  
     printer 194  
 CLOZ 178, 194  
 CNENTR 206  
 CNST (.SFMTCNS parameter) 102  
 CNTL (.SELFCHK parameter) 117  
 CNTL (.SFMT series parameter) 93  
 CNTL (.SFMTCNS parameter) 102  
 CNTL (.SFMTCTL parameter) 98  
 CNTL (.SFMTEND parameter) 106  
 CNTL (.SFMTFLD parameter) 104  
 CNTL (.SFMTST parameter) 97  
 CNTR (.KBCRT parameter) 55  
 coding symbols 14  
 COL (.FMTFLD parameter) 86  
 comments  
     on control statements 51  
     on instructions 124  
 common function labels 275  
 common function routines 286  
 common functions 120  
 communications 69  
 communications access method (CAM) 69  
 communications control 170  
 communications instructions 168  
 compare logical character strings 262  
 compression 75  
 condition character, data directed formatting 85  
 condition code format 24  
 condition number, external status 24  
 conditional branch instructions 154, 158  
 conditional bypass 33, 98  
 constant insert field 31, 102  
 constant, store 236  
 constants 14, 125  
 control statement summary 49  
 conversion of data, in edit format 86  
 conversion program errors 379  
 convert data 237  
 CRBR (.KBCRT parameter) 53  
 cross-reference listing 280  
 CRTMM 244  
 CSPS (.SFMT series parameter) 95  
 CSPS (.SFMTCNS parameter) 102  
 CSPS (.SFMTCTL parameter) 99  
 CSPS (.SFMTFLD parameter) 105  
 current position pointer 22, 55  
 current record buffer 53, 200  
 current record pointer 181  
 data directed formatting 35  
 data entry, keyboard 25  
 data movement 229  
 data required field 110  
 data set indicator 157  
 data set label 174  
 data set name 73, 176, 251  
 data set number 70  
 data set type 70, 73  
 data tables 18  
 data types 19, 134  
     change data type 134  
 DCLBL (.FMTFLD parameter) 86  
 DCLBL (.TABLE parameter) 79  
 DECBIN 240  
 decimal register arithmetic 141  
 decimal register relational compare 160  
 decimal register shift instructions 145  
 decimal register zone modification 148  
 decimal registers 10  
     declare a register 65  
     double decimal registers 10  
     load 231  
     store 234  
 declare binary registers 65  
 declare control statements 57  
 declare decimal registers 65  
 declare indicators 65  
 DECR 168  
 decrement binary register and branch 168  
 delete a data set 179  
 delete a record 186  
 delete a table entry 255  
 delete character 176  
 delete flag 77  
 delta 78  
 density of index tables 39  
 detach keyboard 209  
 detach partition routine 298  
 DEV (.DATASET parameter) 73  
 device address 73, 291

device identifier 73  
 DEVID (.DATASET parameter) 73  
 DFLG 176  
 DFLG (.DATASET parameter) 76  
 digits only field 107  
 direct addressing 11, 125  
 directed close, for printer 198  
 diskette control operations 174  
 diskette index 177  
 diskette initialization 74, 181  
 DISP (.DC parameter) 59  
 DISP (.SELFCHK parameter) 119  
 DISPEX 206  
 displacement 11  
 display attributes 94  
 display extra line 206  
 display mode 28  
 display status line 206  
 DISPST 206  
 divide, binary 137  
 divide, decimal 144  
 DL, self-check 361  
 DLTA (.DATASET parameter) 78  
 double buffering 36  
 DR, self-check 361  
 DSN (.COMM parameter) 70  
 DSN (.DATASET parameter) 73  
 DSPLY 105  
 DSPLY (.SFMT series parameter) 94  
 DSPLY (.SFMTCNS parameter) 102  
 DSPLY (.SFMTCTL parameter) 99  
 dump/trace processor (CFDUMPTR) 292  
 DUP 265  
 dup field 31, 104  
 dup key status 93  
 duplicate a byte 265

early write diskette 75  
 early write printers 184  
 EBCDIC charts 369  
 EDCNT (.EDITC parameter) 57  
 EDCOM (.EDITC parameter) 57  
 EDCUR (.EDITC parameter) 56  
 EDDEC (.EDITC parameter) 56  
 EDIT (.FMTFLD parameter) 86  
 edit control characters 56  
 edit control count 57  
 edit currency symbol 56  
 edit decimal character 56  
 edit format 34, 81, 183, 246  
 edit format label 85  
 edit format system table 83  
 edit separator character 56  
 edit string characters 86  
 ELAB (.COMM parameter) 70  
 ELAB (.DATASET parameter) 73  
 ELAB (.KBCRT parameter) 53  
 elapsed time counter 228  
 ENABLE 202  
 end of data 174  
 end of extent 174  
 enter mode 26

ENTR 200  
 ENTRIES (.TABLE parameter) 80  
 ENTRY (.LABTAB parameter) 81  
 ENTRY (.START parameter) 52  
 equate 66  
 erase a data set 179  
 erase data set 75  
 error code display routine (CFERC DSP) 294  
 error code with message display (CFERCDSM) 295  
 error label 53, 70, 73  
 error messages on listing 281  
 error messages, assembler 377  
 error messages, conversion program 379  
 error mode 209  
 error recovery during load 249, 252  
 error recovery, printers 197  
 error table 53, 70, 73  
 errors only listing 273  
 ES (.SFMTCTL parameter) 99  
 ETAB (.COMM parameter) 70  
 ETAB (.DATASET parameter) 73  
 ETAB (.KBCRT parameter) 53  
 exchange data 236  
 exchange type 176  
 exclusive OR write, skip on AND mask 166  
 exclusive OR, binary 139  
 EXIT 250  
 EXPR (.EQUATE parameter) 68  
 expression, equate 68  
 expressions 132  
 extern data set 275  
 external status 24  
     condition code format 24  
     system binary registers 25  
 external status subroutines for keyboard/display 304  
 external status, keyboard/display 302  
 extra line 206

field attribute chart 113  
 field correct mode 28  
 field definition, edit format 86  
 field definition, screen format 103  
 field exit minus key status 94  
 field exit required field 110  
 field modification indicators 35  
 field type 103, 107  
 fixed position prompt 101  
 fixed prompt location 54  
 FLDF (.SFMTFLD parameter) 103  
 FLDLEN (.SELFCHK parameter) 115  
 FMT (.SYSTAB parameter) 83  
 foreground partition 208  
 format level zero field 107  
 format, of control statements 51  
 format, of instructions 123  
 FPLC (.KBCRT parameter) 54  
 full data print 273  
 FUNC (.KBCRT parameter) 56

general I/O handler (CFGIOERR) 296  
 generate a self-check digit 263  
 global tables 261  
 GOTAB 150  
 GOTO 150  
 GSCK 263

HDR1 label 187  
 HDR1 label, translate 75  
 HDR1 labels 36  
   for SCS conversion data sets 40  
   update 36  
 hex value of current position 23  
 hexadecimal field 107  
 HEXBIN 240  
 HLIN (.KBCRT parameter) 55  
 horizontal tab table 71  
 HTAB (.COMM parameter) 71

I/O control block 69, 72  
 IF 154  
 IF BRa 159  
 IF BRn 0 154  
 IF fmt 156  
 IF Ra 160  
 IF Rn - 155  
 IF Rn AN 155  
 IF Rn CK 155  
 IF Rn SN 155  
 IF Rn 0 155  
 IFB 164  
 IFC 163  
 IFD 161  
 IFDSI 157  
 IFH 161  
 IFHI 165  
 IFI 157  
 IFIR 158  
 IFLO 165  
 immediate data, binary relational compare 161  
 included lines 116  
 increment binary register and skip 167  
 IND (.EQUATE parameter) 67  
 index of data table 253  
 index tables 38  
   density of index tables 39  
 indexed call 151, 153  
 indicators 8, 258  
   system indicators 8, 283  
   field modification indicators 35  
 indirect instruction execution 263  
 INIT 181  
 INIT (.DC parameter) 60  
 INIT (.TITLE parameter) 114  
 initialization of data areas 60  
 initialize a data area 60  
 initialize a diskette 181  
 initialize a listing title 113  
 initialize communications session 170  
 initialize diskette 74  
 input translate table, self-check 119, 353

INSBLK 187  
 insert a block of records 187  
 insert a record 187  
 insert a table entry 256  
 insert mode 28  
 instruction labels 123  
 instructions 123  
 INTAB (.SELFCHK parameter) 119  
 INXEQ 263  
 IOB 69, 72  
 IOB chain 174, 178

KACCPT 207  
 KATTCH 208  
 KDETCH 209  
 KERRCL 210  
 KERRST 209  
 key entry 199  
 key indexed read data set 74  
 key indexed update data set 74  
 keyboard function bit numbers 371  
 keyboard operations 204  
 keyed data set 38, 184  
 KEYOP 211  
 keystroke counters 285  
 keystroke router routine (CFKEYRT) 296  
 KLEN (.DATASET parameter) 78  
 KPOS (.DATASET parameter) 78

label table 18, 81  
 label update 36, 72, 178, 179  
 labeled addressing 125  
 last line (for SCS conversion) 78  
 LBUF (.COMM parameter) 71  
 LBUF (.DATASET parameter) 76  
 LEN (.DC parameter) 58  
 LEN (.FMTFLD parameter) 86  
 LENG function 134  
 length specification 134  
   changing declared length 134  
   declared length 58  
   of edit format field 86  
 LEVL (.DC parameter) 59  
 line size (for SCS conversion) 78  
 LINSZ (.DATASET parameter) 78  
 listing 278  
 literal spacing 274  
 load  
   binary register 229  
   decimal register 231  
   partition 248  
 LOAD 249  
 load parameters 248, 250  
 load the assembler 272  
 LOC (.DC parameter) 59  
 lock system 268  
 logical buffer 36, 71, 76  
 logical device identifier (ID) 6, 73  
 logical instructions 135

logical record length 176  
 loop control 166  
 LSTLN (.DATASET parameter) 78

magnetic stripe reader 225, 312  
 main storage boundaries 13  
 main storage duplicate 105  
 main storage duplicate filed 31  
 main storage partitions 5  
 main storage store 105  
 main storage store field 31  
 mandatory enter field 111  
 mandatory fill field 111  
 mask 165, 166  
 MAXM (.TABLE parameter) 79  
 MCHK (.START parameter) 52  
 MD (.SFMTFLD parameter) 104  
 MDUP (.SYSTAB parameter) 83  
 message display routine (CFMSGDSP) 297  
 MMCRT 245  
 mnemonic to op code conversion chart 337  
 mnemonics list 127  
 MOD (.SELFCHK parameter) 116  
 MODE (.KBCRT parameter) 54  
 modes of entry 26, 54  
 modification of instruction 263  
 modify zone decimal register 148  
 modulus, self-check 42, 116  
     standard modulus 10 45  
     standard modulus 11 46  
 MOFF 240  
     monocase conversion status 93  
 move data 240  
 MS (.SFMTFLD parameter) 105  
 multiply, binary 137  
 multiply, decimal 144  
 multivolume option 180  
 MVC 242  
 MVCR 242  
 MVCV 243  
 MVER 241

NAME 176  
 NAME (.DATASET parameter) 73  
 NAME (.INCLUDE parameter) 116  
 NFIL (.KBCRT parameter) 54  
 NL, self-check 356  
 NMIN (.KBCRT parameter) 55  
 non-display of the status line 23  
 NOP 150  
 NR, self-check 356  
 null operation 150  
 NUMB (.EQUATE parameter) 66  
 NUMB (.SPACE parameter) 114  
 numeric fill character 54  
 numeric only field 108  
 numeric shift field 107

op code 337  
 OPEN 177, 193  
 open a data set  
     communications 170  
     diskette 177  
     printer 193  
     open keyboard/display 216  
 operator detach routine (CFPERATT) 298  
 OPTION (.START parameter) 52  
 OR, binary 139  
 ORD 75  
 ordered key data set 75  
 ORG (.START parameter) 52  
 output translate table, self-check 119, 367  
 OUTTAB (.SELFCHK parameter) 119  
 overlapped extent check 75  
 overlapped fields 242  
 overlapped I/O 180

page size (for SCS conversion) 78  
 partial overlay 249, 252  
 partition control area 8  
 partition subroutine stack 20, 83  
 partition work area 12  
 partitions 6  
 pass EBCDIC to keyboard 215  
 Pass scan code to keyboard 215  
 PB1 (.DATASET parameter) 75  
 PB2 (.DATASET parameter) 76  
 perform keyboard function 215  
 PGSIZ (.DATASET parameter) 78  
 physical buffer 36, 75, 177  
 PIC (.FMTFLD parameter) 88  
 PIC (.SFMTFLD parameter) 105  
 picture definition 87, 105  
 PLUNAME (.COMM parameter) 72  
 PNAM (.START parameter) 52  
 pointer I/O 37, 75  
 pointer I/O for printer 196  
 pointers 5  
     screen position pointers 92  
     subroutine stack pointer 20  
 POP 202  
 position current record pointer 181  
 positions remaining in field 23  
 POSN 181  
 PRBA 54, 200  
 PRBA (.KBCRT parameter) 54  
 PREFIX (.DC parameter) 59  
 previous record buffer 54, 200  
 PRINTOFF 50, 114  
 PRINTON 50, 114  
 PRMT (.SFMTPMT parameter) 100  
 PRMT (.SYSTAB parameter) 82  
 PROD (.SELFCHK parameter) 120  
 product table 120  
 product table, self-check 355, 359  
 program check error handler (CFPGMCHK) 299  
 program check errors 285, 299  
 prompt system table 82  
 prompts 29  
     fixed position prompts 54, 101  
     standard position prompts 100

quick release data set 75

read  
  a table entry 254  
  communications 172  
  diskette 186  
  magnetic stripe reader 225

READ 186  
READMG 225  
REBF 247  
receive communications record 172  
RECFM (.COMM parameter) 72  
RECL 176  
RECL (.COMM parameter) 71  
RECL (.DATASET parameter) 76  
record length 71, 76  
REG (.EQUATE parameter) 67

registers 8  
  binary registers 9  
    system registers 9, 284, 303  
  decimal registers 10  
  declare a register 65  
  double binary registers 9

relational compare, binary 159  
relational compare, decimal 160  
relative record number 184  
release a data set 179  
release character and edit fields 212  
REPFLD 227  
replace field on screen 227  
rerun mode 27  
rerun/display mode 27  
RESCAL 201  
reset external status bit 203  
RESMXT 204  
resource allocation table 6, 265  
restricted external status indicator 304  
RESUME 201  
resume and call subroutine 201  
RETEXT 204  
retext and resume 204  
RETURN 152  
return and enable external status 204  
returning (RG) exits 34, 93, 96  
RG 34, 93, 96, 309  
RGLT (.START parameter) 53  
right adjust, blank fill field 112  
right adjust, zero fill field 112  
right-to-left field 112  
RL 140  
rotate instructions, binary 139  
rotate left, binary 140  
rotate right, binary 140  
RR 140  
RSTMG 227  
RTIMER 228  
RXORW 267

SCREEN (.KBCRT parameter) 56  
screen attributes 93, 214  
screen format 29, 92, 313  
  secondary screen format 33  
screen format control string 30, 96, 199, 313  
screen position pointer 92, 95  
SCS control characters 343  
SCS conversion data set 40, 75, 184, 186, 194  
  for diskette 40

SEARCH 188  
search a table 258  
search resource allocation table 265  
secondary format 99  
secondary screen format 33  
secure diskette 176, 177  
secure volume processor (CFSECVOL) 300  
security ID 71  
self-check 115  
self-check algorithm 43  
self-check computations 351  
self-check digit 263  
self-check field 41  
self-check modulus 42  
self-check register 41  
sequential search 189  
sequential read data set 73  
sequential update data set 36  
sequential write data set 73  
set bits with mask 266  
set graphics error action (for SCS conversion) 78  
set indicators 266  
SETOFF 266  
SETON 266  
SFMT (.SYSTAB parameter) 83  
SGEA (.DATASET parameter) 78  
share data set, for printer 196  
share data sets 40  
shared tables 261  
shift instructions, decimal 145  
shift left, binary 140  
shift left, blank fill, decimal 146  
shift left, zero fill, decimal 146  
shift right and round, decimal 148  
shift right, binary 141  
shift right, pad blank, decimal 147  
shift right, retain sign, decimal 147  
shift 1 to blank register, decimal 146  
shift/rotate instructions, binary 139  
short branch instructions 158  
SIDH (.COMM parameter) 71  
SIDL (.COMM parameter) 71  
sign control position 11  
signed numeric field 108  
skip on AND/exclusive OR mask 165  
skip on bit mask 164  
skip on constant compare 162  
SKIP WHILE 167  
SL 140, 146  
SLS 146  
SOFF 266

software error mode 209  
SON 266  
special characters only 109  
special characters shift filed 108  
special verify status 94  
SR 141, 146  
SRAT 250  
SRR 148  
SRS 147  
STACK (.SYSTAB parameter) 83  
stack pointer 20  
standard load processor 301  
standard position prompt 100  
statement symbols 14  
status line 22  
    non-display of the status line 23  
storage structures 59  
store  
    binary register 234  
    constant 236  
    decimal register 234  
store field 31, 104  
subfield definition (PIC) 105  
subfields 87  
subroutine instructions 151  
subroutine stack 83  
subroutines 19  
    calls 20, 151  
    calls through table 153  
    partition subroutine stack 20  
    returns 20  
subtract, binary 137  
subtract, decimal 143  
symbolic labels 123  
symbols 14  
    statement symbols 16  
    syntax symbols 15  
SYSACLC 317  
SYSLCK 268  
system control block 5  
    pointers 5  
system indicators 8, 283  
system registers 9, 284, 303  
system table 17, 82, 253  
    for edit formats 83  
    for main storage duplication 83  
    for prompts 82  
    for screen formats 83  
SYSUNL 268

table (data table)  
    argument 253  
    control statements 79  
    index 253  
    system table 253  
tables 17  
    data tables 18  
    label tables 18  
    system tables 17

TBBS 258  
TBDL 255  
TBFH 259  
TBFL 259  
TBFX 260  
TBIN 256  
TBRD 254  
TBRL 254  
TBWE 256  
TBWT 257  
TCLOZ 172  
TCTL 170  
terminate communications session 172  
test binary register for 0 154  
test data set indicator 157  
test decimal register 155  
test format number 156  
test indicator 157  
test indicator and reset 158  
time slice factor 53  
TINIT 170  
TLCK 261  
TLOC (.DATASET parameter) 78  
TMSL (.START parameter) 53  
TOPEN 170  
TRANS 269  
TRANS (.DATASET parameter) 76  
translate and test 268  
translate HDR1 label 75  
translate table 76, 268  
transmit communications record 173  
TRAP 371  
TRAP (.KBCRT parameter) 54  
TREAD 172  
TRT 269  
TTERM 172  
TUNLCK 262  
TWAIT 171  
TWRT 173  
TYPE (.COMM parameter) 70  
TYPE (.DATASET parameter) 73  
TYPE (.DC parameter) 58  
TYPE (.FMTFLD parameter) 86

unconditional branch 149  
unlock system 268  
update mode 26

V= 138  
variable leveling 59  
verify correction keystroke counter 286  
verify mismatch error 27  
verify mode 27  
verify/copy option 179  
vertical tab table 71  
volume ID 251  
VTAB (.COMM parameter) 71

WAIT 180, 193  
wait for I/O completion  
  communications 171  
  diskette 180  
  printer 193  
weighting factors 117  
weights, self-check 355  
WFMCR 248  
WGTS (.SELFCHK parameter) 117  
WRBF 247  
write  
  a table entry 254  
  communications 173  
  diskette 186  
    write delete 186  
    write insert 187  
write-protect 179  
WRT 186  
WRTI 187  
WRTS 186

X= 139

zone 11  
ZONE 148  
zone modification, decimal 148

### READER'S COMMENT FORM

Please use this form only to identify publication errors or request changes to publications. Technical questions about IBM systems, changes in IBM programming support, requests for additional publications, etc, should be directed to your IBM representative or to the IBM branch office nearest your location.

**Error in publication** (typographical, illustration, and so on). **No reply.**

*Page Number    Error*

**Inaccurate or misleading information in this publication.** Please tell us about it by using this postage-paid form. We will correct or clarify the publication, or tell you why a change is not being made, provided you include your name and address.

*Page Number    Comment*

IBM may use and distribute any of the information you supply in any way it believes appropriate without incurring any obligation whatever. You may, of course, continue to use the information you supply.

- No postage necessary if mailed in the U.S.A.

Name \_\_\_\_\_

Address \_\_\_\_\_

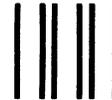
\_\_\_\_\_

Cut Along Line

Fold and tape

Please do not staple

Fold and tape

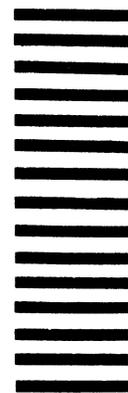


NO POSTAGE  
NECESSARY IF  
MAILED IN THE  
UNITED STATES

**BUSINESS REPLY MAIL**  
FIRST CLASS      PERMIT NO. 40      ARMONK, N. Y.

POSTAGE WILL BE PAID BY ADDRESSEE

IBM CORPORATION  
Product Information Development  
Dept. 997  
11400 Burnet Road  
Austin, Texas 78758



Fold and tape

Please do not staple

Fold and tape



**International Business Machines Corporation**

**General Systems Division**  
4111 Northside Parkway N.W.  
P.O. Box 2150  
Atlanta, Georgia 30301  
(U.S.A. only)

**General Business Group/International**  
44 South Broadway  
White Plains, New York 10601  
U.S.A.  
(International)





**International Business Machines Corporation**

**General Systems Division  
4111 Northside Parkway N.W.  
P.O. Box 2150  
Atlanta, Georgia 30301  
(U.S.A. only)**

**General Business Group/International  
44 South Broadway  
White Plains, New York 10601  
U.S.A.  
(International)**