File No. 1620-36
GC26-5739-4

IBM Systems Reference Library

IBM 1620 Monitor I System Reference Manual

This publication describes the 1620 Monitor I System, a combined operating and programming system. This system includes a Supervisor Program, a Disk Utility Program, an SPS II-D Assembler, and a FORTRAN Compiler. The latter three programs operate under control of the Supervisor Program to provide continuous operation.

Also described is the 1620-1443 Monitor I System. This system is designed to use the IBM 1443 Printer as an integral unit in the processing of source programs.

This is a reprint of an earlier publication incorporating the following Technical Newsletters: N26-0108, N26-0122

Contents

Preface

In data processing installations, a large amount of computer idle time may be spent between jobs to set up the computer for the next job. Programs must be readied, data must be readied, etc. The amount of time spent in these activities is greater in installations where many different jobs are done in the daily schedule. Because computer utilization is usually based upon the time spent in executing a job, utilization may appear to be low where multiple jobs increase setup time.

To increase 1620 computer utilization, the user may now adopt the Monitor I Programming System. The primary function of the Monitor I System is to provide continuous operation during a sequence of jobs which might otherwise involve several independent programming systems. To do this, Monitor I coordinates computer activity by providing a communication region for independent programming systems and by transferring control between them. Operation is continuous and setup time is reduced. This effects a substantial time saving in computer operation and allows greater flexibility in programming. The monitor concept – to control the operation of several unrelated routines and machine runs so that the computer and computer time are used advantageously – is not new. This concept, previously employed by other large-storage-capacity data processing systems, is made possible for the 1620 by the 1311 Disk Storage Drive.

The Monitor I System utilizes the large storage capacity of the 1311 Disk Storage Drive. Thus it is possible to assemble, assemble and execute, compile, compile and execute, and execute programs stored in disk storage. In addition, the normal disk storage maintenance tasks, such as storing programs, can be performed by the system.

A stacked input arrangement provides direction for the system. This direction is given in the form of control records, which are prepared prior to the actual operation by the programmer and/or operator. These records direct the sequence of jobs without interrupting continuous operation. A typical sequence of jobs could be a compilation of FORTRAN programs, assembly of SPS programs, compilation and execution of a FORTRAN program, execution of a disk-stored program, and punching of a disk-stored program into cards.

In addition to substantial saving of computer time, Monitor I reduces the amount of programming time required by the user. This is made possible through sharing common subroutines by unrelated programs. For example, input and output for all SPS object programs can be performed by a common input/output subroutine. Thus, programming time is reduced because a common subroutine need only be written once. Because most programs require a subroutine of this nature, it has been made an integral part of the Monitor I System, available to all 1620 user programs.

To use the Monitor System, the programmer must pay particular attention to control records and stacked input arrangements described in this publication. In addition to directing the sequence of jobs, control records allow the user the flexibility of assigning the specifications for each job. Various jobs, each with its own control record, are entered into the 1620 in the stacked arrangement. Jobs are performed in the order in which they are encountered under control of the Monitor and without operator attention. However, some messages generated by the Monitor regarding the status of processing may require operator intervention.

The IBM 1620 Monitor I System is comprised of four separate programs:

Supervisor Program
Disk Utility Program
SPS II-D
FORTRAN II-D

The material contained in this publication is organized and presented under these headings.

IBM 1620 Monitor I System

Monitor I (Figure 1), a collective name for four distinct but interdependent programs — Supervisor, Disk Utility, SPS II-D, and FORTRAN II-D programs — is a powerful, combined operating and programming system. Systems of this type have previously been available only on other large-storage capacity computers. The 1311 Disk Storage Drive with two-million positions of storage makes possible the implementation of such a system on the 1620. Although both SPS and FORTRAN are included with the Monitor System, either may be deleted from the system, if desired.

The complete Monitor System resides in disk storage and only those routines or programs required at any one time are transferred to core storage for execution. This feature, which is common to a Monitor System, minimizes core storage requirements and permits segmenting of long programs. It makes any one of many programs accessible to the computer with minimum delay or manual operation.

Inclusion of SPS and FORTRAN programming languages in the Monitor System facilitates development of a library of user object programs. Programs can be stored in cards or paper tape, as they were stored in the past. In addition, they can be stored in disk storage without the necessity of assigning actual storage areas, remembering or documenting the storage assignments, and updating assignments and documentation as conditions change. These disk-stored programs can be referred to by a name or number when called for execution. If a program is added to the user's repertoire of programs, the storage locations of the other programs may be adjusted to prevent the overlapping of data in disk storage. An account of available disk storage is kept for the user as adjustments are made to disk storage.

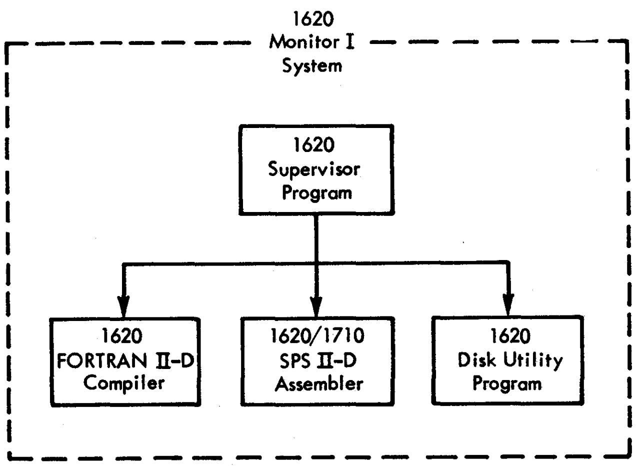


Figure 1. 1620 Monitor I System

To make effective use of disk storage, a common disk working area is maintained for all facets of the Monitor System. One use for this area is to store intermediate output from FORTRAN compilations or SPS assemblies to speed up these operations.

Approximately 22 percent of disk module zero is used for the Monitor System itself. The remainder of disk storage, with the exception of a few sectors on each additional disk module, can be used for user programs and data records.

The use of disk storage is controlled by three disk-stored tables. The first and major table is the Disk Identification Map known as DIM. Each disk-stored item (program, data, or table) has a DIM entry which contains information on where the item is stored, how many disk sectors it occupies, and, if it is a program, its core storage address. Thus, to refer to an item, it is only necessary to use its DIM entry number. For those who prefer to use an alphabetic name instead of a number, a second table, called Equivalence table, lists the names and their equivalent DIM entry numbers. The third table, a Sequential Program table, shows the assignment of disk storage by DIM entry numbers and the availability of unassigned storage. Maintenance of these tables is performed automatically by the Monitor without user supervision or direction.

To examine the Monitor System, it is best to summarize the functions of its various parts: Supervisor Program, Disk Utility Program, SPS, and FORTRAN. Subsequent sections present the detailed information for each, required to program and operate.

The four primary functions of the Supervisor Program are:

- Input/output
- I/O error detection and correction
- Program loading
- Control

Individual routines which perform these functions follow.

Input/Output routine, which performs all I/O functions (card, paper tape, typewriter, or disk) for the Monitor System.

I/O Error routine, which provides for error detection and correction for all I/O operations performed by the Input/Output routine.

Loader routine, which loads object programs into core storage from card, paper tape, typewriter or disk input.

Monitor Control Record Analyzer routine, which records and interprets jobs to be performed, and transfers control to other routines or programs while maintaining communication so that control may eventually be returned to itself.

Because all input/output is processed by the Input/Output routine, all data, programs, or control information from any input unit enter the 1620 CPU through this routine. The Input/Output routine is able to segment the continuous flow of input information into discrete jobs. Certain information is recognized as control information. With the proper control information, the Monitor could, for example, execute a particular series of jobs consisting of (1) compiling a FORTRAN program and executing it using supplied data from any input source, (2) loading an SPS object program from disk storage and executing it using data supplied from any input source. All of this can proceed automatically from one task to the next without stopping the machine for operator action. Thus, ease of operation is achieved with increased efficiency and throughput.

The Input/Output routine recognizes Monitor control information by two record marks (‡‡). If the first two columns of a card or tape record contain record marks, the control information is examined by the Monitor Control Record Analyzer routine to see which one of twelve possible tasks is to be performed next.

1. ‡‡ JOB, initiates a new job.
2. ‡‡ SPS, indicates that an SPS source program(s) is to be assembled.
3. ‡‡ SPSX, indicates that an SPS source program is to be assembled and executed.
4. ‡‡ FOR, indicates that a FORTRAN source program(s) is to be compiled.
5. ‡‡ FORX, indicates that a FORTRAN source program is to be compiled and executed.
6. ‡‡ DUP, indicates that a Disk Utility routine is to be executed.
7. ‡‡ TYPE, indicates the next control record is to be entered from the typewriter by the operator.
8. ‡‡ PAUS, allows operator action.
9. ‡‡ XEQ, initiates loading and execution of an SPS object program.
10. ‡‡ XEQS, initiates loading and execution of FORTRAN or SPS object programs with subroutines.
11. ‡‡‡‡, is used to indicate the end of a job.
12. ‡‡, serves as a Comments record.

These control records can be classified as being one of three types. Type 1 provides control information which is used as instructions to the Supervisor Program. Type 2 provides for loading and starting the execution of user-written programs, using data from control records to initialize various subroutines used by programs. Type 3 provides for functions similar to type 2 except that the FORTRAN, SPS or the Disk Utility Program with its functional capabilities is also executed.

When FORTRAN, SPS, or the Disk Utility Program is executed, all input records are examined for an asterisk (*) in the first position to determine if they are control records. These records have two functions: (1) they provide information about specific Disk Utility jobs to be performed, (2) they provide the specifications for FORTRAN compilations or SPS assemblies.

Ten Disk Utility Control records provide various user options. Each of these control records transfers control to an individual routine to perform the desired function.

1. *DWRAD, indicates sector addresses are to be written on a disk pack.
2. *DALTR, allows the operator to alter disk-stored data from the typewriter.
3. *DDUMP, indicates disk-stored data or programs are to be outputted in cards, paper tape or on the typewriter.

4. *DLOAD, indicates a program is to be loaded into disk storage from cards, paper tape, or the working cylinders.
5. *DREPL, indicates a disk-stored program is to be replaced by another.
6. *DCOPY, indicates that data or programs are to be copied into another area of disk storage.
7. *DELET, indicates a program is to be deleted from disk storage.
8. *DFINE, allows changes to be made to the Monitor System specifications.
9. *DLABL, is used to write identity labels on disk packs.
10. *DFLIB, allows names to be assigned to FORTRAN library subroutines.

Examples of FORTRAN and SPS asterisk control records are:

FORTRAN	SPS	Description
*POBJP4	*OUTPUT CARD	Punch object program into cards
*PSTSN4	*PUNCH SYMBOL TABLE	Punch Symbol Table into cards

The many routines of the Disk Utility Program are designed to perform the necessary but tedious disk housekeeping functions. Items stored on the disk or items to be stored on the disk can be referred to symbolically; that is, by a symbolic name. The Monitor System will allocate and keep track of disk storage areas for the user.

The SPS II-D assembler translates programs written in symbolic language into machine language. The symbolic language is an extension of 1620/1710 SPS. Only one pass of the source statements is required because intermediate output is stored on the disks. Also, the Symbol table is disk stored, thereby allowing for assembly of programs with a great number of labels. An additional symbol table, known as the System Symbol table allows different SPS source programs to use common labels.

I/O macro-instructions are provided in the SPS language to relieve the programmer of writing Input/Output routines in source programs for card, paper tape, or disk operations. Thus, all I/O functions including error detection and correction can be standardized. GET and PUT macro-instructions are used for reading and writing. These instructions generate appropriate linkages to the Input/Output routine which does the actual reading or writing (card, paper tape, or disk).

To eliminate the necessity of having an entire object program in core storage for execution at any one time, a CALL macro-instruction is provided. This instruction enables an object program in core storage to be overlayed with instructions from disk storage. Thus, the size of an object program can be virtually unlimited.

Immediately following assembly, an object program is located in the disk working cylinders. It can be stored in disk storage without outputting it in some other form, thereby greatly reducing program loading time. Program listings can be obtained in either card or typewriter form. The program can also be immediately executed or punched out into cards or paper tape for loading and executing at any later time.

Any library subroutine set may be used by an object program. User-written relocatable programs can be automatically added to the library subroutine set, if desired.

The FORTRAN compiler translates programs written in the FORTRAN II-D language into 1620 machine language. Because of the large storage capacity of the 1311, source programs are read into the computer just once. Intermediate output is stored in the disk work area to minimize input/output time. Following compilation, an object program may be executed immediately, punched into cards or paper tape, or stored in disk storage for execution at a later time.

Statements are provided in the FORTRAN language that permit the use of disk storage. These statements provide for (1) defining the size and quantity of data records to be stored, (2) reading and writing on disks, (3) positioning the disk access mechanism for reading or writing, (4) returning control to the Supervisor after execution of a FORTRAN object program, (5) overlaying programs in core storage with other programs from disk storage and executing the overlaying programs.

Subprograms can reside in either core or disk storage. Disk-stored subprograms can be called into core storage only when needed. Thus very large problems can be accommodated.

New floating-point and subscripting subroutines, which use the Indirect Addressing feature and compiler algorithms, provide for efficient object program execution. Subroutines written in SPS II-D language can easily be added to the FORTRAN subroutine library. Also, subprograms written in SPS can be called for execution by FORTRAN programs or subprograms.

Object programs, either SPS or FORTRAN, can be punched into cards or paper tape following assembly or compilation, if desired. However, assembly or compilation time is shortened if the object program is stored in disk storage rather than cards or paper tape. A standard System Output format, which may be in absolute or relocatable form, is used to output object programs. Only the programs outputted in the standard format can be reloaded by the Monitor System.

SPS and FORTRAN source programs are assembled or compiled using a given mantissa length, noise digit, subroutine set, etc., specified in a common Communications Area. These given specifications are used for all assemblies or compilations; however, the user may change one or more of the specifications, such as mantissa length, with control records for a particular assembly or compilation.

SPS or FORTRAN source programs can be entered into the 1620 from cards, paper tape, or typewriter. Object programs, however, can be entered from cards, paper tape, or disk storage. Data, of course, can be entered from any input source as directed by an object program.

SPS II-D and FORTRAN II-D object programs will be in a System Output format. System Output consists of relocatable or absolute records. The relocatable format allows the programmer to specify a different core storage address for the start of a program each time it is loaded. All addresses within a program that are being relocated are adjusted relative to a new starting address. The starting address (relocation address) is specified by means of a control card at load time. Absolute format, which can be generated by the SPS II-D assembly program, is assembled to load to a fixed core storage area.

Machine Requirements

The Monitor I System operates on a 1620 System, Models 1 or 2, which has a minimum of 20,000 positions of core storage, a 1311 Disk Storage Drive, Model 3, and the special features, Indirect Addressing and Automatic Divide. Automatic Divide is required only for execution of FORTRAN II-D and for SPS II-D programs which use the Arithmetic and Functional subroutines.

Operation

The Monitor I System is available in either card or paper tape versions, ready for loading to disk storage. After either version is loaded, both cards and paper tape may be used with the programs.

The system is self-loading, i.e., it contains the loading instructions that enable it to load itself into disk storage. For handling convenience, a sequence number is punched in card columns 76-80 of card input. Disk storage drive 0 (address block 00000-19999) is required for storing the Monitor I System. A description of the loading process, including operating procedures, is included under MONITOR I SYSTEM LOADER.

To begin operation of the Monitor I System, load the Supervisor Program into core storage from disk storage. This is accomplished by entering from the typewriter or a card, the following instructions:

Core Storage Address of Instruction	Instruction
00000	34 00032 00701
00012	36 00032 00702
00024	49 02402
00031	X
00032	Y1963611300102

X identifies the form of input for the Monitor Control records: 1 for typewriter, 3 for paper tape, and 5 for cards. Y specifies the disk drive code (1, 3, 5, 7) identifying the disk module where the Monitor I System is loaded. (Any time the Monitor pack is moved to a different drive, this sequence of instructions must be repeated.)

When the Release and Start key or Load key is depressed, the Supervisor Program is read into core storage and the first Monitor Control card is read in under control of the Supervisor Program by the Monitor Control Record Analyzer routine.

This routine reads all control records and types them out to inform the operator of the status of processing. If operator intervention is required for any reason, the routine will type a message and halt the 1620. Processing can be resumed as explained in the section concerning the MONITOR CONTROL RECORD ANALYZER ROUTINE.

The Parity, I/O, and Overflow Program switches should be on to operate the Monitor I System.

Disk Storage Requirements

Portions of cylinders 24-25 and 80-99 of module 0 are used by the Monitor I System. Unused portions of these cylinders can be listed using the Disk-to-Output routine to obtain an availability list. A list of the assigned cylinders and DIM entry numbers associated with the programs, routines, and tables stored in this disk area follows.

Program/Table	Cylinders	DIM Numbers
Working Storage	00-23	1
DIM table	24	3
Equivalence table	25	2
FORTRAN Subprogram Loader	80, 84	138, 147, 149, 150, 152, 157
FORTRAN Library Subroutines	81	*10-39
SPS Library Subroutines	82-83	**40-130
FORTRAN I/O and Arithmetic Subroutines	84, 97	144-146
FORTRAN Compiler	86-90	136-137, 153, 156
Disk Utility Program	85, 91-92	139-143, 154, 155
SPS Assembler	93-96	8, 9, 131, 132
SPS Subroutine Supervisor	85	133
Supervisor Program with I/O Routine	98	134, 148
System Output Loader Routine	98	135
1440, 1401, 1410, Systems Header Label Area ***	99	166-169
Monitor Disk Pack Label***	99	158-161
Mutual Disk Pack Identification Label ***	99	162-165
Sequential Program table ***	99	4, 5, 6, 7
System Area	99	151

*Only DIM entries 10-25 are in use when the system is delivered.

**Only DIM entries 40-57, 70-87, 100-117, and 130 are in use when the system is delivered.

***Present on all available modules.

Any program or routine not to be used may be deleted from disk storage using the Disk Utility Program, Delete Programs routine. All areas are assigned automatically to module 0 when the system is initially loaded; however, certain assignments can be altered by the user as described in the section concerning 1620 Disk Utility Program, under DEFINE PARAMETERS (DFINE) CONTROL CARD. With this entry it is possible to utilize more than one disk storage drive and to enlarge or shorten the tables used by the Monitor System.

WORK CYLINDERS

Cylinders 00-23, reserved for working storage, are available to every program. This area is not available for permanent storage of programs and data. This area is identified by DIM entry 0001.

The working cylinders may be thought of as a "scratch pad"; i.e., an area for storing intermediate results. These results should be moved to another disk area if they are to be retained for further use. Because the Monitor I System uses this area to perform its function, the contents of the area are continually changing. However, an object program will occupy the area at the beginning of the work cylinders immediately after compilation or assembly is completed.

EQUIVALENCE TABLE

When a user's object program, identified by name, is loaded into disk storage by its compiler or by the DLOAD or DREPL routines of the Disk Utility Program, its name as well as its DIM entry number is entered in the Equivalence table. Sixteen-digit positions are required for each entry, twelve for the 6-character alphabetic name and four for the DIM entry number that starts with 0001. Eighty sectors with a capacity of 500 program names are reserved to store the Equivalence table immediately following the standard DIM table (cylinder 24).

When the Monitor System is delivered, the first 51 entries are reserved for the system, fifty for FORTRAN library subroutine names, and one for the SPS II-D modification program. Unused names in the first 50 entries will be identified by a field of 16 nines.

When a name, other than a FORTRAN subroutine, is added to the table, it is placed in the 16 positions following the last entry. When a name, other than a name from the first 50 entries, is deleted from the table, all entries in higher-numbered positions are shifted left to overlay the deleted entry. Thus, the table, with the exception of the first 50 entries, will contain only those entries that are in force at any one time. The rightmost position of the table is identified by a record mark.

DISK IDENTIFICATION MAP (DIM) TABLE

The Supervisor Program, Disk Utility Program, SPS assembler, and FORTRAN compiler use this table to find subroutines, data areas, or tables in disk storage. The DIM table on cylinder 24 can accommodate up to 999 20-digit entries. One entry is required for each program or data area permanently stored in disk storage.

The format of the 20-digit DIM entry follows:

$$D\bar{D}DDDD\bar{S}SS\bar{C}CCCC\bar{E}EEEE \ddagger$$

$D\bar{D}DDDD$ is the disk sector address of the program or data.

$\bar{S}SS$ is the sector count.

$\bar{C}CCCC$ is the core address. If this field is all 9's, the program is in System Output format (see SYSTEM OUTPUT FORMAT). If the units position is flagged, the Subroutine Supervisor is used to load the program.

$\bar{E}EEEE$ is the entry address. This address is relative to the load address (first core address to be loaded) for programs in relocatable format.

$\bar{\ddagger}$, .$\bar{\bar{\ddagger}}$, $\ddagger$, or $\bar{\ddagger}$ is the rightmost character of a DIM entry.

These characters indicate the following conditions about a referenced program.

Character	File Protected	Permanently Assigned
⯒	Yes	No
⯒̄	Yes	Yes
ǂ	No	No
ǂ̄	No	Yes

NOTE: (1) That a file-protected program can be read but not written because it has read-only flags written in all (or any one) of the sector addresses in the disk area which it occupies. If read-only flags are written by a user's program in sector addresses where a program is stored, that disk storage area will not be "file protected," however, data cannot be written in individual sectors which contain read-only flags. (2) That a permanently assigned program cannot be repositioned (moved) in disk storage because it has been assigned to a given address by the user.

The Supervisor Program locates a DIM entry in the following manner:

1. It refers to the Equivalence table to find the 4-digit DIM entry number.
2. It doubles this number and adds the sum to 048000.
3. It uses the leftmost five digits of the result to locate the disk sector of the DIM table.
4. It uses the rightmost digit of the result to find the particular DIM entry.

The DIM table can be expanded (see DEFINE PARAMETERS ROUTINE) to contain up to 4995 DIM entries, an increase of 999 entries for each additional cylinder, 4 additional cylinders maximum.

SEQUENTIAL PROGRAM TABLE

The second through eighty-first sectors (80 sectors) of cylinder 99 are reserved on each disk pack for a Sequential Program table which lists the programs, tables, and data areas sequentially by DIM numbers, and available storage space by special coding. The Sequential Program table is used by the 1620 Disk Utility Program to determine the order of programs and available storage space. When a program is added to or deleted from disk storage, the table is updated to reflect the new squence. Each 80-sector table will accommodate up to 2000 4-digit entries. Three types of entries are included in a table.

1. DIM entry numbers for every program or data area specified in the DIM table.
2. Available sector count to indicate the number of available sectors between programs or data within cylinders.
3. Cylinder entry numbers to identify the beginning of each cylinder.

Available sector numbers always begin with 9 (9xxx); the three rightmost digits denote the number of consecutive available sectors. For example, 9021 indicates that 21 consecutive sectors are unused within a cylinder. A maximum of 200 available sectors can be represented by an entry.

Cylinder entry numbers always begin with 70 (70xx); the two rightmost digits represent the cylinder number. One hundred of these entries are contained in the table, one for each of the 100 cylinders numbered 00-99.

An example of how the three types of entries might appear in a Sequential Program table for cylinders 48-52 follows:

```
7048 0434 0435 7049 0436 9010 0437 7050 0437
7051 0437 7052 0437
```

where the programs identified by DIM entry numbers 0434 and 0435 occupy all 200 sectors of cylinder 48, and the programs identified by DIM entry numbers 0436 and 0437 occupy all sectors of cylinders 49-52, with the exception of 10 unused sectors between the two programs in cylinder 49. Note that programs that overlap cylinders will have the associated DIM entry number repeated for each cylinder on which it is stored.

The length of the Sequential Program table is 80 sectors unless changed by a DFINE control record (see DISK UTILITY PROGRAM). Fifty sectors should provide sufficient space in the table if 1000 programs are to be written on a disk pack. Therefore, the user may want to lengthen or shorten the table for his particular needs.

IBM 1440, 1401, 1410, Systems Header Label Area

To facilitate the processing of common disk packs, a standard alphabetic identification label is created on the 1401, 1410, or 1440 Systems. This label is not used by the 1620 System. The disk storage area (first 19 sectors of the last disk track of the last cylinder) reserved for this label can be released for other storage purposes, using the DELET Disk Utility routine, if a disk pack will be used with the 1620 only. The DIM entries for the four modules that may be connected to the system are 166, 167, 168, and 169.

Mutual Disk Pack Label

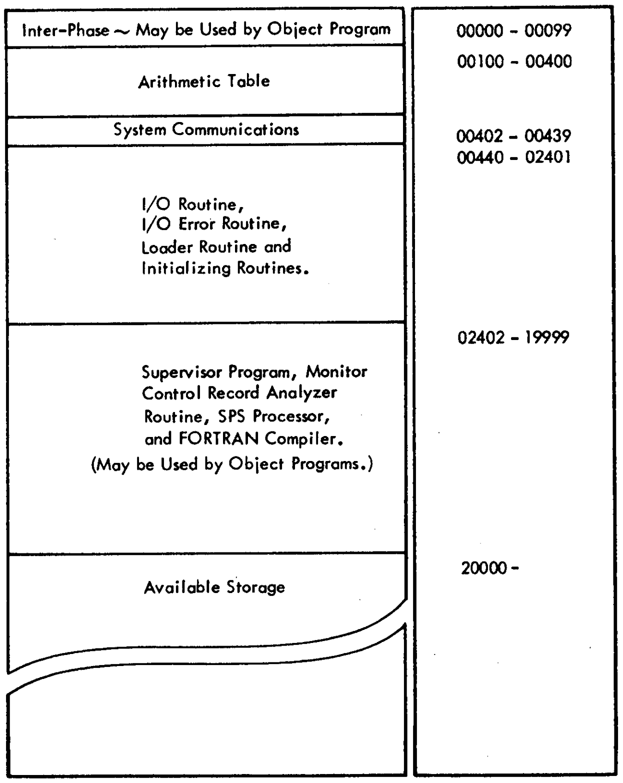
A 5-digit disk pack identification label that can be used by the other systems (1440, 1401, or 1410) must be written on the 32nd through 36th position on the last sector of cylinder 99. This sector should be given the sector address 00199 regardless of the addressing scheme used on the remainder of the disk pack. The sector can be labeled automatically using the Define Disk Pack Label routine of the Disk Utility Program.

Monitor Disk Pack Label

The first sector of cylinder 99 is a label sector, that is, it contains a label to identify the disk pack. Each disk pack used by the Monitor System must include this label. A 5-digit disk pack identification number in the five leftmost positions of the sector constitutes the label. This number is used to provide protection for user's records as explained in the section entitled DISK PACK IDENTIFICATION NUMBERS. This file-protected label must be generated using the Define Disk Pack Label routine. The DIM numbers for these labels are: 0158, 0159, 0160, and 0161 for packs placed on modules 0, 1, 2, and 3, respectively.

Core Storage Requirements

The Monitor I System requires certain areas of core storage in order to operate. Core storage positions 00100-02401 must be permanently assigned to the Monitor I System; however, positions 00000-00099 and 02402-19999 are only temporarily assigned to the system. A temporary area is available to the user for execution of object programs. The core storage layout for the Monitor I System follows.


Inter-Phase ~ May be Used by Object Program
00000 - 00099
Arithmetic Table
00100 - 00400
System Communications
00402 - 00439
00440 - 02401
I/O Routine, I/O Error Routine, Loader Routine and Initializing Routines.
02402 - 19999
Supervisor Program, Monitor Control Record Analyzer Routine, SPS Processor, and FORTRAN Compiler. (May be Used by Object Programs.)
20000 -
Available Storage

Supervisor Program

The Supervisor Program performs the control functions and Input/Output (I/O) functions for the 1620 Monitor I System. The FORTRAN II-D compiler and the SPS II-D processor, under control of the Supervisor Program, can be used to compile or assemble machine language object programs. The Disk Utility Program, also under control of the Supervisor Program, can be used to write disk addresses, to alter a disk sector of data from the typewriter, to load, update, and move programs in disk storage, to delete programs from disk storage, and to copy data, disk to disk.

SPS II-D or FORTRAN II-D object programs can be loaded from cards or paper tape into disk storage under control of the Supervisor Program. Because the I/O functions are performed by a routine contained in the Supervisor Program, the programmer need not concern himself with writing these routines in SPS source language. By use of a macro-instruction in the SPS source program, cards and paper tape can be read and punched, data can be stored and retrieved from disk storage, and data can be read and typed from the typewriter under control of the I/O routine in the Supervisor Program. When a macro-instruction is encountered in a source program, linkage instructions, which provide an exit to an I/O routine, are created in the object program. If desired, the user may manually code linkage instructions without the use of macro-instructions. Manually coded linkage instructions offer certain input/output options, unobtainable with macro-instructions. Error checking and correction procedures are a part of the I/O routine. The Supervisor and Disk Utility Programs use the I/O routine to perform their assigned tasks.

Monitor Control Records

Although the Monitor Control records are described in terms of cards, these records can be in paper tape or typewriter form.

The input to the Supervisor Program consists of one or more "job decks" (Figure 2). A job deck, as the term is used in this manual, may be a program to be compiled or assembled, a combination of these two (including data); it may also be a series of Disk Utility Program operations. The Processing of each job deck is controlled by the Supervisor as specified by the Monitor Control card that precedes it.

When a Monitor Control card is read, the program required to do the job is read into core storage from disk storage. The program then processes input until the end of the job deck is reached, a new Monitor Control card is encountered, or an error occurs. When the end of a job deck is reached or a new Monitor Control card is encountered, the Supervisor Program is reloaded into core storage from disk storage, and the process is repeated. If an error occurs, a message will print to identify the error, and the remainder of the job will be processed. If it is not possible for the job to continue, the Supervisor Program will skip to the next job. All Monitor Control records, with the exception of those entered from the typewriter, will be typed out on the 1620 console typewriter.

The 1620 Monitor I System uses eleven Monitor Control cards to indicate the processing required of the 1620 Supervisor Program. The manner in which the Supervisor handles each of these cards is described in Figure 3.

Operation Codes

An alphabetic pseudo operation code, left-justified in columns 3-6, is used to identify each of the eleven Monitor Control cards. By examining the operation code, the Supervisor Program is able to determine what processing action is required.

JOB

A JOB operation causes (1) the description or operating instructions contained in the JOB Monitor Control card to be typed, (2) modifies the module, if required, and (3) checks to ensure that the proper disk packs have been attached by the operator.

SPS

This operation causes the SPS II-D assembly program to be read into core storage from disk storage and to be executed. The assembled object program may be stored in disk storage and an entry made in the DIM (Disk Identification Map) table.

SPSX

The SPSX operation is similar to the SPS operation, with one exception: after the object program is assembled, it is then executed.

FOR

A FOR operation causes the FORTRAN II-D compiler program to be read into core storage from disk storage and to be executed. The object program can be stored in disk storage. If this occurs, an entry will be made in the DIM table.

FORX

The FORX operation is the same as the FOR operation, with one exception: the object program is executed after it is compiled.

XEQ

The XEQ operation causes the SPS II-D object program, identified by the Monitor Control card data, to be read into core storage from the input device indicated in column 27, and then to be executed. If the object program requires any of the SPS subroutines to operate, or if the object program is a FORTRAN compiled program, the XEQS operation must be used instead of the XEQ operation. Each disk-stored program, called by the XEQ operation, must have a DIM entry to enable the Supervisor Program to find it.

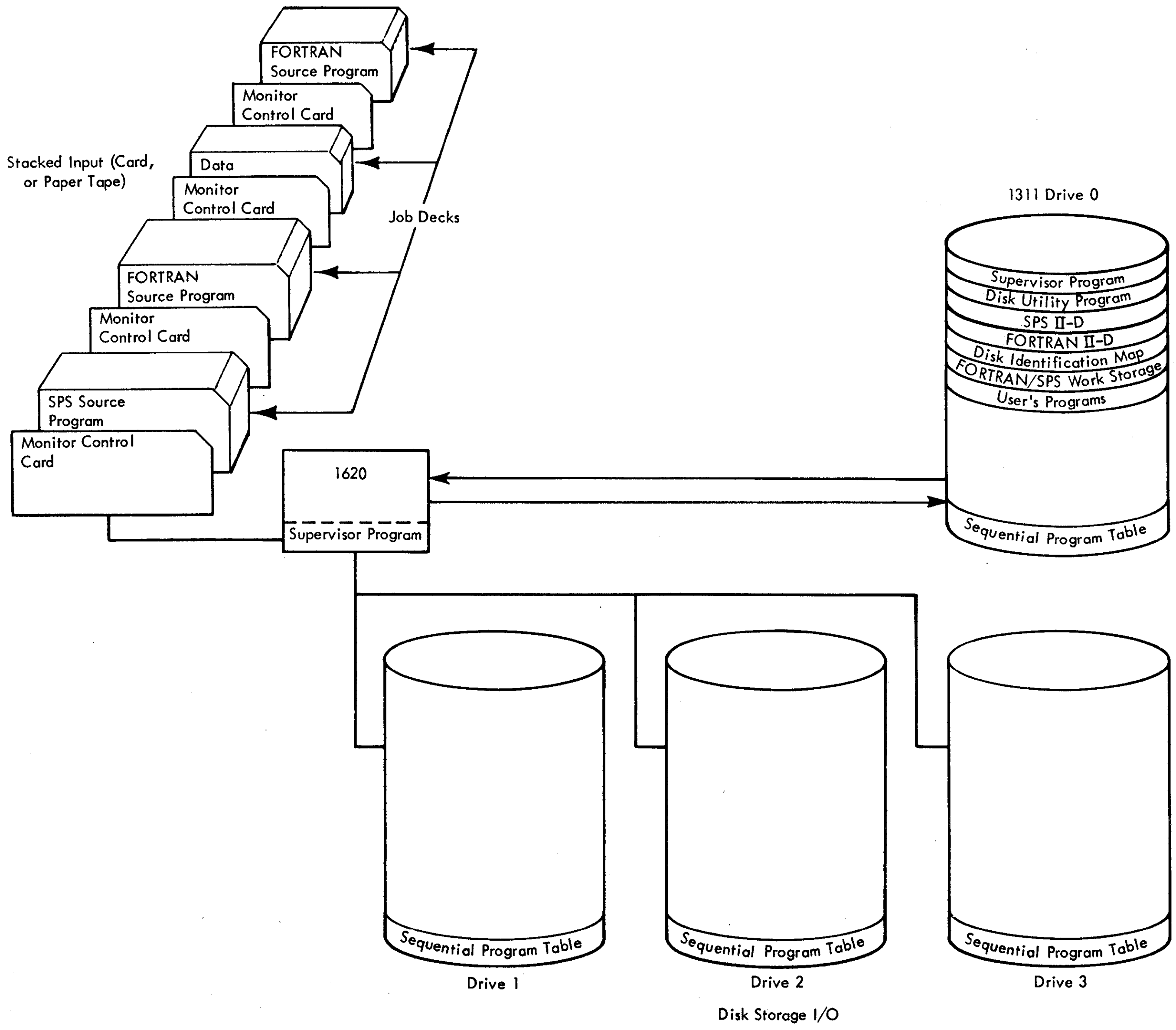


Figure 2. Processing Input Data Under Supervisor Control

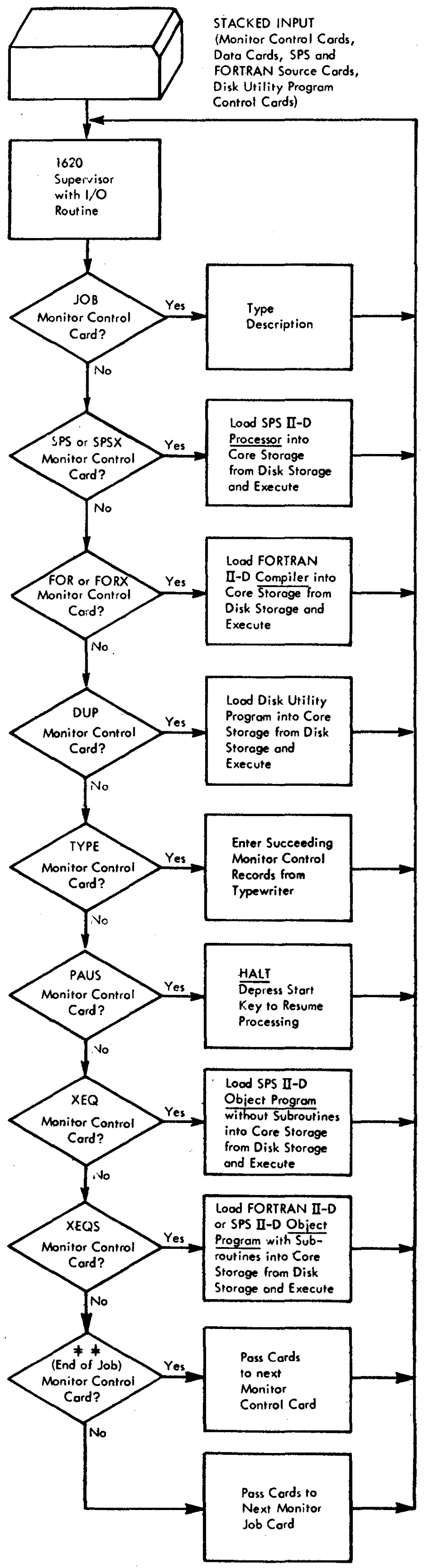

STACKED INPUT (Monitor Control Cards, Data Cards, SPS and FORTRAN Source Cards, Disk Utility Program Control Cards)
1620 Supervisor with I/O Routine
JOB Monitor Control Card?
Yes
No
Type Description
SPS or SPSX Monitor Control Card?
Load SPS II-D Processor into Core Storage from Disk Storage and Execute
FOR or FORX Monitor Control Card?
Load FORTRAN II-D Compiler into Core Storage from Disk Storage and Execute
DUP Monitor Control Card?
Load Disk Utility Program into Core Storage from Disk Storage and Execute
TYPE Monitor Control Card?
Enter Succeeding Monitor Control Records from Typewriter
PAUS Monitor Control Card?
HALT Depress Start Key to Resume Processing
XEQ Monitor Control Card?
Load SPS II-D Object Program without Subroutines into Core Storage from Disk Storage and Execute
XEQS Monitor Control Card?
Load FORTRAN II-D or SPS II-D Object Program with Subroutines into Core Storage from Disk Storage and Execute
‡ ‡ (End of Job) Monitor Control Card?
Pass Cards to next Monitor Control Card
Pass Cards to Next Monitor Job Card

Figure 3. Logic of Supervisor

XEQS

This operation causes an SPS II-D or FORTRAN II-D object program, identified in the Monitor Control card, to be read into core storage from disk storage, cards, or paper tape, and to be executed. If the object program uses any of the SPS subroutines, the XEQS operation must be used. Each disk-stored program called by the XEQS operation must have a DIM entry to enable the Supervisor Program to find it.

DUP

The DUP operation causes the 1620 Disk Utility Program to be read into core storage from disk storage and to be executed.

TYPE

The TYPE operation causes a message – which requests the operator to enter the next Monitor Control record from the typewriter – to be typed and the program to stop to await keyboard input. After the operator enters the Monitor Control record, the Release and Start key should be depressed to resume computer operation. All succeeding control records must be entered from the Typewriter until a JOB Control record is entered to change the source of input.

PAUS

The PAUS operation halts the program to allow the operator to change paper tapes, load input cards, etc. Job processing is resumed by depressing the Start key.

‡‡ (END-OF-JOB)

The ‡‡ (end-of-job) operation causes the message

END OF JOB

to be typed, if a job has actually started, and control is to be resumed by the Supervisor Program. An end-of-job record must follow each job. If this record is not present erroneous results may be obtained.

Monitor Control Card Formats

JOB.

Columns	1-2	‡‡ (identification record marks)
	3-6	Operation (JOB, left-justified).
	7	Source of input,
		5 = card.
		3 = paper tape.
		1 = typewriter.

	8-11	Module change numbers (for disk input only).
	12-31	Disk pack identification numbers (for disk input only), 12-16 drive 0. 17-21 drive 1. 22-26 drive 2. 27-31 drive 3.
	32-80	Description.

SPS, FOR, DUP, TYPE, PAUS.

Columns	1-2	‡‡ (identification record marks).
	3-6	Operation (SPS, etc., left-justified).
	7	Source of input, for SPS, FOR, or DUP Monitor Control cards, 5 = card. 3 = paper tape. 1 = typewriter.

SPSX.

Columns	1-2	‡‡ (identification record marks).
	3-6	Operation (SPSX).
	7	Source of input, 5 = card. 3 = paper tape. 1 = typewriter.
	*8-9	SPS subroutine set identification number.
	*10	N (Noise) digit.
	*11-12	Two digits for indicating length of mantissa.

*Required only when the program to be executed uses other than the standard operating specifications (02 standard subroutine set, 0 standard N digit, 08 standard mantissa length).

FORX.

Columns	1-2	‡‡ (identification record marks).
	3-6	Operation (FORX).
	7	Source of input, 5 = card. 3 = paper tape. 1 = typewriter.
	8	FORTRAN subroutine set identification number.
	9-10	Control card count (number of LOCAL control cards).

XEQ.

Columns	1-2	‡‡ (identification record marks).
	3-6	Operation (XEQ, left-justified).
	7-12	Name of user's program, to be executed (same name assigned in Equivalence table).
	13-16	DIM (Disk Identification Map) entry number. Note that either the name or the DIM entry number must be given (if program is in disk storage), but if both are given, the name takes precedence.
	17-21	Address where loading of user's program begins if program is not in core image. If not supplied, address 02402 is assumed.
	22-26	Address where execution of user's program is to begin if program is not in core image. This address must be relative to the start of the program if the program is relocatable; otherwise, the absolute entry address must be supplied.
	27	Source of input, Blank = disk. 5 = card. 3 = paper tape. (Note that card or paper tape input must be in System Output format).

XEQS.

Columns	1-2	‡‡ (identification record marks).
	3-6	Operation (XEQS).
	7-12	Name of user's program to be executed (same name assigned in Equivalence table).
	13-16	DIM (Disk Identification Map) entry number. Note that either the name or the DIM entry number must be given (if program is in disk storage), but if both are given, the name takes precedence.
	17-21	Address where loading of user's SPS object program begins if program is not in core image. If not supplied, address 02402 is assumed.
	22-26	Address where execution of user's SPS object program begins if program is not in core image.

This address must be relative to the start of the program if the program is relocatable; otherwise, the absolute entry address must be supplied.

27	Source of input, Blank = disk. 5 = card. 3 = paper tape. (Note that card or paper tape input must be in System Output format).
28	Subroutine set identification number for FORTRAN programs only.
29-30	Control card count (number of LOCAL control cards) for FORTRAN programs only.
*31-32	SPS subroutine set identification number (e.g., 00 = fixed-length divide subroutine for machines not equipped with the Automatic Divide feature, etc.).
*33	N (Noise) digit for SPS subroutines.
*34-35	Two digits for indicating length of mantissa for SPS subroutines.

*Required only when the program to be executed uses other than the standard operating specifications (02 standard subroutine set, 0 standard N digit, 08 standard mantissa length).

‡‡ *(End-of-Job).*

Columns	1-2	‡‡ (identification record marks).
	3-4	Operation (‡‡, end-of-job).

Comments Records

Comments records – in card, paper tape, or typewriter form – can be used to specify operating instructions and identify each job. Any number of these records may be inserted in front of a job in the stacked input. Usually they are inserted behind a JOB Monitor Control record. Comments records, unlike Monitor Control records, have no control over the Supervisor. When Comments records are encountered in the stacked input, they are typed out. The format of the Comments record in terms of cards follows:

Columns	1-2	‡‡ (identification record marks).
	3-6	Operation (blanks or any combination of letters and/or digits other than the eleven monitor pseudo operation codes, JOB, SPS, SPSX, etc.).
	7-80	Comments.

When the Supervisor Program reads a Comments card, it will pass subsequent cards until another card with ‡‡, columns 1-2, is encountered. Therefore a Comments card should be followed by another Comments card or a Monitor Control card.

Module Change Numbers

Module change numbers, punched in card columns 8-11 of the JOB Monitor Control cards, can be used on 1620 Systems with more than one 1311 Disk Storage Drive to alter the normal assignment of disk storage drives for any job. For example, a job that uses drive 0 in the execution of its program could use drive 1 instead of drive 0 by the entry of a JOB Monitor Control card with the appropriate module change numbers.

Card columns 8, 9, 10, and 11 of the JOB card represent disk storage drives 0, 1, 2, and 3, respectively. A change to the normal program assignment of a disk storage drive is made by punching the number of the substitute drive into the card column which represents the normal drive. Therefore, in the preceding example, a digit 1 would be punched into card column 8 to alert the program that drive 1 should be used for the job instead of drive 0. Card columns 9, 10, and 11 could be left blank because only the assignment changes must be punched. The assignment of disk storage drives, placed in effect by a JOB card, remains in effect until changed by a succeeding JOB card. If the system is redefined to use more than one drive, a JOB card with module change numbers punched in columns 8-11 must be entered before the additional drives will be accepted by the Supervisor program.

To overlap the time required to change disk packs for one job with the processing time for a different job, the operator may choose to alternate the use of disk storage drives from one job to the next. Alternating drives is possible only when all drives are not in use for any one job. For example, assume that job A is to be followed by job B in the stacked input and the programs for both of these jobs use disk storage drives 0 and 1. Assume further that four disk storage drives are available to the 1620 System that is to perform these jobs. By entering a module change number in the JOB Monitor Control card for job B, the operator can use disk storage drives 2 and 3 for job B in place of drives 0 and 1. Therefore, while job A is being done, the operator could mount the disk packs for job B on drives 2 and 3, thus saving valuable operating time. The JOB card module change numbers should be punched

Card columns	8	9	10	11
Module change numbers	2	3		

so drives 2 and 3 will be used in place of drives 0 and 1, respectively, for job B.

Disk Pack Identification Numbers

Card columns 12-31 of the JOB Monitor Control card can be punched with the four 5-digit disk pack identification numbers, one identification number for each disk storage drive.

Disk Pack Identification Number	Disk Storage Drive
Card columns 12-16	0
17-21	1
22-26	2
27-31	3

The disk pack identification number from the JOB card is compared with the identification numbers recorded on the respective disk packs. If the proper disk packs are not attached by the operator, the Supervisor will halt for operator instructions. If it is desired to omit this check for any disk storage drive, the card field representing the disk storage drive may be left blank. When the operator enters a module change number, no change to the disk pack identification numbers (card columns 12-31) is required.

Stacked Input

Stacked input consists of control records (Monitor, Disk Utility Program, SPS, and FORTRAN), source programs, object programs, and data arranged logically by job. Each job consists of phases which must fit into one of four categories:

1. SPS source program(s) to be assembled.
2. FORTRAN source program(s) to be compiled.
3. Disk Utility routine(s) to be executed.
4. FORTRAN or SPS object program(s) to be called from disk storage with subroutines and executed; or SPS program(s) to be called from disk storage without subroutines and executed.

The order in which jobs are executed is not important, i.e., a Disk Utility routine may be executed be fore a FORTRAN compilation or vice versa. Jobs are executed in the order in which they are encountered in the stacked input. Each job must be preceded by a JOB Monitor Control card and followed by an end-of-job (‡‡‡‡) Monitor Control card.

Job Arrangement

A Disk Utility, FORTRAN or SPS job is always represented by at least three Monitor Control records.

Card	Purpose
1. JOB Monitor Control Record	Identify beginning of job.
2. DUP, SPS, SPSX, FOR, FORX, XEQ, or XEQS, Monitor Control record	Transfer control to Disk Utility Program, FORTRAN compiler, SPS Assembler, user's object program.
3. End-of-job Monitor Control record.	Identify end of job.

In addition to the Monitor Control records, there may be one or more Disk Utility Program, SPS or FORTRAN system control records. These records are a part of the input for the individual system.

TYPE or PAUS Monitor Control records may be inserted immediately preceding any of the records in the above sequence. Any number of Comments records may be inserted in front of type 2 records. Source programs or input data can be entered immediately following type 2 records.

The following three points must be taken into consideration when arranging the input for any job.

1. All Monitor Control records, with the exception of Monitor records that follow a TYPE Monitor Control record, must be read from the same input source. The input source can only be changed at the beginning of each job or by a TYPE Monitor Control record.
2. A job, with the exception of a Disk Utility job, may consist of several system functions possibly terminated by execution of a user's program. Execution of a user's program is considered as the end-of-job. If any cards remain in the stacked input for a job when it is ended in this manner, they will be passed without processing. Processing resumes with the first Monitor Control card of the next job in sequence.
3. If an error is detected in an SPS assembly or FORTRAN compilation, the resulting object program or any programs that follow within the job cannot be executed.

Examples of stacked jobs are given in Figure 4 and Figure 5. In these examples:

Job A assembles two SPS source programs and stores the assembled programs in disk storage. The second SPS program is executed after assembly. This job includes Comments cards to instruct the operator and a PAUS Monitor Control card to allow the operator to intervene and change program switch settings.

Job B compiles and executes a FORTRAN source program.

Job C replaces an existing object program in disk storage with a new object program and copies a program from disk-to-disk. Job C calls a program from disk storage with subroutines and executes it.

Monitor Control Record Analyzer Routine

This routine, a part of the Supervisor Program, is used to read the Monitor Control records and Comments records, which are identified by ‡‡ in columns 1-2, and to analyze these records, and to perform the operations or transfer control as directed by the pseudo operation codes. The first Monitor Control record is read from the input source that is specified by the operator when the Supervisor Program is originally loaded into core storage from disk storage to

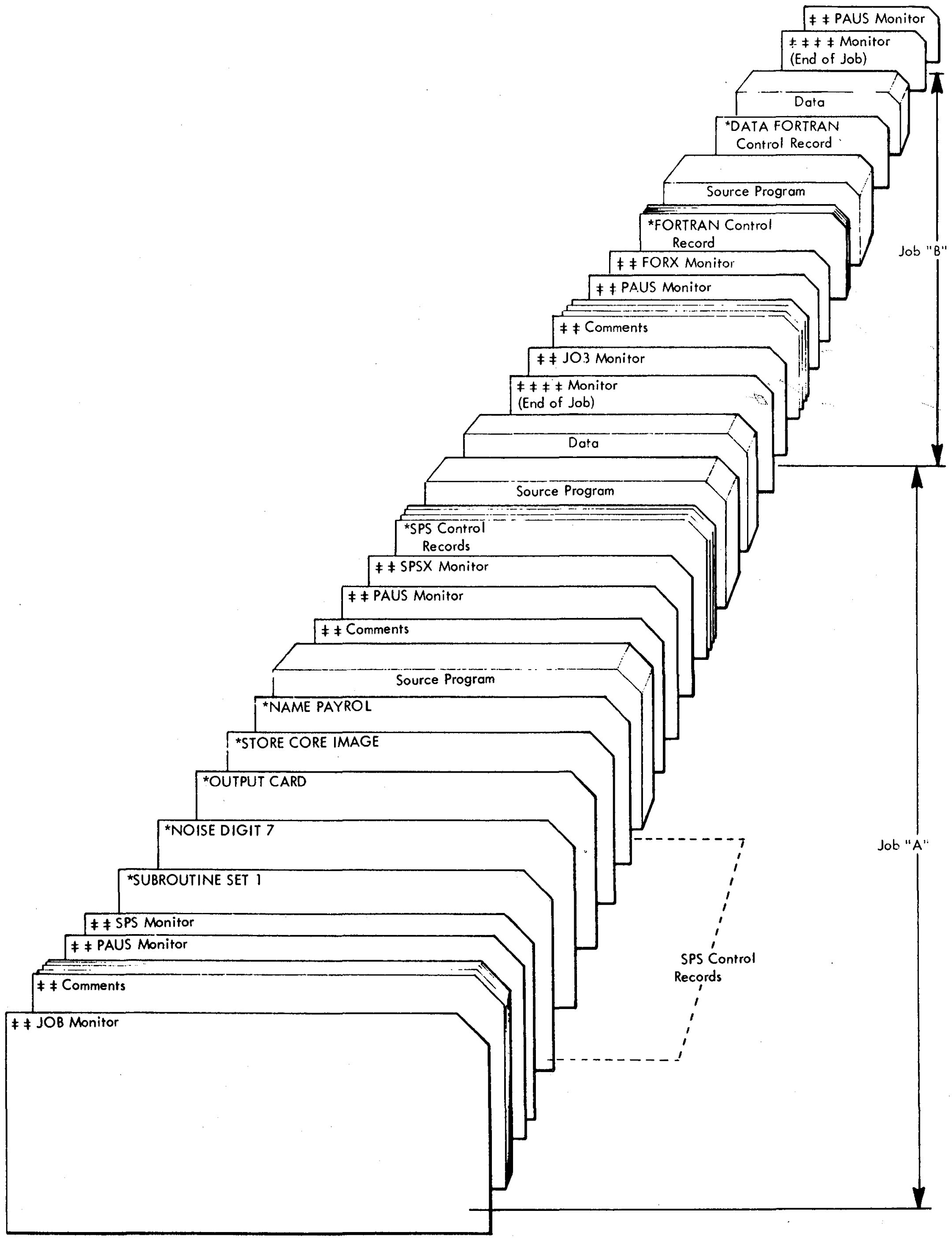


Figure 4. Stacked Input, SPS and FORTRAN Jobs

start the entire operation. Subsequent Monitor Control records and Comments records are read from the same input source until a JOB Monitor Control record changes the input source by specifying a different "source of input" or a TYPE card is encountered. Reading of Monitor Control records continues from the new source until again changed by another JOB Monitor Control record.

When the input source is the "typewriter," the Monitor Control Record Analyzer routine types the message

ENTER MONITOR CONTROL RECORD

The operator may then enter the next control record. The record is not typed out if the entry is made by the typewriter.

If the operator makes a mistake while entering the record, he may correct the error by turning on Program Switch 4 and depressing the R-S key on the typewriter. Switch 4 should then be turned off and the entire record re-entered.

When an SPS, SPSX, FOR, FORX, DUP, XEQ, or XEQS Monitor Control record is read, control is transferred from the Analyzer routine to the individual program specified by the control record. Control is returned to the Analyzer routine after the program is executed. When control is returned, the Analyzer routine will pass records, provided the input source is other than the typewriter, until a Comments or Monitor Control record is encountered in the stacked input. Therefore, the last job to be executed should be followed by a TYPE or PAUS control card. If this control card is not present, the 1620 will stop on a Read Select instruction, expecting another control card.

ERROR MESSAGES

During execution of the Monitor Control Record Analyzer routine, certain error messages may be typed. After typing a message, the 1620 will stop if any operator action is required. A list of these messages, the conditions which cause them, and the corrective actions required of the operator, follows.

Message	ERROR IN FIELD AT COL. XX. SET SW4 TO IGNORE, OFF TO RE-ENTER CARD
Cause	An illegal character has been detected in a JOB Record data field.
Action	To ignore the error turn Program Switch 4 on and depress the Start key. The message "CONDITION IGNORED" is typed and processing continues. To correct the error, turn Program Switch 4 off and depress the Start key. The Monitor Control record input source will be changed to the typewriter, and the operator may then re-enter the control record. If it is desired to read succeeding records from the original input source, column 7 must identify the input source.
Message	PACK NUMBER. ERROR ON MODULE X. SET SSW4 TO IGNORE OFF TO RECOMPARE
Cause	Disk pack identification numbers compare "unequal."
Action	To ignore the error, turn Program Switch 4 on and depress the Start key. The message "CONDITION IGNORED" is typed and processing is resumed. To correct the error, place the correct disk packs on the disk drives and depress the Start key. The disk pack identification number will again be checked by the program. If the pack involved was a Monitor pack, the instruction sequence previously described under *operation* must be repeated.
Message	END OF JOB
Cause	The end of a job has been reached. (This message will not be typed, if the input source is the typewriter.)
Action	None required.
Message	CANNOT RESTORE COMMON —RESET AND START TO RETRY
Cause	Common area does not read into core storage from disk storage correctly.
Action	Depress Reset and Start keys to retry the read operation.
Message	EXECUTION
Cause	Loading and execution of user's object program has started.
Action	None required.
Message	JOB CARD GROUP ONLY
Cause	Control Record Analyzer routine is expecting a JOB, TYPE, or PAUS Monitor Control record, but it does not find one.
Action	Enter JOB, PAUS, or TYPE, Monitor Control record from typewriter and depress the Release and Start keys.

Message	ERROR IN FIELD AT COLUMN XX. PHASE TERMINATED
Cause	A Monitor Control record contains an invalid code in the field starting at column xx.
Action	The phase is skipped and the supervisor will pass records from the control record source until it encounters the next Monitor Control record.

Message	EXECUTION IS INHIBITED
Cause	An error has occurred within a job which may prevent successful execution of the user's object program.
Action	None required. No user object program can be executed until the next JOB Monitor Control record is encountered.

Message	OBJECT DIM ERROR PHASE TERMINATED
Cause	The Supervisor is unable to find the DIM entry specified by an XEQ or XEQS control record.
Action	None required. The phase in which the error occurred is terminated and processing continues.

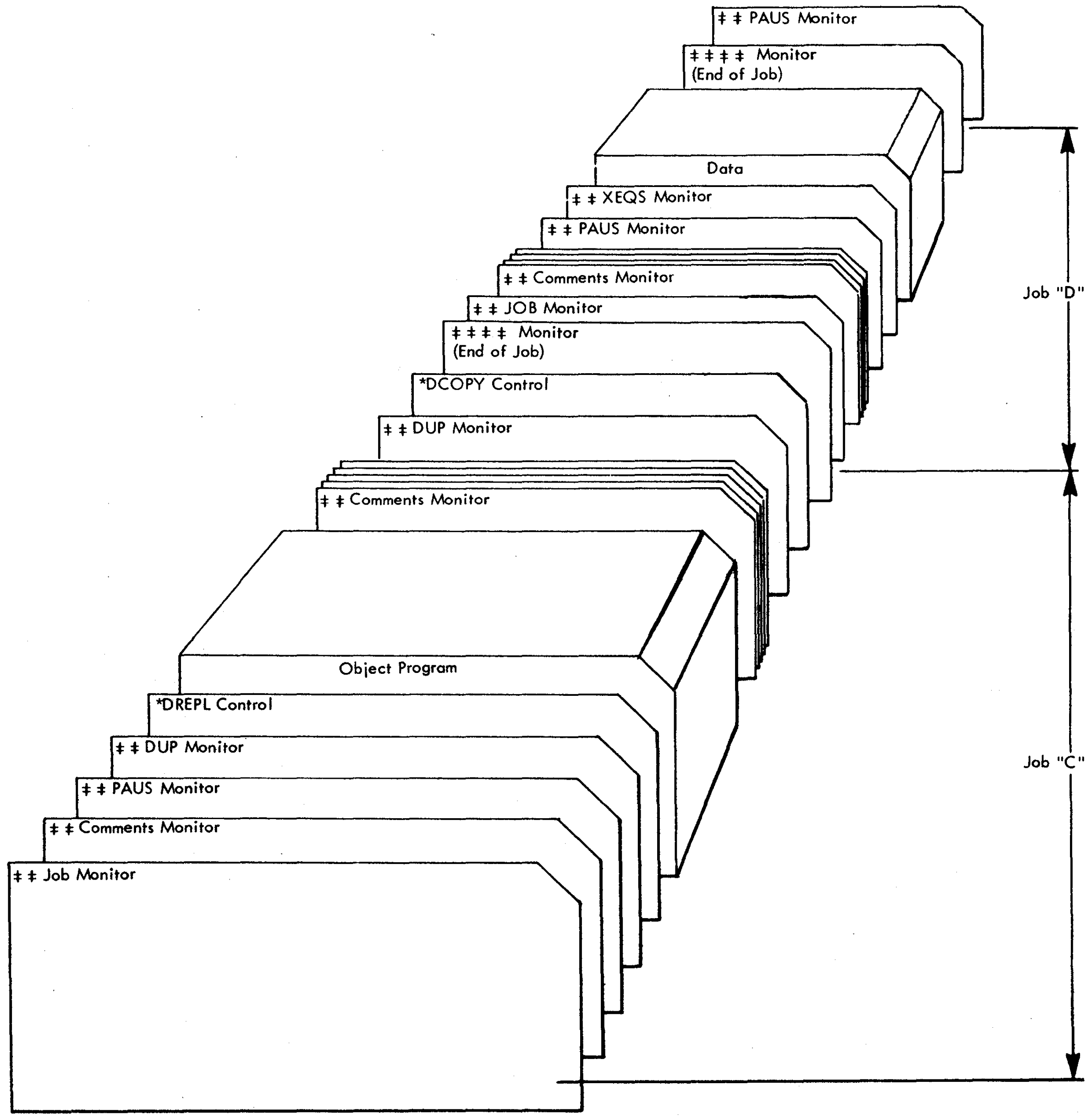


Figure 5 Stacked Input, DUP and XEQS Jobs

Message	OBJECT NAME ERROR PHASE TERMINATED
Cause	The Supervisor is unable to find a name in the Equivalence table which corresponds to the name supplied in an XEQ or XEQS control record.
Action	None required; the phase is terminated.
Message	ENTER MONITOR CONTROL RECORD
Cause	A "‡‡ TYPE" Monitor Control record has been encountered.
Action	Enter a Monitor Control record from the typewriter. (Monitor input source is changed to the typewriter.)
Message	SYSTEM DIM ERROR PHASE TERMINATED
Cause	Supervisor is unable to find DIM entry for SPS assembler, FORTRAN compiler, or Disk Utility Program.
Action	None required; the phase is terminated.

I/O Routine

The I/O routine is designed to relieve programmers of the necessity for writing input/output subroutines. The I/O function is performed automatically by the I/O routine. Therefore, the programmer can concentrate on describing his files and disregard the actual operation of the I/O function. Provision is also made in this routine for error detection and correction. If Parity, Wrong-Length Record Check, or Address Check disk errors occur in a disk operation, the routine will repeat the operation which had the error, up to nine times, in an attempt to correct the error. The Monitor System uses this routine for I/O operations.

The I/O functions performed by the I/O routine include reading and punching cards or paper tape, reading or writing typewriter, reading or writing disk records, and seeking disk cylinders. These functions may be used in an SPS object program by entering I/O macro-instructions (GET, PUT, SEEK, or CALL) in the user's source program. These macro-instructions, as well as the associated declarative statements for defining declarative constants (DTN, DTA, etc.), are described in the section concerning SPS II-D.

All linkages for I/O routines are generated automatically through the use of macro-instructions in SPS source programs or the I/O statements (e.g., FIND, RECORD, FETCH, PUNCH, READ, etc.) in FORTRAN source programs. The data and addresses supplied in a macro-instruction or the parameters in a FORTRAN statement are incorporated into the linkage instructions where they are made available for use by the I/O routine.

Each time the I/O routine is entered as the result of an SPS macro-instruction or FORTRAN statement, the read, write, and parity check indicators are turned off. If a read or write error occurs that cannot be corrected without operator intervention, an error message is typed and the program halts. A restart procedure is specified for all error conditions (see I/O ERROR ROUTINE). An error count is maintained by the I/O routine for inspection by the user or for diagnostic analysis by an IBM Customer Engineer.

In addition to using the I/O routine with SPS macro-instructions and FORTRAN statements, the routine may be used by coding the general form of I/O routine linkage directly in the user's program.

I/O Routine Linkage

General Form.

```
TFM      IORT, * + 23
B        ENTRY, DEF, 7
```

IORT is the address (00565) of a 5-position storage area in the I/O routine.

ENTRY may be any one of the four possible entry points in the I/O routine represented by the following symbolic addresses:

Entry Point	Actual Address	Function
IORBC	00520	Write record into disk storage with Read-Back Check.
IOPT	00532	Write a record to an output device.
IOSK	00554	Seek a disk record.
IOGT	00566	Read a record from an input device.

DEF can be the address of any I/O declarative constant (see I/O CONSTANTS).

CALL LINK or CALL LOAD Linkage. These linkages are usually used to call programs from disk storage, with or without execution. Linkages may be in either a short or long sequence form. Both forms are alike with the exception that the long sequence form contains a relocation address.

Short Sequence

```
TFM         IORT, * + 19
B7          IOCAL
DC          1, M0
DC  ⎫
 or ⎬       1, M1
DSC ⎭
DSC         1, 0
DC          5, IIII @
```

Long Sequence

```
TFM         IORT, * + 19
B7          IOCAL
DC          1, M0
DC  ⎫
 or ⎬       1, M1
DSC ⎭
DC          1, 0
DC          4, IIII
DSA         LLLLL
DSC         1, @
```

IOCAL is an entry to the I/O routine (core storage address 00716).

M_0 M_1 is a constant $\bar{3}2$ for CALL linkages, M_1 is flagged for CALL LINK only.

IIII is the DIM entry number of the program to be called.

LLLLL is the relocation core storage address where the program is to be loaded.

If the short sequence is used to call a relocatable program, LLLLL is assumed by the I/O routine to be the address contained in the "high" indicator field of the Communications Area. If the long sequence is used to call a core image program, the I/O routine will disregard LLLLL.

CALL EXIT Macro-Instruction Linkage.

```
B7          MONCAL
```

MONCAL (core storage address 00796) is an entry to the I/O routine which will call in the Monitor Control Record Analyzer routine.

CALL EXIT linkage is used at the end of the execution of an object program to return control to the Monitor Control Record Analyzer routine to read another Monitor Control record. If, during execution of an object program, an error is encountered which will not allow normal exit to the Analyzer routine, the operator may manually branch the program to MONCAL (00796) to resume processing.

I/O Constants

An I/O constant for card, paper tape, or typewriter consists of eight digits.

$$\bar{C}CCCC\ \bar{M}_0\ M_1\ ‡$$

$\bar{C}CCCC$ is the address of an I/O area.

$\bar{M}_0$ M_1 is one of the following codes which identifies the operation:

$\bar{0}0$	Typewriter Numerical
$\bar{0}2$	Paper Tape Numerical
$\bar{0}4$	Card Numerical
$\bar{0}6$	Typewriter Alphameric
$\bar{0}8$	Paper Tape Alphameric
$\bar{1}0$	Card Alphameric

Disk I/O constants may be in any of the following four forms:

1. $M_0\ M_1\ \bar{D}DDDD$ ‡
2. $M_0 M_1\ \bar{D}DDDD\ \bar{L}LLLL$ ‡
3. $M_0\ M_1$ 0 $\bar{I}III\ \bar{L}LLLL$ ‡
4. $M_0\ M_1$ 0 $\bar{I}III$ ‡

$\bar{D}DDDD$ is the address of the leftmost position of the associated disk control field.

$\bar{L}LLLL$ is a relocation core storage address of a program to be called.

$\bar{I}III$ is the DIM entry number of a program to be called.

M_0 and M_1 provide various disk options for the user. A list of these codes and their associated options follows.

M_0 (code)	Option
0	Add the starting address of the work cylinders from the Communications Area (core positions 422-425) to the sector address in the disk control field. (Used with constant types 1 and 2 only.)
1	Same as option zero, except the "high"indicator in the Communications Area will also be updated for disk read operations only. This indicator is merely a field which contains the core storage address of the highest position to be loaded plus one.
2	Use the sector address in the disk control field for the disk operation (SEEK, READ, or WRITE).

3	Use the sector address in the disk control field for the disk operation. Also, update the "high" indicator in the Communications Area for read operations only.
$\bar{n}$	A flag over the code $\bar{n}$, ($n = 0, 1, 2,$ or 3) causes the read/write heads to be repositioned to an assigned cylinder (specified in the Communications Area) after any disk I/O operation, except seek.

M_1 (code)	Option
0	Disk read or write in sector mode with WLRC. NOTE: The user must place a group mark (‡) in the core storage location following the last character position of the last sector of the record.
2	Disk read or write in sector mode without WLRC.
4	Disk read or write in track mode with WLRC. NOTE: The user must place a group mark (‡) in the core storage location following the last character position of the last sector of the record.
6	Disk read or write in track mode without WLRC.
$\bar{n}$	A flag over code $\bar{n}$($n = 1, 2, 4,$ or 6) causes the I/O routine to branch to a given address after a disk read operation. The given address will be the "execution address" if an extended disk control field is used. Otherwise, it will be the "core address" of the disk control field. If code n is unflagged, the I/O routine will branch to the first instruction following the disk operation calling linkage in the object program. If the entry address is not specified, the entry is made to the (possibly relocated) first card address of the deck to be loaded.

DISK CONTROL FIELD

The disk control field, associated with I/O constants, types 1 and 2, may be in either of the following formats:

D$\bar{\text{D}}$DDDD $\bar{\text{S}}$SS $\bar{\text{C}}$CCCC ‡
D$\bar{\text{D}}$DDDD $\bar{\text{S}}$SS $\bar{\text{C}}$CCCC $\bar{\text{E}}$EEEE ‡

DDDDDD is the first sector address of the data or program.
SSS is the number of sectors to be read or written.
CCCCC is the core address (must be an even-numbered address) of the data or program.
EEEEE is the execution address where program execution is to continue after a disk read operation is completed. The second disk control field, known as an extended disk control field, is used when M_1 of the I/O constant is flagged.

Card I/O

Cards are read or punched in alphameric or numerical form from a user-specified constant (generated from an I/O declarative statement) designated in general linkage. If a punch error is detected during a write instruction, the instruction is again executed to correct the error. If the error persists or a parity error occurs during a write operation, an error message is typed and the program halts (see I/O ERROR ROUTINES). Error messages will be typed for all read errors.

Typewriter I/O

A specified I/O declarative constant designated in general linkage will be used by read or write typewriter instructions (alphameric or numerical). If a read error or a parity error occurs during reading, the program will branch back to the read instruction and await entry of data. The operator can then type in the data and return control to the program. If a parity error or write check occurs during writing, it will be counted and the indicators will be turned off — but the program will not halt. Control operations (RCTY, SPTY, TBTY) are not executed in the I/O routine. These must be handled in the main program coding.

Paper Tape I/O

Paper tape is read or punched in alphameric or numerical form from a specified I/O declarative constant designated in general linkage. If a parity read error occurs during a read operation, an error message is typed and the program halts.

Disk Storage I/O

Disk storage will be read or written as specified by the I/O declarative constant designated in general linkage. Also, disk seek operations will be initiated to disk addresses contained in the I/O declarative constant designated in general linkage. For a CALL macro-instruction, disk data records or programs will be written in the area of core storage designated by the relocatable address in CALL linkage. If this address is not present for a relocatable program, the processor selects the address. If the relocation address is present but the program is not relocatable (i.e., it is in either Absolute or Core Image format), the relocation address is ignored and the program is stored at the core address specified by the DIM entry.

Disk storage indicators are reset by the I/O routine. If the Cylinder Overflow indicator (38) is turned on before the sector count reaches zero, a seek to the next cylinder is initiated and reading or writing is resumed. If read, write, or parity indicators, or indicators 36 or 37 are turned on, the instruction associated with the error will again be executed up to nine times. If the error persists, an error message is typed and the program halts.

The seek only linkage

```
TFM    IORT, * + 23
B      IOSK, DEF, 7
```

will allow computing time and seek time to overlap. DEF refers to any disk I/O constant.

The cylinder location of the arm on each drive is entered into a list by the I/O routine. Every time the I/O routine executes a disk operation, the current entry cylinder is compared to the previous cylinder and the arm is instructed to SEEK only if it is not already located at the current cylinder.

REPOSITIONING OF ACCESS ARMS

The I/O routine contains four 2-digit cylinder indicators that can be used to reposition the access arm on each of the four possible disk drives to a new cylinder following a read or write operation. The four cylinder indicator core storage locations and their associated drives follow:

Indicator Addresses	Drive
00512 - 00513	0
00514 - 00515	1
00516 - 00517	2
00518 - 00519	3

These indicator positions are reset to $\bar{0}0\bar{0}0\bar{0}0\bar{0}0$ by the Monitor Control Record Analyzer routine. Therefore, a program which uses other cylinders for repositioning must provide for changing the indicators.

Repositioning the access arm following a GET or PUT macro-instruction is optional. If M_0 of the I/O constant used by a GET or PUT is flagged, the read/write heads will be repositioned.

FULL TRACK OPERATION

If any I/O operation is to be attempted with the Write Address light on, the programmer must set a flag at OLDDA + 14 (core position 00455) before entering the I/O routine. The flag will prevent accumulating error counts (which is a write disk sector operation). The flag must be cleared before terminating the routine in which the Write Address light "on" condition is present.

If an I/O operation is attempted with the Write Address light on, no flag present at OLDDA + 14, and an indicator 06, 07, 16, 17, 36, 37, or 38 on or turned on by the I/O operation, the program will stop with the instruction at 00728. To save the error count, the operator must (1) turn the Write Address light off, (2) depress the STOP/SIE key, (3) turn the Write Address light on, if desired, and (4) depress the Start key to resume automatic operation.

I/O Error Routine

Each time the I/O routine begins execution, it tests indicator 19 (any check) to determine if an error had occurred prior to entry. If the indicator is on, the I/O error routine will be called into core storage and executed. This routine records a count of errors by type (for indicators 06, 07, 16, 17, 36, 37, and 38), and provides the necessary error messages and corrective operating options. In addition the error routine turns off the individual indicators (06, 07, 16, 17, 36, 37, and 38), by testing them.

After an I/O function is executed, indicator 19 is again tested. If it is on, the I/O error routine is entered to process the error.

Error Detection and Correction

During execution of the I/O error routine, error messages are typed to describe errors and the operator is allowed to intervene to decide how errors should be treated by the program. A list of error correction options available to the operator follows.

Error Correction Code	Option
00	Ignore the error. When this option is used, the I/O routine will finish processing the I/O operation as though the error had not occurred.
05	Re-execute the I/O operation. If an error recurs during the next execution, an error message is

	again typed, the computer stops, and the operator can exercise the same option or another option.
10	Skip this phase of the job if error occurs at system time (SPS assembly, FORTRAN compiling, Disk Utility Program, or Supervisor Program execution time) and return control to the Monitor Control Record Analyzer routine and pass records to the next Monitor Control record.
15	Discontinue execution and return control to the Monitor Control Record Analyzer routine and pass records to the next JOB Monitor Control Record.
20	Continue processing by branching to a specified core storage address without further processing of the I/O request. When this option is exercised, the operator enters the 5-digit branch address from the typewriter.

After each error message is typed, the computer halts. The operator then depresses the Start key, enters a 2-digit error correction code from the typewriter, and depresses the R-S key to resume processing.

If an error is made while entering a 2-digit correction code, it may be corrected by turning Program Switch 4 on, depressing the typewriter R-S key, turning Program Switch 4 off, and re-entering the 2-digit code.

The I/O error routine has the facility to handle any of the following errors.

Entry Check
Typewriter write
Typewriter read
Paper tape write
Paper tape read
Card write
Card read
Cylinder overflow
Write error count
Illegal DIM entry
System
Unavailable disk drive
Control record trap

Error messages, conditions, and corrective operator action associated with each type of error is described as follows:

Entry Check: If indicator 19 (any check) is on, when tested in the preprocessing phase, the message

ENT ERROR $\bar{0}6\bar{0}7\bar{1}6\bar{1}7\bar{3}6\bar{3}7\bar{3}8$

is typed on the console typewriter. Each pair of indicator numbers is flagged in the leftmost digit. If an indicator was on when tested, the rightmost digit will also be flagged.

Typewriter Write: For this error, no error message is typed; however, the error is automatically indicated by over-printing the error character(s) with a horizontal line.

Typewriter Read. For this error, the message

TYP ERR XXXXX $\bar{0}6\bar{0}7\bar{1}6\bar{1}7\bar{3}6\bar{3}7\bar{3}8$

is typed, where XXXXX is the 5-digit return address to the calling program, and the indicators are as described for ENT ERROR. To restart the computer, the operator exercises one of the error correction options.

Paper Tape Write. For this error, the message

PTP ERR XXXXX $\bar{0}6\bar{0}7\bar{1}6\bar{1}7\bar{3}6\bar{3}7\bar{3}8$

is typed, where XXXXX and the indicators are as described above. To restart the computer, the operator should exercise one of the error correction options.

Paper Tape Read. The message

PTR ERR XXXXX $\bar{0}6\bar{0}7\bar{1}6\bar{1}7\bar{3}6\bar{3}7\bar{3}8$

is typed, where XXXXX and the indicators are as described above. The operator must backspace the tape to the beginning of the record before exercising error option 05.

Card Write. For this error, the I/O error routine retries the operation once for a write check error (indicator 07). If the error is corrected by the retry, control is returned to the I/O routine; if the error is not corrected, the message

CDP ERR XXXXX $\bar{0}6\bar{0}7\bar{1}6\bar{1}7\bar{3}6\bar{3}7\bar{3}8$

is typed, where XXXXX and the indicators are as described above. The error option can then be exercised by the operator.

Card Read. For this error, the I/O error routine retries the operation once for a read check error (indicator 06). If the error is corrected by the retry, control is returned to the I/O routine; if the error is not corrected, the message

CDR ERR XXXXX $\bar{0}6\bar{0}7\bar{1}6\bar{1}7\bar{3}6\bar{3}7\bar{3}8$

is typed, where XXXXX and the indicators are as described above. The error option can then be exercised by the operator.

Cylinder Overflow. For this error, the I/O error routine tests to determine if a legitimate overflow has occurred during a disk read or write operation. For a legitimate overflow, a seek operation to the next cylinder is automatically initiated and reading or writing continues. A maximum of three seek operations will be performed if sufficient core storage is available to accommodate the data being read or written.

If the disk read or write operation results in an attempt to read or write data beyond the highest sector address of the addressed disk module, the message

DSK OFL

is typed. All error correction options, except 05, are available to the operator. If 05 was inadvertently entered, it would have the same effect as error correction option 00.

For a disk error, which is other than a legitimate overflow, the disk read or write operation causing the error is retried up to nine times; if this fails to correct the error, the message

DSK ERR XXXXX $\bar{0}6\bar{0}7\bar{1}6\bar{1}7\bar{3}6\bar{3}7\bar{3}8$

is typed, where XXXXX is the 5-digit return address to the object program. A flag is typed over the leftmost position of each pair of indicator numbers. The indicator(s) which identifies the type of error will also be flagged in the rightmost position. If indicator $\bar{3}8$ is flagged in its rightmost position, it may mean either of two things:

1. A legitimate overflow did occur, but another type of error occurred in attempting to transmit data to or from the succeeding cylinder.
2. A machine malfunction occurred.

If the operator exercises error correction option 05, but this does not correct the error, he should turn the Disk, Parity, and I/O Check switches to STOP, and again exercise option 05. The console lights may then be examined to determine the nature of the error.

If a legitimate cylinder overflow condition occurs, a seek operation to the next cylinder is automatically initiated and reading or writing continues.

Write Error Count. If an error occurs while the I/O error routine is writing the error count in disk storage, the message

BAD DISK WRITE, RESET START

will be typed. In this case, the operator does not exercise an error correction option, but he must:

1. Clear the Select-Lock light, if it is on.
2. Depress the Reset and Start keys.

Illegal DIM entry. If the user supplies an illegal DIM number (not in DIM table) in a CALL statement, the I/O routine will transfer control to the I/O error routine and the message

MAP ERR XXXXX IIII

is typed, where xxxxx is the core storage position immediately following the call linkage, and IIII is the illegal DIM entry number. The operator then enters error correction code 00 and depresses the Release and Start keys. The computer will again halt. The operator must then type in a corrected 4-digit DIM entry number and depress the Release and Start keys.

System. If the I/O error routine cannot interpret the nature of the error, the message

IMP ERR

is typed and control is returned to the Monitor Control Record Analyzer routine without stopping to allow operator intervention.

In addition to the system error just described, the computer may halt in the I/O routine at core storage address 00467 without typing a message. This halt occurs if a read error occurs while the I/O routine is reading one of its subroutines from disk storage. To retry the operation, the operator should

1. Clear the Select-Lock light if it is on.
2. Depress the Reset and Start keys.

If this error persists, it may mean either of two things.

1. The user inadvertently altered the I/O routine in core storage.
2. A machine malfunction occurred.

Unavailable Disk Drive. If the programmer specifies a logical module for which there is no physical disk storage drive, the message

MOD ERR XXXXX

is typed, where xxxxx indicates the return address to the object program. The operator must enter the error correction code 00 and depress the Release and Start keys. The computer will again halt. The operator must then enter a corrected 1-digit drive code and depress the Release and Start keys to continue.

Illegal Drive Code. If the user gives an illegal drive code in the disk control field for a disk operation, the message

MOD ERR XXXXX

is typed, where xxxxx is the 5-digit return address to the object program. To continue, the operator should enter an error correction code 00 and depress the Release and Start keys. The computer will halt to allow the operator to enter a corrected 1-digit drive code from the typewriter. Depress the Release and Start keys to resume operation.

Control Record Trap. To prevent the I/O routine from inadvertently reading a control record as a data record, the I/O routine is designed to trap control records, if they are read from the Supervisor input source. Each record read is tested for ‡ ‡ in its first two positions. If present, control is transferred to the I/O error routine and the message

TRP ERR

is typed. If the control record was read in numerical mode and it was not an end-of-job record (‡‡‡‡), an additional message is typed:

MUST RELOAD

The operator then depresses the Start key and re-enters the record. The Monitor Control Record Analyzer routine assumes control and processes the trapped control record. If the control record was read in alphameric mode, it is processed in the normal manner by the Supervisor.

Error Count Retrieval Routine

Each time an error is detected by the I/O error routine, an error count is incremented by one. An error count is maintained for each of the following error indicators:

06	Read Check
07	Write Check
16	MBR-E Check
17	MBR-O Check
36	Address Check
37	WLR-RBC
38	Cylinder Overflow

The error counts can be typed out and reset to zeros by entering the following instructions and data from the typewriter:

```
34      00032 00701
36      00032 X0702
49      00070 0
11975400100046 (disk control field)
```

X is the drive code for the Monitor I System.

The Release and Start keys are depressed to start the operation. The seven indicator counts are then typed in sequence in 14 consecutive positions with a flag over the leftmost position of each count.

$\bar{X}X\ \bar{X}X\ \bar{X}X\ \bar{X}X\ \bar{X}X\ \bar{X}X\ \bar{X}X$

The error counts are reset to zeros after the typeout.

Loader Routine

The Loader routine, a part of the Supervisor Program, is used to load user's object programs into core storage from cards, paper tape and disk storage. To perform the loading function, the Loader routine is called into core storage whenever an object program is to be loaded into core storage. The user's object program could be any program in System Output format.

Programs are sequence checked as they are loaded if input is from cards. This check is performed on the last five digits of each input record. If any records are out of sequence, an error message is typed and the operator is allowed to intervene to correct the sequence error. Patch cards may be interspersed with other cards of an object program to be loaded.

The sequence number of card input appears in columns 76-80. Sequence numbers start with $\bar{0}0001$ and must have a flag over their leftmost position in order to be sequence checked.

System Output Format

The general format in which FORTRAN II-D and SPS II-D object programs will be outputted to paper tape, disk, and cards is shown below:

Columns	1-5	Address of data.
	6	Indicator code.
	7-8	Length of data.
	9-75	Data, indicator codes, etc.
	76-80	Sequence number.

Patch cards should be prepared in the same format. However, the sequence number must be all zeros without flags. All patch cards must precede the card that defines the end of a relocatable program (see INDICATOR CODE 6). The entries shown above are described as follows:

NOTE: The descriptions given here will be in terms of cards; however, paper tape and disk formats will be the same with the exception of the sequence number.

Address of Data – This entry will always refer to the location where the first digit of data on the card is to be loaded. This entry will appear in columns 1 through 5 of a reloadable card. The address is an absolute address, i.e., no relocation increment (presuming this program is relocatable) has been added yet.

Indicator Code – This 1-digit entry is used to either define the type of data that is to follow or to convey certain loading instructions to the loader. There are thirteen different indicator codes that may be used; some are applicable to SPS II-D only (see INDICATOR CODES).

Length of Data – This field is used in conjunction with certain indicator codes (1, 2, $\bar{2}$, 3, $\bar{3}$, 4, $\bar{4}$) to specify how many digits of data are to follow. With other indicator codes, this field becomes part of a larger field and assumes a different role.

Data – This field contains actual data to be loaded. Data may be instructions, constants, etc., depending upon the indicator code. All instructions for relocatable programs will contain flags over O_0, O_1 of the operation code to specify if the P and Q addresses, respectively, should be incremented by the relocation address. If patch cards are prepared, these flags must be punched for addresses to be adjusted. Instructions of programs in absolute format must not be flagged.

INDICATOR CODES

Although the output format is shown divided into specific fields, these same fields do not always make up the columns that are indicated. As will be seen by the descriptions of the various indicator codes, the format varies considerably as the type of data on the card changes.

The indicator codes are described as follows:

NOTE: Those codes marked with an asterisk are used in SPS II-D output only.

$\ddagger$ – This digit indicates that a change is being made in the sequence of loading addresses for the program. The five digits that follow the record mark denote the new address or origin. After the 5-digit address, there will be another indicator code to define the data that follows.

$\bar{\ddagger}$ – This digit is used whenever the data that follows is an instruction or relative address which cannot be fully contained before the seventy-fifth column of the card has been reached.

* 0 – This digit is used when a TRA-TCD declarative combination is assembled in any relocatable SPS program. The five digits that follow the zero constitute a branch address for entrance to a routine.

* $\bar{0}$ – This digit is used in the same manner as 0 above, except the flag denotes an SPS program with absolute addresses.

1 – This digit indicates that the data to follow after the "length of data" field are instructions.

2 – This digit indicates that the data following the "length of data" field are constants that are to be relocated.

$\bar{2}$ – This digit indicates that the data following the "length of data" field are constants that are *not* to be relocated.

3 – This digit indicates that the data following the "length of data" field are relative addresses to be relocated.

* $\bar{3}$ – This digit indicates that the data following the "length of data" field are relative addresses that are *not* to be relocated.

* 4 – This digit is used to supply numeric blanks when a relocatable program is loaded. The 2-digit "length of data" field following the indicator specifies how many numeric blanks are desired. Thus a $4\bar{1}2$ will cause twelve numeric blanks to be inserted into core storage when loading.

* $\bar{4}$ – This digit is used in the same manner as the digit 4 above except that the flag indicates the program is in "absolute" form.

6 – This digit indicates the end of a relocatable program. In SPS II-D, the five digits immediately preceding a 6 or $\bar{6}$ (described below) will be the number of core positions needed for this program. In SPS or FORTRAN, the card that follows a card containing a 6 or $\bar{6}$ will contain five 9's in columns 1-5.

* $\bar{6}$ – This digit indicates the end of an "absolute" program.

EXAMPLE

The first 35 columns contain the following:

$$\bar{0}10121\bar{1}2\bar{4}90036600000\ddagger\bar{0}10192\bar{0}5\bar{0}042\bar{8}\bar{\ddagger}$$

Columns 36 through 75 are blank and columns 76-80 contain $\bar{0}0101$.

Explanation

Column	*Contents*	*Description*
1-5	$\bar{0}1012$	Loading address of first information.
6	1	Code indicating the following information is an instruction.
7-8	$\bar{1}2$	Length of the following information.
9-20	$\bar{4}90036600000$	The information to be loaded, which is a Branch instruction with a relocatable P field.
21	$\ddagger$	Code indicating a change in the loading address sequence.
22-26	$\bar{0}1019$	Loading address of the following information.

27	2	Code indicating the following information to be loaded is a relocatable constant.
28-29	$\bar{0}5$	Length of the following information.
30-34	$\bar{0}042\bar{8}$	The information to be loaded, which is a relocatable constant.
35	$\overline{\ddagger}$	Code indicating nothing further is to be loaded from this card.
76-80	$\bar{0}0101$	Sequence number.

With a relocation factor of 14000 the above data would be loaded starting at 15012 as $\bar{4}914366\bar{0}042\bar{8}$.

Error Messages

If an error occurs during execution of the Loader routine, an error message will be typed and the 1620 will stop to await operator action. A list of error messages, the conditions which cause them, and the corrective action required, follows.

Message – XXXXX LD1 (XXXXX is the sequence number of the last card read in correct sequence).
Cause – Card sequence error.
Action – Correct the order of input cards, starting with the card following XXXXX, and place them in the card reader. Depress the Start key.

Message – LD2
Cause – Card read error.
Action – Reread card by depressing the Check Reset and Start keys on the card read punch.

Message – LD 3
Cause – Disk read error.
Action – Depress Start key and retry.

Message – LD 4
Cause – Disk read error while reading Loader routine into core storage.
Action – Depress Start key to retry.

Monitor I System Communications Areas

Core storage positions 402 through 439 and disk sector 19663 are reserved for use by the Supervisor Program, SPS assembler, FORTRAN compiler, Disk Utility Program, and other programs as common communications areas. The Communications Areas are automatically established when the Monitor I system is loaded. Changes to the communications areas are made as specified by control records (see DFINE CONTROL CARD) or by the Supervisor program itself. Care should be taken by the user, so that the communications areas are not inadvertently altered. A description of each of the fields in the core storage and disk sector communications areas follows.

Core Storage Area

Core Storage Positions	Description
402-421	A 20-digit DIM entry or a 14-digit disk control field being used by the I/O routine, Disk Utility Program, or other programs.
422-425	Starting address of work cylinder. Only the four leftmost positions of the sector address are given. This address will be $1\bar{0}00$ (with the flag) unless changed by a DFINE control card.
426	Source of SPS or FORTRAN source program input, 1 = typewriter. 3 = paper tape. 5 = card.
427	If this position is flagged, loading resumes after a TCD at core storage address 00000. If unflagged, loading resumes at the core storage address specified by positions 435-439 of this Communications Area.
428	Source of object program being loaded, 3 = paper tape. 5 = card. 7 = disk. A flag in this position indicates that a DEND type entry starts execution of the object program. No flag indicates that a TRA-TCD type entry starts execution of the object program.
429	Source of monitor control input, 1 = typewriter. 3 = paper tape. 5 = card.

	A flag is present in this position if library subroutines are to be called with SPS or FORTRAN objects programs.
430-434	"High" indicator, i.e., the core storage address of the highest position to be loaded plus one.
435-439	Address where loading is resumed following an SPS TRA statement. This address will always be one of the following: $\bar{0}$0000, $\bar{0}$0075, $\bar{0}$0150, or $\bar{0}$0225.

Disk Sector Area (Sector 19663)

Disk Sector Positions	Description
00-19	DIM entry used by I/O routine and Supervisor program.
20-21	Not Used. Available for use by the 1620 user.
22	$\bar{0}$ indicates that the program to be loaded into disk storage is in core image format; $\bar{1}$ indicates that the program to be loaded into disk storage is in relocatable format.
23	$\bar{0}$ indicates card output; $\bar{1}$ indicates paper tape output.
24-35	Six-character alphabetic name of user's source program to be loaded into disk storage after assembly.
36-39	Four-digit DIM entry number of user's source program to be loaded after assembly.
40-41*	Two digits ($\bar{x}$x) indicating length of mantissa for SPS subroutines. (Standard mantissa length is 08.)
42-43*	Two-digit SPS subroutine set identification number. (Standard set number is 02.)
44 *	N (noise) digit for SPS subroutines. (Standard N digit is 0.)
45-46*	Two digits (ff) indicating length of mantissa for FORTRAN subprograms. (Standard mantissa length is 08.)
47-48*	Two digits (kk) indicate FORTRAN fixed-point word length (04 standard length).
49	Digit indicates number of disk storage drives available to the Monitor System.
50-72	Supervisor Program indicators.
73	Source of input, other than disk, for FORTRAN subprograms (from DFINE control record; 5 is standard, 3 = paper tape, 5 = card).
74-75	Number of control cards for FORTRAN at load time.
76	Object machine core size (from DFINE control record; 1 when system is delivered).
77	N (noise) digit for SPS subroutines (from *Noise Digit* control record).
78-79	Mantissa length for SPS subroutines (from *Mantissa Length* control record).
80-81	SPS subroutine set identification number (from *Subroutine Set* control record).
82	FORTRAN A and I/O subroutine set numbers (from FORX or XEQS control record).
83*	FORTRAN A and I/O standard subroutine set number (from DFINE control record; 1 when system is delivered).
84-88	First core storage address of a relocatable object program.
89-93	Computed relocation address of a relocatable object program.
94-98	Card sequence number.
99	A record mark (‡).

* These items are the systems standards. See *Define Parameters* under Disk Utility Program.

Disk Utility Program

In every data processing installation there are certain operations that must be performed frequently. These operations may differ in detail, depending on the user's particular machine configuration and data format, but essential functions remain the same. Because of their frequent use, the burden of programming these operations can become a costly, time-consuming task. Therefore, there is a need for generalized routines which will satisfy specific functions and allow the user the flexibility of assigning the specifications for his particular problem.

The generalized routines, provided by IBM Programming Systems, described in this publication, are grouped under the heading Disk Utility Program. They are designed to assist the user in the day-to-day operation of his installation. By means of these routines, certain frequently required operations, such as loading or unloading disk storage (data or programs) from cards or paper tape, etc., can be pérformed with minimum programming effort by the user.

The routines described in this publication are:

1. *Write Addresses.* This routine writes sector addresses on a disk pack as specified by the user. Data on the disk pack can be replaced by zeros or left unchanged.
2. *Alter Sector.* This routine uses the typewriter to change data in a sector of disk storage. In most cases, only the digits to be changed must be typed.
3. *Disk-to-Output.* This routine unloads disk storage containing data or programs into cards, paper tape, or on the typewriter.
4. *Load Programs.* This routine loads one or more programs from cards or paper tape to disk storage at either a specified address or an address selected by the load routine itself, and checks for an overlap of previously stored programs.
5. *Replace Programs.* This routine implements the changes or additions necessary to update a program on disk storage. Input can be in either card or paper tape form.
6. *Disk-to-Disk.* This routine copies data or programs from one area of disk storage to another.
7. *Delete Programs.* This routine effectively deletes programs from the system by deleting their associated DIM entries and Equivalence table entries without actually removing the programs themselves.
8. *Define Parameters.* This routine redefines certain essential parameters of the 1620 Monitor I System.
9. *Define Disk Pack Label.* This routine writes the "label sectors" (first and last sectors, cylinder 99) and establishes a Sequential Program table on a disk pack. It can be used to initialize new disk packs.
10. *Define FORTRAN Library Subroutine Name.* This routine generates an entry in the Equivalence table for FORTRAN subroutines that have multiple entries. Thus, a name can be assigned to all entries in a subroutine.

    Each routine can be entered and executed by means of control records read by the Disk Utility Program. In addition, the routines are used by both SPS II-D and FORTRAN II-D to output assembled programs into cards or paper tape and to load and replace programs in disk storage.

The Equivalence table, DIM table, and Sequential Program tables are used and modified by the Disk Utility Program in the execution of its routines. These tables are updated automatically for each disk storage change when the user adds, deletes, or replaces a program. Entries are created in the tables whenever a new program is loaded to disk storage.

Operation

The Disk Utility Program, a self-loading program, is loaded into disk storage along with the other programs that make up the Monitor I System. When a DUP Monitor Control card (‡‡ DUP in card columns 1-5) is recognized by the Supervisor Program, the Disk Utility Program will take control and select the appropriate Disk Utility routine as identified by the next card in seqence, which should be a Disk Utility Program Control card. This card is identified by an asterisk in card column 1. Card columns 2-6 contain a code word to identify the Disk Utility routine desired: such as, Alter Sector, Load Programs, etc., and the remaining card columns provide additional control information to be used by the Disk Utility routine itself. The user supplies the control information which describes the function he desires. Because the control information for each type of Disk Utility Program Control card is different, the format of each is described separately in the separate routine descriptions. After the execution of a Disk Utility routine is completed, control is returned to the Monitor Control Record Analyzer routine.

A DUP Monitor Control card, as well as a Disk Utility Program Control card, is required each time a Disk Utility routine is to be executed. These cards are stacked with the other input cards to be processed by the Monitor I System. This stacked input may be in card or paper tape form or it may be entered from the typewriter.

If the code word contained in a Disk Utility Program Control card is not one of the ten legitimate codes (DWRAD, DALTR, DDUMP, DLOAD, DREPL, DELET, DFINE, DCOPY, DLABL or DFLIB) an error message will be typed and the computer will halt. This message will be comprised of the data from the control card and a constant, ERR CONTROL. When the Start key is depressed, the Disk Utility routine will return control to the Monitor Control Record Analyzer routine which will pass all cards until the next Monitor Control card is reached.

If the control record is entered from the typewriter, the message

ENTER DUP CNTRL REC

is typed and the computer halts. The user may then enter the next Disk Utility Control record from the typewriter and depress the R-S key to continue processing. Records are entered in the alphameric mode

The Disk Utility Program uses the I/O routine of the Supervisor Program to perform its I/O functions Therefore, error messages associated with that routine will be typed if an I/O error occurs. The I/O error messages, as well as operating options for I/O errors, are described under, I/O ERROR ROUTINE, in the Supervisor Program section.

Whenever a Disk Utility routine is instructed by a control record to write read-only flags with sector addresses, the message

DUP * TURN ON WRITE ADDRESS KEY, START

is typed and the program halts. The operator should turn on the Write Address key (to allow read-only flags to be written) and depress the Start key to continue processing. After the program has been completely loaded, the message

DUP * TURN OFF WRITE ADDRESS KEY, START

is typed and the program again halts to allow the operator to turn off the Write Address key.

Whenever a program of less than 200 sectors is assigned to disk by any Disk Utility routine, it will always be placed in consecutive sectors of one cylinder. However, a program can be assigned by the user, to any available disk storage area as described under LOAD PROGRAMS ROUTINE.. Programs as large as 999 sectors long can be processed by the Disk Utility Program.

After a program is loaded to disk by any Disk Utility Program routine, the message

DK LOADED AAAAAA IIII D$\bar{D}$DDDD $\bar{S}$SS $\bar{C}$CCCC $\bar{E}$EEEE ‡

is typed to inform the user about the assigned DIM entry.

AAAAAA is the assigned program name, IIII is the assigned DIM entry number and the remainder of the message is the DIM entry itself.

Write Addresses Routine

The Write Addresses routine is used to write sector addresses on a disk pack. Addresses may be written with or without read-only flags over the leftmost positions. Data positions of each sector may be changed to zeros or left unchanged.

When the Write Addresses routine is executed, the Write Address key must be turned on. The message:

DUP * TURN ON WRITE ADDRESS KEY, START

is typed to signal the operator to turn the switch on. After the routine has been executed, the message:

DUP * TURN OFF WRITE ADDRESS KEY, START

is typed to signal the operator to turn the switch off.

The format of the control card follows.

Control Card (DWRAD).

Columns	1	Asterisk (*)
	2-6	Code word, DWRAD.
	7-12	Disk sector address where writing is to start (seek address).
	17	Letter P if read-only flags are to be written over addresses; otherwise leave blank.
	18	Letter Z if data positions are to be changed to zeros; letter S if data positions are to remain unchanged.

21-26	Address to be written at sector where writing is to start.
27-32	Final address to be written.

When the DWRAD Control card is read, control is transferred to the Write Addresses routine and the message

```
WRITE AND SAVE
  SEEK    START    STOP
X̄XXXXX  X̄XXXXX  X̄XXXXX
```

or the message

```
WRITE AND ZERO
  SEEK    START    STOP
X̄XXXXX  X̄XXXXX  X̄XXXXX
```

is typed to allow verification of control record data and the computer is halted. (Note that the second form of the message indicates that sector data positions are to be changed to zeros.) Six X's indicate the respective seek, start, and stop addresses. Depressing the Start key causes execution of the routine. The routine seeks the disk sector address specified in columns 7-12. The address specified in columns 21-26 is written in that sector; the address is then incremented by one and written in the next sector. Writing continues in this manner until the incremented address is equal to the final address (columns 27-32) and the final address has been written.

If the program is unable to find the starting address (columns 7-12), or any address that should be on the specified track in disk storage, an error message ER SK XXXXXX will be typed. XXXXXX is the disk address on the last sector examined when no equal comparison could be made with the sector addresses that should be on the track that has been read. In addition, the 20 sector addresses from the selected track will be typed and the program will halt. When the Start key is depressed, control is returned to the Monitor Control Record Analyzer routine to read the next Monitor Control record.

Alter Sector Routine

This routine allows the user to alter the data in any selected sector of disk storage. The sector data to be changed is typed out. All, or selected portions of the sector may then be updated. After the changes have been made, the old and the new data are typed out for visual comparison and verification. If the changes are satisfactory, the new data is stored on the disk pack. As many sectors as desired may be altered each time this routine is used. Control is transferred to the Alter Sector routine when the control card is read.

Control Card (DALTR).

Columns	1	Asterisk (*)
	2-6	Code word, DALTR.

After the control card is read, the message

```
SECTOR
```

is typed and the program halts.

The operator types in the 6-digit address of the sector to be altered and depresses the Release and Start key. If more or less than six digits are typed, the message

```
SECTOR ADDRESS ILLEGAL,
START TO RE-ENTER *DALTR
```

is typed and the machine halts. Pushing the Start key will restart all operations on the given sector. The routine reads the sector and types it out in the following format.

```
1st Half  xxxxxxxxxx xxxxxxxxxx xxxxxxxxxx xxxxxxxxxx xxxxxxxxxx ORIGINAL
2nd Half  xxxxxxxxxx xxxxxxxxxx xxxxxxxxxx xxxxxxxxxx xxxxxxxxxx ORIGINAL
```

Note that the two halves of the sector are identified by the phrases, "1st Half" and "2nd Half," respectively. (Typewriter margins must be at least 70 spaces apart to permit this format.)

Each group of ten characters is assigned a section number by the routine. The first five groups are assigned numbers 01-05; the last five groups are assigned numbers 06-10. After the sector data is typed out, the routine requests the number of the section in which the first change will be made.

The message typed out is shown below:

```
SECTION
```

The user now types in one 2-digit section number between 01 and 10. After depressing the Release and Start key, the message

```
SECTION NUMBER ILLEGAL,
START TO RE-ENTER *DALTR
```

is typed if the section number is greater than 10. If the section number is correct (between 01 and 10), the selected section is typed out for verification as shown below.

```
XXXXXXXXXX  TYPE  CHANGE
```

The changes can now be entered directly under the typed section. If a particular character does not require changing, an "x" may be typed under that character, or the character itself may be retyped. Although only one section is typed out for any one selection, succeeding sections may be altered by continuing to type changes. Spacing is optional except that the number of characters (including spaces) cannot exceed 100. Spaces (alpha blanks) will not become part of the sector data. For example, assume that section 3 is selected and is typed out as shown below:

```
3574246798
```

The operator desires to make some changes in section 3 and section 4. For this example, the typewritten page may look like this:

```
3574246798 TYPE CHANGE
XXX7625X82 75234XX479
```

Typing may be terminated as soon as the last digit to be changed is typed; that is, if the fifth digit in section 4 (previous example) is the last change, the last five digits of section 4 do not have to be typed in.

If the user does not type changes but simply depresses the Release and Start key, the message

CORRECTIONS HAVE NOT BEEN ENTERED

is typed and the computer will halt on a read alphameric instruction to allow the user to enter the changes.

If more digits are entered than the sector can contain, the message

TYPE-IN EXCEEDS SECTOR LENGTH, START

is typed. Depressing the Start key allows the operator to begin again and enter a new sector address. Spaces are optional and do not become part of the sector data; however, they are counted toward the maximum allowable number of characters which is 100.

After all changes have been made, the operator depresses the Release and Start key. The routine then types out the original sector data along with the changes that were typed in. The output appears as shown below:

```
1st Half  1234567890 1234567890 3574246798 8654213212 0987654321 ORIGINAL
1st Half  1234567890 1234567890 3577625782 7523413479 0987654321 CORRECTED
2nd Half  7265417623 0176421432 8543217290 5482797654 8243176521 ORIGINAL
2nd Half  7265417623 0176421432 8543417290 5482797654 8243176521 CORRECTED
```

At this point the routine will again type the word

SECTION

If other changes must be made, the operator enters a new (or possibly the same) section number and depresses the Release and Start key. The SECTION change routine is now repeated. When all the changes prove satisfactory, the operator enters a record mark instead of a section number and depresses the Release and Start key. The routine then writes the updated sector back on the disk pack and types the message

DISK SECTOR DDDDDD CORRECTED

DDDDDD is the sector address that was selected to be changed. The routine then branches to the part of the routine that types the message "SECTOR" to allow the user to choose another sector address and change another sector.

When all desired sectors have been altered, this routine is concluded by typing a record mark instead of a sector address after the word SECTOR has been typed out. This will cause control to be returned to the 1620 Supervisor Program, which will read another 1620 Monitor I control statement.

Operating Notes

When the routine is ready to accept the new data (after the section number is typed in), it positions the console typewriter in the "alphameric shift" mode. Therefore, typing numerical data requires the operator to manually shift into numerical mode.

Flagged digits 1-9 may be inserted by typing the corresponding alphabetic letters J-R. Flagged zeros, numeric blanks, flagged record marks, and flagged group marks may be entered by using minus (—) key, @ key, W key, and G key, respectively. Alpha blanks (spaces) do not become a part of the sector data.

Disk-to-Output Routine

The Disk-to-Output routine transfers data from selected portions of disk storage to cards, paper tape, or the typewriter. This routine enables the user to preserve original records before they are updated or changed, thus providing an audit trail. The card or paper tape output will be in numerical form and will contain record marks and group marks. Numerical blanks result in blank card columns.

This routine can be used to obtain any of the following items of output as directed by the user.

1. Program or data identified by name.
2. Program or data identified by DIM number.
3. Data between sector limits.
4. DIM table.
5. Equivalence table.
6. Availability list (extracted from Sequential Program table).
7. Sequential Program table.

The routine is executed whenever a DDUMP control card is read by the Disk Utility Program or wherever a FORTRAN compilation or SPS assembly requires punching into cards or paper tape.

The Disk-to-Output routine can be used to transmit any number of disk sectors to cards, paper tape, or typewriter. Transmission will start with the first sector specified in a DIM entry or with a beginning sector specified by the user. Transmission will end when the sector count in the DIM entry reaches zero or when a specified ending sector is found. The output following compilation or assembly is terminated by a "9's" trailer record. The trailer record format is five 9's followed by a record mark, 69 zeros and a sequence number. This record always follows all SPS and FORTRAN object programs. During execution of the Disk-to-Output routine, resulting from DUP control records, error messages 01, 04, 06 or 20 may be typed (see ERROR DETECTION AND CORRECTION).

The control card used to transfer control to the Disk-to-Output routine is punched in the following format.

Control Card (DDUMP).

Columns	1	Asterisk (*).
	2-6	Code word, DDUMP.
	7-12	Alphabetic name of program or data to be punched or typed (same name that appears in Equivalence table).
	13-16	DIM entry number of programs to be punched or typed. (If the letter M is present in column 18, either a name or DIM entry number must be present, but both need not be present.)
	17	Output device, C = card, P = paper tape, T = typewriter, L = PRINTER
	18	Identify output: I = Disk Identification Map (DIM). E = Equivalence table. A = Avalability entries from the sequential program table from the disk module specified by column 19 (typed output only). S = Entire sequential program table from the disk module specified by column 19. M = Program identified by columns 7-12 or 13-16 of this card. L = The sectors between limits as specified by columns 21-26 and 27-32 of this card.
	19	Module number (0, 1, 2, or 3) to be used if output options S or A are exercised (see column 18 above).
	21-26	Beginning disk storage address of output (lower limit).
	27-32	Ending disk storage address of output (upper limit).

Output Format

CARD

Each 300 positions of disk storage (three sectors) will be punched into four successive cards; 75 columns of disk data followed by a five-column sequence number in each card. When 2 sectors are to be outputted, 3 cards are punched. When 1 sector is outputted, 2 cards are punched. Therefore, all disk data is punched from 1 or 2 sector outputs.

A special trailer card containing 9's in columns 1-5, a record mark in column 6, zeros in columns 7-75, and a sequence number in columns 76-80 will be punched following the last output card. This record is used to terminate loading when the output is reloaded by the Load Programs routine or Replace Programs routine. If the output deck is reloaded by the Load Programs routine or Replace Programs routine, the trailer card must remain behind the deck for control purposes.

The 9's trailer record will load into the work cylinders along with the balance of the data; however, the trailer record will not require extra disk storage positions if the data is moved to another disk location. If the output is a program to be reloaded by the loader routine, the entire program must be outputted. The System Output Loader routine requires an entire program, with all of its indicator codes, in order to operate.

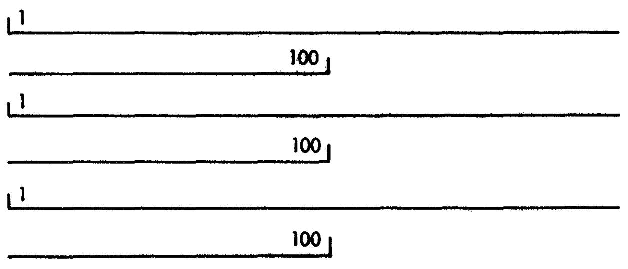
When either the DIM table or Equivalence table is punched out, they will be in a loadable format; i.e., alphameric characters will be in 2-position alphameric coding form.

PAPER TAPE

This output will be in standard loadable format, i.e., it may be reloaded to disk storage by the Replace Programs routine. The output will be identical to the card format described above, except that the sequence number will not be punched with paper tape records. The last output sector will be followed by a trailer record for control in the event that the output is reloaded into disk storage.

TYPEWRITER

With the exception of the Availability list and Equivalence table, all typewriter output will be in a standard format. Each 100-character sector will be typed on two lines as shown below.



Availability Lists. Availability lists are typed as follows:

```
AAAAA
BBBBB CCCCC
BBBBB CCCCC
BBBBB CCCCC
```

AAAAA is the disk pack identification number.
BBBBB is the starting disk address of an unused area of storage.
CCCCC is the ending disk address for the unused area.

These entries, one per line, are extracted from the Sequential Program table by the Disk-to-Output routine.

Equivalence Table. Equivalence tables are typed in the following format with five entries per line.

```
NNNNNN ĪIII NNNNNN ĪIII NNNNNN ĪIII NNNNNN ĪIII NNNNNN ĪIII
NNNNNN ĪIII NNNNNN ĪIII NNNNNN ĪIII NNNNNN ĪIII NNNNNN ĪIII
```

NNNNNN is the alphameric name.
ĪIII is the DIM entry number. Available FORTRAN library entries will appear as RRRRRR 9̄999 in place of NNNNNN ĪIII.

Load Programs Routine

The Load Programs routine is used initially to load SPS II-D or FORTRAN II-D object programs from the working cylinders or programs previously dumped by the Disk-to-Output routine, from cards or paper tape into disk storage. Overlap of occupied areas of disk storage is prevented by the routine. Programs cannot be loaded in the work cylinders with this routine. Programs will be loaded into areas of disk storage selected by the routine itself, if the user does not specify a storage area preference. If the routine selects the storage area, it will always store the program on a single cylinder, without overlapping cylinders, unless it is longer than an entire cylinder. If the user selects the storage area, it will be stored in the selected area regardless of cylinder overlap conditions.

This routine provides the following program loading options.

1. A name may be assigned to the program and placed in the Equivalence table.
2. A DIM entry may be assigned to the program.
3. The disk storage location can be specified and permanently assigned (fixed).
4. An entry address (execution address) can be assigned in the DIM entry to the program.
5. Read-only flags can be written in the sector addresses.
6. The disk storage location for the program can be specified by cylinder(s) without causing permanent assignment. Thus, several associated programs can be assigned to the same cylinder or group of cylinders by the user without actually specifying sector addresses.

7. Programs in either core-image or system output formats can be loaded; and programs in system output format can be converted to core image while loading.

It is possible, by exercising option 3, to permanently assign the sectors where the program is to be loaded in disk storage. This capability is provided in this routine only. When using this option, any programs already in the specified load area, but not permanently assigned, will be moved. The overlapped program is moved to the area which immediately follows the new program. If this in turn would result in additional overlapping of other programs, the process of moving programs continues until available space is found. If any program, in this move, is a permanently assigned program, or contains read-only flags in its sector addresses, no programs are moved and the new program will not be loaded.

A program is considered immovable if it is either permanently assigned by the Load Programs routine or if it contains read-only flags in any of its sector addresses. Permanently assigned programs can be

1. Deleted by the Delete Programs routine.
2. Copied by the Disk-to-Disk routine.
3. Changed by the Alter Sector routine.
4. Dumped into card or paper tape or printed on the typewriter.
5. Read for any purpose with normal read commands.

However, a permanently assigned program will not be moved in disk storage by the Disk Utility Program.

Programs being loaded can be "file protected" by writing read-only flags over the disk addresses of the storage sectors. All loading options are indicated by the control card.

Control Card (DLOAD).

Columns	
1	Asterisk (*)
2-6	Code word, (DLOAD).
7-12	Alphabetic name (left-justified) of program to be loaded into disk storage.
17-20	A DIM entry number to be given the program to be loaded. (This number will not be used by the routine if it is already assigned to another program.)
21-26	Beginning disk sector address in the work cylinders that contains the program to be permanently loaded. The first digit of the sector address selected must be 1, 3, 5, or 7.
27-32	Ending disk address in the work cylinders that contains the program to be permanently loaded. The first digit of the sector address selected must be 1, 3, 5, or 7.
33-38	Assigned disk storage address of the program to be loaded. (If this address is included, the program will be permanently assigned to the given address.) The first digit of the sector address selected must be 1, 3, 5, or 7.
39-43	Core storage address for a program that is placed in disk storage in core-image format. This address will be placed in the $\bar{C}$CCCC portion of the associated DIM entry.
44-48	Entry address (address of the first instruction to be executed) for a program that is being loaded. This address is placed in the $\bar{E}$EEEE portion of the associated DIM entry. This address is used for reading programs from disk in core-image format with the I/O routine.

49	Input device,
	C = card
	P = paper tape.
	D = disk storage work cylinders.

50	Letter I, if program to be loaded is in Core Image format. S, if program to be loaded is in System Output format. M, if program to be loaded is in System Output format, and it is to be converted to Core Image format before loading to disk storage.
51	Letter P, if read-only flags are to be written over disk addresses of storage sectors;

	otherwise leave blank.
52-54	Beginning cylinder (three digits XYY, where X is the module, 0, 1, 2, or 3, and YY is the cylinder number 00-99) to define lower limit where program can be loaded.
55-57	Ending cylinder (three digits XYY, where X is the module 0, 1, 2, or 3, and YY is the cylinder number 00-99) to define upper limit where program can be loaded. (Note that both the upper and lower limits will be ignored by the routine if columns 33-38 of this card are punched.)
60	Any non-blank Character, if program to be loaded is a FORTRAN or SPS object program which requires subroutines.

Replace Programs Routine

The Replace Programs routine is used to replace programs in disk storage with updated, changed, or new programs. Programs can be loaded to a disk storage area from cards, paper tape, or from another assigned disk storage area. In addition to loading disk stored programs, identified by DIM entries, programs can be loaded from work cylinders.

A program can be given another name in the Equivalence table by reloading the program over its original assigned disk area using a different name. The program can then be called by either name since both names are maintained in the Equivalence table.

With this routine, it is possible to load a program to itself adding read-only flags to the disk addresses. A permanently assigned program in disk storage cannot be replaced by this routine. To replace a permanent program, (1) delete the program with the Delete Programs routine and, (2) load the replacement program with the Load Programs routine.

The format of the control card for this routine follows. All fields are optional with the exception of columns 1-6, 17-20, and 49.

Control Card (DREPL).

Columns	1	Asterisk (*).
	2- 6	Code word, DREPL.
	7-12	Alphabetic name of program.
	13-16	The DIM entry number which identifies the program to be loaded if the program is from another assigned disk storage area. (The program to be loaded will be deleted from its original disk storage location.)
	17-20	The DIM entry number which identifies the program to be replaced.
	21-26	Beginning disk sector address if the program to be loaded is in the work cylinders. The first digit of the sector address selected must be 1, 3, 5, or 7.
	27-32	Ending disk sector address if the program to be loaded is in the work cylinders. The first digit of the sector address selected must be 1, 3, 5, or 7.
	39-43	Core storage address for a program that is to be placed in disk storage in core-image format. This address will be placed in the $\bar{C}$CCCC portion of the associated DIM entry.
	44-48	Entry address (address of the first instruction to be executed) for a program that is being loaded. This address is placed in the $\bar{E}$EEEE portion of the associated DIM entry.
	49	Input device, C = card. P = paper tape. D = disk storage.
	50	Letter I, if program to be loaded is in Core Image format. S, if the program to be loaded is in System Output format. M, if program to be loaded is in System Output format, and it is to be converted to

core image format before loading to disk storage.

51 Letter P, if read-only flags are to be written on disk addresses of storage sectors; otherwise leave blank.

60 Any non-blank character, if program to be loaded is a FORTRAN or SPS object program which requires subroutines.

Disk-to Disk Routine

This routine can be used to copy data or programs in disk storage to any available (unoccupied) disk storage area including the work cylinders. A program to be copied should be specified by a DIM entry, an alphabetic name that is in the Equivalence table, or a sector address given by the user. The program cannot be copied into an area which is already identified by a DIM entry number, except the work area (DIM entry 0001). Read-only flags may be written with the disk sector addresses of the copy, except in work cylinders, at the option of the user. When this routine is used, the DIM table and the original program remain unchanged. It is not possible to copy a program over a portion of that same program. It is not possible to copy a program into the work cylinders if that program exceeds the work cylinder limits. Data can be copied from one portion of the work area to another; however, no check will be made for overlapping of data within the work area. If a program or data to be copied is less than 100 sectors, there is no danger of overlap.

If any read-only flags are encountered in sector addresses within the copy area, an I/O routine error message is indicated. The program will be copied up to the point of the error.

The options offered by this routine are identified in the control card that follows.

Control Card (DCOPY).

Columns	1	Asterisk (*)
	2-6	Code word, DCOPY.
	7-12	Alphabetic name of program to be copied.
	13-16	The DIM entry number which identifies the program to be copied.
	21-26	Beginning sector address of program or data to be copied.
	27-32	Ending sector address of program or data to be copied. (Note that the beginning and ending sectors will always be used if present.)
	33-38	Beginning disk sector address of the new copy. This address must be that of work cylinders or available disk storage. This field must always be punched.
	51	Letter P, if read-only flags are to be written on disk sector addresses at the new location of the program; otherwise leave blank.

The sectors that are to contain the copy must not have read-only flags in the sector address initially or an error will be indicated and copying will be terminated.

After the data is successfully copied, the message

N̄NNNN SECTORS OF DATA
COPIED FROM XX̄XXXX TO YȲYYYY

is typed, where N̄NNNN specifies the number of copied sectors and XX̄XXXX and YȲYYYY are the beginning sector addresses of the From and To areas, respectively. If the copied data is written with read-only flags, an additional message is typed.

AND FILE PROTECTED

To move a program or data from one disk area to another, it should be: (1) copied to the work cylinders from the original area, (2) deleted from the original area, and (3) loaded to the new area from the work cylinders. This can be accomplished by using the Disk-to-Disk, Delete Programs, and Load Programs routines, in that order. A *DCOPY control record is used to copy the program into the work cylinders; a *DELET control record is used to make the original storage area available by deleting its DIM entry and Equivalence table entry; and a *DLOAD control record is used to load the program to a specified sector address and to generate the new DIM entry and Equivalence table entry.

Delete Programs Routine

This routine can be used to delete a program and its associated DIM entry, Sequential Program table entry, and Equivalence table entry (if any) or entries (where the program has more than one name) from disk storage. When a program is deleted, read-only flags will be removed and programs in successively higher disk storage positions will not be moved up to fill the vacated storage area.

The format of the control card follows.

Control Card (DELET).

Columns	1	Asterisk(*)
	2-6	Code word, DELET.
	7-12	Alphabetic name of program to be deleted (same name that appears in Equivalence table).
	13-16	DIM entry number of program to be deleted. (Note that either a Name or DIM entry number must be present, but not both.)

Define Parameters Routine

This routine can be used to alter the assignment of work cylinders, DIM table, Equivalence table, Sequential Program table, or certain system specifications in the System Communications Area for the Monitor I System. The DIM table may be lengthened or shortened, but may not be moved from cylinder 24. The Sequential Program table may be shortened, but must remain on cylinder 99. The Equivalence table may be lengthened or shortened. It will always immediately follow the DIM table, even if the DIM table is altered in length. This routine can also be used to indicate that more than one disk storage drive is to be used with the 1620.

When the size of the DIM table is changed, the Equivalence table will be moved to immediately follow the DIM table. When redefinition of an area (work cylinders, DIM table, or Sequential Program table) is attempted, the area must be available; i.e., it must not be occupied by programs with assigned DIM entries. If an area is unavailable, it will not be redefined and the message

DUP * ERROR 08

will be typed.

The normal assignment of disk storage for the above mentioned tables is as follows.

Description	Cylinder Assignment
Work Cylinders	00-23
DIM table	24
Equivalence table	25 (first eighty sectors)
Sequential Program table	99 (second through eighty-first sector)

To make any of the allowable alterations to these assignments, or to the system specifications, the user must enter a control card containing the new parameters. Only the parameters to be changed need to be punched in the control card. The parameters from a control card are processed from left to right by the routine. If any parameter is invalid, those parameters to its right will not be processed.

Control Card (DFINE).

Columns	
1	Asterisk(*).
2-6	Code word, DFINE.
7-12	Beginning disk sector address of work cylinders (must be first address of a cylinder).
14-16	Number of cylinders to be reserved for work cylinders (11 minimum, 99 maximum).
18	Number of disk storage drives on system (1-4). This must be defined before the user attempts to reassign the work cylinders to any disk drive other than the first.
20-22	Number of sectors to be reserved for DIM table (35 minimum, 999 maximum).
24-26	Number of sectors to be reserved for Equivalence table (9 minimum, 999 maximum).
28-30	Number of sectors reserved for Sequential Program table (80 sectors maximum). (Note that the same number of sectors is reserved for each disk storage drive on the system as defined in column 18.)
37-38	Standard length of mantissa for SPS floating-point subroutines (disk sector positions 40-41 of the Communications Area; 08 when the system is delivered). This value may be any number between 02-45.
40-41	Standard SPS subroutine set identification number (disk sector positions 42-43 of the Communications Area; 02 when the system is delivered). 00 – Divide Subroutine only. 01 – Fixed length mantissa (08) floating-point subroutines for machines equipped with Automatic Divide feature. 02 – Variable-length mantissa floating-point subroutines for machines equipped with the Automatic Divide feature. 03 – Variable-length mantissa floating-point subroutines for machines equipped with the Automatic Floating-point feature.
43	Standard N (noise) digit (any number 0–9) for SPS subroutines (disk sector position 44 of Communications Area; 0 when the system is delivered).
45-46	Standard length of mantissa (any number 02-28) for FORTRAN programs (disk sector positions 45-46 of Communications Area; 08 when the system is delivered).
48-49	Standard fixed-point word length (any number 04-10) for

	FORTRAN programming system (disk sector positions 47-48 of Communication Area, 04 when the system is delivered).
51	Source of Input, other than disk input, for FORTRAN subprograms (disk sector position 73 of Communications Area; 5 when the system is delivered). 3=paper tape 5=card
53	Core storage capacity of object machine (disk sector position 76 of Communications Area; 1 when the system is delivered). 1=20,000 3=40,000 5=60,000
57	FORTRAN Arithmetic and I/O subroutine set identification number (disk sector position 83 of Communications Area; 1 when the system is delivered). 1 = disk storage version 2 = core storage version 3 = disk storage version for machines equipped with the Automatic Floating Point feature. 4 = core storage version for machines equipped with the Automatic Floating-point feature.

The number of disk storage drives on the system may be 1, 2, 3 or 4. The Supervisor Program and the Disk Utility Program will need to know this number in order to utilize all available disk storage. The system will utilize only the first disk storage drive unless additional drive availability is specified by a DFINE control card. Therefore, it may be necessary for the user to process a DFINE control card immediately after initially loading the Monitor I System. When loading programs and assigning addresses, the Monitor System will start with the first available sector on the first available disk drive and proceed sequentially higher to available drives. Also, the user may want to change some of the other parameters of the system before any actual processing is initiated. If any errors are found in any data on a DFINE control card, all data to the left of the data in error will have been processed and data to the right will be ignored. See ERROR DETECTION AND CORRECTION for a description of possible DFINE errors.

When the routine is used to enlarge or shorten the tables or to change the number of disk storage drives for the system, the *DFINE record should be followed by a ‡ ‡PAUS record. After the routine is executed, ‡ ‡PAUS record will halt the computer to allow the operator to reinitialize the System; i.e., to reload the Monitor System into core storage. The procedure for calling the System into core storage from disk storage is described under OPERATION in the Monitor I System Section.

Define Disk Pack Label

This routine can be used to initialize a new disk pack for the Monitor System by writing the disk pack identification number in the label sectors (first and last sectors of cylinder 99) and the Sequential Program table in cylinder 99. All disk packs used by the Monitor System must be labeled and must contain a Sequential Program table. The disk pack identification number is written in the first five positions of the first sector in cylinder 99 and a read-only flag is written over the corresponding sector address. The same number is also written in the 31st through 35th positions of positions 1-100 of the last sector of the disk pack. This sector address is changed to 00199 regardless of the addressing scheme used for the remainder of the disk pack.

Note that it is necessary to initialize the disk pack which contains the Monitor system because the system pack is not automatically initialized when the system is loaded. Label sectors on a pack which contains the Monitor I System may be changed by this routine; however, the Sequential Program table will not be re-initialized. The Monitor System disk pack is identified by $\bar{0}4800$ in the sector address portion of DIM entry 3.

The format of the control card follows.

Control Card (DLABL)

Columns	1	Asterisk (*).
	2-6	Code word, DLABL.
	7-11	Disk pack indentification numder to be assigned.
	12	Disk drive number (0, 1, 2, or 3) of the disk drive that contains the disk pack to be labeled.

Both a disk pack identification number and disk drive number must be given. If either one is missing, the message

DUP * ERROR 01

is typed and the computer halts without writing label sectors. To correct the error, the operator may enter a corrected control card in the stacked input. Depressing the Start key will return control to the Monitor Control Record Analyzer routine to read the next Monitor Control record in the stacked input.

Only numerical characters may be entered for the disk pack identification number. If this number is all zeros or any position contains a letter or special character, the message

DUP * ERROR 10

is typed and the computer halts without writing a label sector. The restart procedure is the same as that given above for ERROR 01.

Define FORTRAN Library Subroutine Name

This routine permits the user to assign additional names (synonyms) for the FORTRAN library subroutines or to assign names to user-written library subroutines. Any user-written library subroutine with more than one entry will require this routine in order to place the name of the additional entries in the Equivalence table. These names are added to the Equivalence table within the first 50 entries of the system disk pack.

The control card format follows.

Control Card (DFLIB).

Columns	1	Asterisk (*).
	2-6	Code word, DFLIB.
	7-12	Name of library subroutine, left-justified.
	14-15	DIM number. This number was originally specified by the user when the subroutine was added to the system disk pack. The abbreviated 2-digit DIM numbers for the sixteen standard library subroutines may be found in the FORTRAN II-D Library Subroutines table in the FORTRAN section.

When a name is entered in the Equivalence table, the message

FORTRAN LIB NAME ENTERED NNNNNNĪIII

is typed, where NNNNNN is the name specified in columns 7-12 and ĪIII is the DIM number specified in columns 14-15 (preceded by two zeros).

Both a Name and DIM entry number must be given. If either is omitted, error message 01 will be typed.

Error message 10 will be typed for any of the following conditions.

1. The Name is all numbers, its first character is not alphabetical, or it contains special characters, including nonterminating blanks.
2. Columns 7-15 contain a record mark or group mark in any position or column 13 is not blank.
3. The DIM number is outside the range 10-39 or it contains letters or special characters.

Error message 54 will be typed if no space is available for the Name within the first 50 entries of the Equivalence table. Also, the Name itself will be typed.

Error message 51 will be typed if the name is a duplicate of another name in the Equivalence table. If operator action is required for any of the above messages, refer to ERROR DETECTION AND CORRECTION.

Error Detection and Correction

In addition to the messages described with the individual Disk Utility routines, other numbered error messages may be typed. These messages, described here, may be common to more than one Disk Utility Program as well as FORTRAN or SPS output operations that follow compilation or assembly. Table 1 indicates, by message number, the error messages that may be generated by each routine. A list of the error messages and their cause, and a list of operator actions for the associated messages follow.

Error Messages	Cause
DUP * ERROR 01	Field missing from control card.
DUP * ERROR 02	"TO" DIM entry number specified in DREPL control card is not in use in DIM table.
DUP * ERROR 03	"TO" DIM entry number, specified in a DREPL control card refers to permanently assigned program.
DUP * ERROR 04	"FROM" DIM entry number specified in a DDUMP, DREPL, or DCOPY control card is not in use in the DIM table
DUP * ERROR 05	Work cylinders illegally specified for program stor-

	age by DLOAD or DREPL control card entry.
DUP * ERROR 06	DIM entry number specified in a DDUMP, DLOAD, DREPL, DCOPY, or DELET control card is out of range of DIM table entry capacity.
DUP * ERROR 07	"FROM" DIM entry number in a DREPL control card refers to an immovable program.
DUP * ERROR 08	Insufficient available storage space at location specified by a DLOAD, DREPL, DCOPY, DFINE control card.
DUP * ERROR 09	DIM table is full.
DUP * ERROR 10	Field in DFLIB, DCOPY or DLABL control card contains invalid data.
DUP * ERROR 11	Number of modules specified in DFINE control card is greater than 4, or less than 1.
DUP * ERROR 12	Beginning disk sector address of work cylinder, in DFINE control card, is not first address in cylinder.
DUP * ERROR 13	Insufficient available storage for specified work cylinder (DFINE control card).
DUP * ERROR 14	Number of sectors specified by DFINE control card for Sequential Program table exceeds 80 sectors.
DUP * ERROR 15	Sector address is non-numerical in a DWRAD, DDUMP, DLOAD, DREPL, DCOPY, or DELET control record.
DUP * ERROR 16	Storage location specified by a DCOPY control card would cause program storage to overlap work cylinders if allowed.
DUP * ERROR 17	Starting sector address is greater than ending address for DWRAD, DLOAD, DREPL, or DCOPY control card.
DUP * ERROR 18	Sequential Program table is defined as less than required by the present contents of that table (DFINE control card).
DUP * ERROR 19	Core storage address of a program to be placed in

Table 1. Numbered Error Messages Generated by Disk Utility Routines

ROUTINE	ERROR MESSAGE NUMBER																																
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	24	51	52	53	54	55	56	57	58	59	60	61
Write Addresses (DWRAD)	X														X		X																
Alter Sector (DALTR)																																	
Disk-to-Output (DDUMP)	X			X		X									X					X													
Load Programs (DLOAD)	X				X	X		X	X						X		X		X			X	X	X	X	X	X						
Replace Programs (DREPL)	X	X	X	X	X	X	X	X							X		X		X				X		X	X	X						
Disk-to-Disk (DCOPY)	X			X		X		X		X					X	X	X			X													
Delete Programs (DELET)	X					X									X					X	X												
Define Parameters (DFINE)	X							X			X	X	X	X				X															
Define Disk Pack Label (DLABL)	X									X																							
Define FORTRAN Library Subroutine Name (DFLIB)	X									X													X			X							
FORTRAN or SPS Output																							X		X	X		X	X	X	X	X	X

	disk storage in core image format is less than 02302. A blank address will be treated as 02402. If the program is a Library function (DIM entries between 10 and 130), the message will not be indicated.
DUP * ERROR 20	Name specified by DDUMP, DCOPY, or DELET control card is not used in Equivalence table.
DUP * ERROR 21	DIM number specified by DELET control card is not in use.
DUP * ERROR 24	Cylinder limits specified in *DLOAD control card are greater than allowed by system parameters.
DUP * ERROR 51	Name specified is a DLOAD, DREPL, or DFLIB control card or a FORTRAN or SPS control card has been rejected because a duplicate name exists in the Equivalence table.
DUP * ERROR 52	"TO" DIM entry number specified in DLOAD control card is in use in DIM table (The routine will load the program and assign the DIM entry.)
DUP * ERROR 53 AAAAAA	Name specified in a DLOAD or DREPL control card or a FORTRAN or SPS control card has been rejected because the Equivalence table (with the exception of the first 50 entries) is full. AAAAAA is the rejected name.
DUP * ERROR 54 AAAAAA	Name specified in a DLOAD, DREPL, or a DFLIB control card or a FORTRAN or SPS control card has been rejected because the first 50 entries of the Equivalence table are full. AAAAAA is the rejected name.
DUP * ERROR 55 CARD SEQUENCE NNNNN	Sequence error has been found while reading a program to be loaded to disk storage. NNNNN is the sequence number of the card that is out of sequence. Only cards with an eleven punch over the leftmost position of the sequence number (column 76) are sequence-checked; therefore, patch cards are excluded from the check.
DUP * ERROR 56	DIM number supplied in FORTRAN or SPS control card is in use in DIM table, and Name specified in the same card has a different DIM number in the Equivalence table.
DUP * ERROR 57	DIM number supplied in FORTRAN or SPS control card is in use in DIM table, and Name specified in the same card has no matching name in the Equivalence table.
DUP * ERROR 58	"TO" DIM entry number specified in FORTRAN or SPS control card refers to a permanently-assigned program storage area.
DUP * ERROR 59	DIM entry number specified in a FORTRAN or SPS control card is out of range of DIM table entry capacity.
DUP * ERROR 60	Insufficient available storage space for a function specified by a FORTRAN or SPS control card.
DUP * ERROR 61	DIM table full.

Operator Action

Messages 1-24. After the message is typed, the computer halts. The operator may then enter a corrected control record. Depressing the Start key returns control to the Monitor Control Record Analyzer routine to read the next Monitor Control record.

Message 51. No operator action required. The routine continues and loads the program without placing the name in the Equivalence table.

Message 52. The routine continues; it assigns a DIM entry and loads the program.

Messages 53, 54. No operator action required. The routine continues and loads the program without placing a name in the Equivalence table.

Message 55. After the message is typed the computer halts. To restart:

1. Remove the cards from the hopper.
2. Depress the Nonprocess Runout key.
3. Remove the last two cards from the stacker.

4. Arrange cards from steps 1 and 3 in correct sequence and place them in the hopper.
5. Depress the Reader Start key. Note that paper tape contains no sequence number, therefore, it can never generate this type of error.

Message 56. No operator action required. The routine continues and loads the program, generating a DIM entry, without placing the name in the Equivalence table.

Message 57. No operator action required. The routine continues and loads the program, generating a DIM entry, and places the name in the Equivalence table.

Messages 58, 59. No operator action required. The routine continues generating a DIM entry and loading the program to disk storage.

Messages 60, 61. After the message types, the computer halts without loading the programs, providing no other output is requested. (If a FORTRAN or SPS control record is included with the source data to indicate that the compiled or assembled object program is to be punched, the program is outputted without halting the computer.) To correct the error, a DLOAD, DREPL, or DDUMP Monitor Control record can be entered in the stacked input to load the program to a different location from the work cylinders or to output the program. Depressing the Start key returns control to the Monitor Control Record Analyzer routine to read the next Monitor Control record.

FORTRAN and SPS Output

The Disk Utility Program contains the output routines for both FORTRAN and SPS. These routines load object programs to disk storage and punch them out into cards or paper tape.

Following compilation or assembly, the message

DK LOADED AAAAAA IIII D$\bar{\text{D}}$DDDD
$\bar{\text{S}}$SS $\bar{\text{C}}$CCCC $\bar{\text{E}}$EEEE ‡

is always typed to inform the user about the assigned DIM entry. AAAAAA is the supplied name, IIII is the DIM entry number, and the remainder of the message is the DIM entry.

For programs being loaded into disk storage, the user may select the DIM entry and/or name. Names and DIM numbers are supplied in FORTRAN (*LDISK) or SPS (*NAME, *ID NUMBER) control records.

Processing of the Equivalence table and DIM table and the actual loading is dependent upon whether the user supplies a Name and DIM number, Name only, or DIM number only.

Name and number supplied by user.

1. If DIM entry is in use in DIM table.
   a. and Name is already in Equivalence table.
      1) If Name in table references number supplied, replace old program with new program.
      2) If Name in table references another number, type error message 56, and load program with available DIM entry number and no Name.
   b. and Name is not in Equivalence table, type error message 57 and load new program with available DIM entry number and add Name in Equivalence table.
2. If DIM entry is not in use in DIM table.
   a. and Name is already in Equivalence table, type error message 51 and load program with assigned DIM entry number without assigning Name.
   b. and Name is not in Equivalence table, load program and place Name in Equivalence table.

Name only supplied by user.

1. If an identical Name is in the Equivalence table, the program is loaded *without* the supplied name (error 51 will be indicated).
2. If an identical Name is *not* in the Equivalence table, the object program is loaded, an available DIM entry is assigned, and the name is added to the Equivalence table.

DIM entry number only supplied by user.

1. Program is loaded. If number was in use, object program replaces old programs. Names referencing the old program are deleted from the Equivalence table.

SPS II-D

SPS II-D is a disk-oriented assembly program designed to simplify the preparation of programs for the IBM 1620 Data Processing System and the IBM 1710 Control System. The development of larger and more versatile data processing systems like the 1620 and 1710 has resulted in a greater number of, and more complex, machine language instructions. The difficulties of coding in machine language — a tedious and time-consuming task — have been recognized and one of the efforts toward simplification is the system known as Symbolic Programming.

A Symbolic Programming System (SPS) permits the programmer to code in a symbolic language that is more meaningful and easy to handle than numerical machine language. SPS II-D automatically assigns and keeps a record of storage locations and checks for coding errors. By relieving the programmer of these burdensome tasks, SPS II-D significantly reduces the amount of programming time and effort required.

This section is intended to serve as a reference text for the SPS II-D Programming System. It assumes that the programmer is familiar with the methods of data handling and the functions of instructions used in the 1620 Data Processing System and the 1710 Control System. For those without this knowledge, information on 1620 and 1710 systems can be found in the appropriate reference manuals. Refer to IBM *1620 Bibliography* (Form A26-5692) and IBM *1710 Bibliography* (Form A26-5695).

Introduction

The SPS II-D Programming System may be divided into the symbolic language used in writing a program, the library containing the subroutines and the linkage instructions (macro-instructions) that may be incorporated into the program, and the processor program that is used to assemble the user's program.

Symbolic Language

Symbolic language is the notation used by the programmer to write (code) the program. The program written in SPS language is called a "source program." This language provides the programmer with mnemonic operation codes, special characters, and other necessary symbols. The use of symbolic names (labels) makes a program independent of actual machine locations. Programs and routines written in SPS language can be relocated and combined as desired. Routines within a program can be written independently with no loss of efficiency in the final program. Symbolic instructions may be added or deleted without reassigning storage addresses.

Macro-Instructions, Subroutines, and Subprograms

The macro-instructions that are written in a source program are commands to the processor to generate the necessary linkage instructions. Linkage instructions provide the path to a subroutine or subprogram and a return path to the user's program. These subroutines may be special subroutines prepared by the user or any of seventeen IBM Library subroutines, such as floating divide, square root, and arctangent. (Subprograms are user-written only.) The ability to process macro-instructions simplifies programming and reduces the time required to write a program.

SPS II-D Processor

After a source program is written, it is punched into cards, or into paper tape if the system is equipped with an IBM 1621 Paper Tape Reader. It is then "assembled" into a finished machine language program known as the "object program."

Assembly is accomplished by the SPS II-D processor program which is stored on the disk. The function of the processor program is to translate the symbolic language of the programming system into the language of the 1620 or 1710. The translation is one for one — the processor produces one machine language instruction for each machine instruction (except macro-instructions) written in symbolic form.

Symbolic Programming

Symbolic programming may be defined as a method wherein names, characteristics of instructions, or closely related symbols are used to write programs. The core of the symbolic language is the operation code. SPS II-D permits the programmer to write programs in a simple, familiar language. It does not require a detailed knowledge of the machine because, in coding the program, the programmer uses operation codes that are in easily remembered mnemonic form rather than in the numerical language of the machine. Operation codes are of three types: Declarative, Imperative, and Control.

Declarative Operation Codes

Declarative operation codes are used for the assignment of core storage for input areas, output areas, and working areas. The assigned areas are utilized by the object program and may contain the data to be processed and/or the constants (numerical or alphameric characters) required in the object program when the data is processed. Declarative statements never generate instructions in the object program, but may generate constants that are assembled as part of the object program.

Imperative Operation Codes

Imperative operation codes specify the operations or instructions that the object program is to perform. In this group are included all arithmetic, branching, and input/output statements. Most statements on the coding sheet prepared by the programmer are of this type. These statements are translated one for one (except macro-instructions) and are assembled as the machine language instructions of the object program.

Control Operation Codes

Control operation codes are commands to the processor that provide the programmer with control over portions of the assembly process. Instructions of this type do not normally generate instructions in the object program.

The actual and mnemonic operation codes within these categories are presented under PROGRAMMING THE 1620/1710 USING SPS II-D.

The statements or instructions in the source program must be entered by the programmer in logical sequence on the coding sheet.

Coding Sheet

The programmer enters all information relevant to the coding of the source program and subsequent assembly of the object program on a coding sheet, Form X26-5627 (Figure 6). Figure 7 shows a sample input card, Form J59692. The format of the input card or paper tape record follows the headings on the coding sheet. In paper tape, the first punching position of a record is said to be column 1. The card columns assigned to a single heading are referred to as a field. The following is an explanation of the headings in the order of their appearance on the sheet.

Heading Line

Space is provided at the top of each page for the name of the Program, Routine, Programmer, and for the Date. This information does not constitute part of the source program language and is not punched.

Page Number (Columns 1-2)

A 2-digit page number is entered to maintain the order of the program sheets. This normally numerical entry becomes the first two digits of each statement that is punched from the sheet.

Line Number (Columns 3-5)

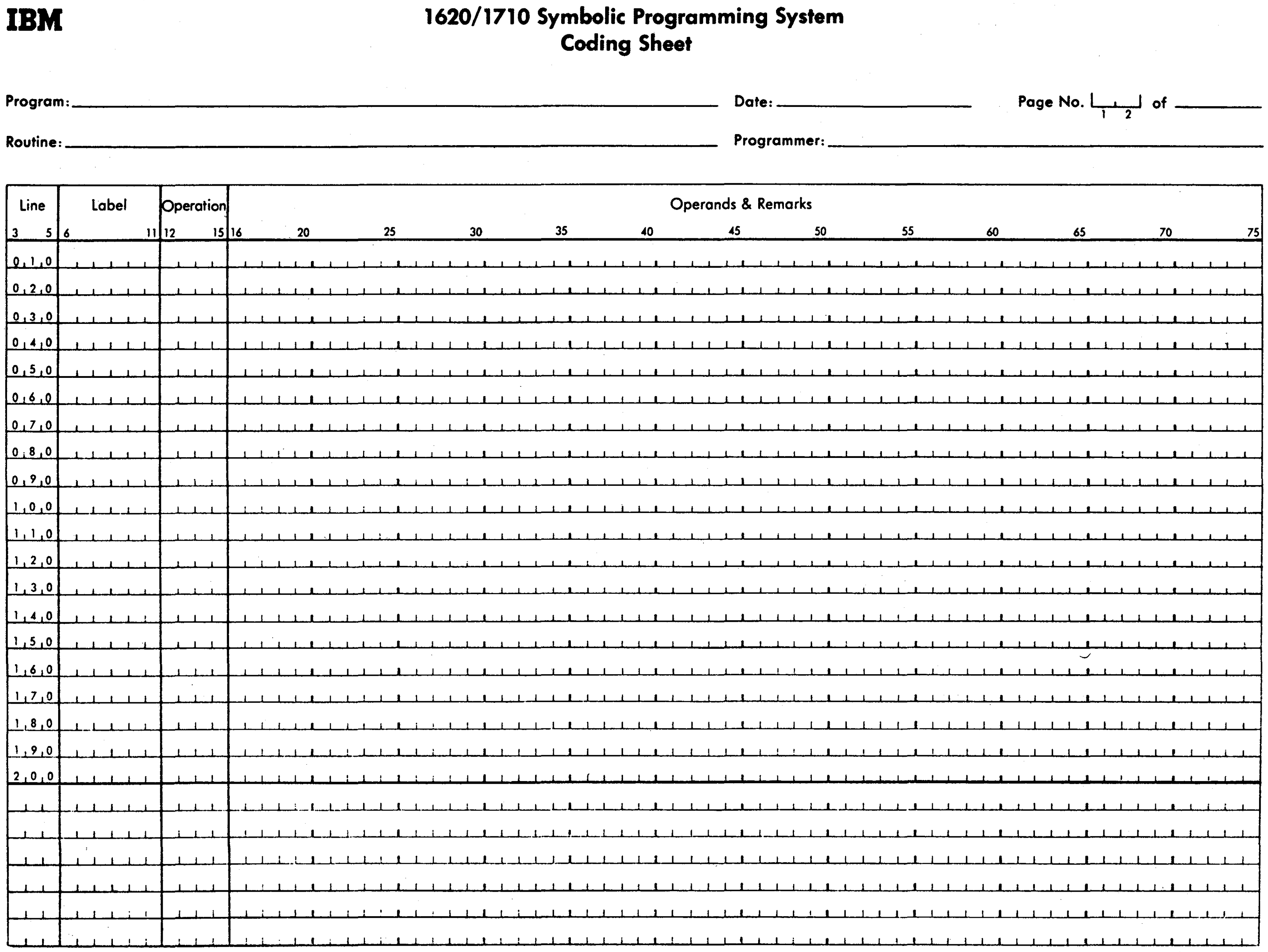
A 3-digit line number is entered on the sheet to maintain the sequence of the statements coded. The first 20 lines on each sheet are prenumbered 010, 020, 030, etc., through 200. At the bottom of the sheet, six unnumbered lines are provided for inserts or for continuing the line numbering. The inserted statement should be numbered so that it falls sequentially between the statements immediately preceding and following it. The arrangement of the prenumbered lines, 010, 020, etc., permits up to nine statements to be inserted between any two statements. After the cards for each of the lines are punched, they should be placed in correct numerical order.

Label (Columns 6-11)

The label field represents the machine location of either data or instructions. The field may be left blank or may be filled with a symbolic address. Only the data or instructions that are referred to elsewhere in the program need a label.

A label may consist of up to six alphameric characters beginning at the leftmost position in the label field. At least one of the characters must be alphabetic or one of four permissible special characters, namely, the equal sign (=), slash symbol (/), "at" sign (@), and period (.).

The best labels to select are those that are mnemonically descriptive of the area or instructions to which they are assigned. Labels that have an obvious meaning not only provide easily remembered references for the original programmer but also assist others who may assume responsibility for the program.



IBM

1620/1710 Symbolic Programming System
Coding Sheet

Program: ______________________ Date: __________ Page No. [1 2] of ______

Routine: ______________________ Programmer: ______________________

Line	Label	Operation	Operands & Remarks
3 5	6 11	12 15	16 20 25 30 35 40 45 50 55 60 65 70 75
010			
020			
030			
040			
050			
060			
070			
080			
090			
100			
110			
120			
130			
140			
150			
160			
170			
180			
190			
200			

Figure 6. 1620/1710 SPS Coding Sheet

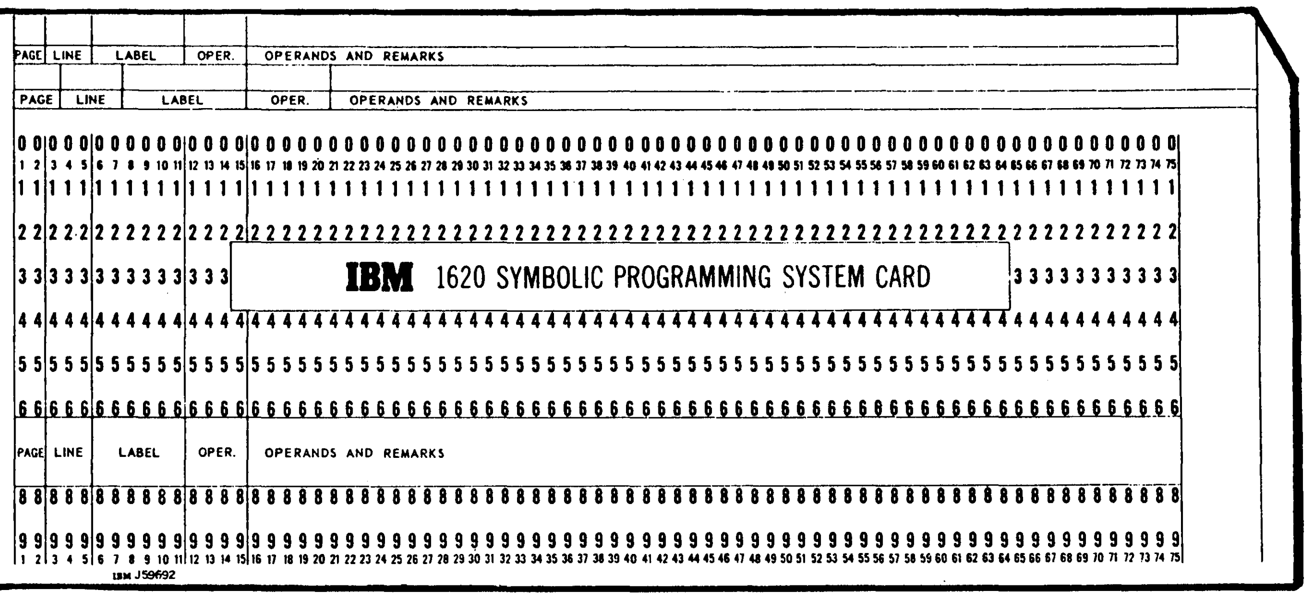

PAGE LINE LABEL OPER. OPERANDS AND REMARKS
IBM 1620 SYMBOLIC PROGRAMMING SYSTEM CARD
IBM J59692

Figure 7. SPS Source Program Card

Operation (Columns 12-15)

The 4-position operation field contains the actual 2-digit numerical operation code or the mnemonic representation of the operation code to be performed. In either case, the first character of the operation code must start in the leftmost position, column 12, of the operation field. Listings of permissible mnemonic codes and actual operation codes are shown under PROGRAMMING THE 1620/1710 USING SPS II-D.

Operands and Remarks (Columns 16-75)

The operands and remarks field is used to specify the information that is to be operated upon and may contain, if desired, any additional remarks concerning the statement.

For declarative operation statements, the first operand usually defines the length; the remaining operands, if present, specify constants, an address, and remarks.

For imperative operation statements, the operands and remarks field contains, at most, four items: three of these are operands and the fourth, remarks. The first two operands may be the symbolic or actual addresses of data or instructions, i.e., the P and Q portions of the instruction. The third operand, which should be numerical, is called the flag indicator operand and is used to set flags in the assembled instruction. The final item consists of the remarks associated with each statement. Imperative statements need not contain all four items. Any one or more than one may be omitted. The two special characters which may not be used in an operand are the right and the left parenthesis,) (.

A control operation statement normally contains only one operand.

Statement Writing

Certain rules must be observed in writing or coding the statements that make up the source program. This section contains rules that apply to the statements and their elements, rules governing the length and types of statements, use of special characters, the flag indicator operand and immediate (Q) operand, types of addresses used as operands, and address adjustment by arithmetic, a method that relieves the programmer of considerable effort and reduces the number of symbols required for a source program.

Statements

Symbolic statements are classed according to the operation code they contain, and thus are designated Declarative, Imperative, or Control statements. In addition to the page and line number a statement may contain a label, operation code, operands, and remarks. No statement in the source program may exceed 75 characters in length. Since page number, line number, label, and operation require 15 positions, the operands and remarks field may not exceed 60 characters. In the case of the paper tape SPS, the end-of-line character is considered to be part of the operands and remarks field.

Use of Special Characters in Statement Writing

The comma, asterisk, end-of-line character, blank, "at" (@) sign, and dollar sign are special characters that possess distinct meanings in the writing of source programs. Their use, as well as that of the special characters used as operators for address adjustment, are explained in detail in this section.

Comma

The comma is normally used to separate items in a statement. The term item refers here to parts of the operands and remarks field, such as the P and Q operands, the flag indicator operand, remarks, length, constants, etc. An imperative statement may consist of four items: the P and Q operands, the flag indicator operand, and remarks, but need not contain all four items. Any one or more than one may be omitted.

If one item is omitted and more items follow, the comma that normally follows the omitted item must be present. For example, if the flag indicator operand is omitted but remarks are present in the instruction, the format of the field will be:

Line 3–5	Label 6–11	Operation 12–15	Operands & Remarks 16–50
010		TF	DELTAX,X,,TRANSMIT VALUE

All imperative statements that contain remarks must include three commas in the operands field, even when the operands are omitted. During assembly, the omitted P or Q operands will be replaced by zeros in the P or Q portion of the assembled instruction.

Commas indicating omission need not be present in statements in which the last item(s) is omitted. For example, in the statement in which both the flag indicator operand and remarks are omitted, no commas need be used following the second operand.

Line 3–5	Label 6–11	Operation 12–15	Operands & Remarks 16–50
010		TF	DELTAX,X

EXAMPLES

Line 3–5	Label 6–11	Operation 12–15	Operands & Remarks 16–50
010		H	,,,HALT INSTRUCTION
020		A	16352,17865,,ADD FACTOR B TO A

Statement 010 is a halt operation that requires three commas (,,,) in front of the remarks; these take the place of the P, Q, and flag indicator operands. In statement 020, the first two commas set off the P and Q operands, whereas the third comma takes the place of the omitted flag indicator operand. The number of commas required for declarative statements may be one or two as explained under DECLARATIVE OPERATIONS.

Asterisk

The asterisk has three uses: in writing comments, as an operand or term of an operand, and in address adjustment.

Lines of descriptive information may be inserted in the program by placing an asterisk (*) in column 6 of the label field. Comments then may be written in columns 7 through 75. Comments inserted in this way will appear in the symbolic output, but will not affect in any way the operation of the program. A comment statement does not produce an entry in the object program.

Line 3–5	Label 6–11	Operation 12–15	Operands & Remarks 16–50
010	*REMAR	KS C	AN BE WRITTEN IN SUCH A MANNER SO
020	* AS T	O DE	SCRIBE THE PROGRAM
030	* MAIN	PRO	GRAM FOLLOWS

Statements 010, 020, and 030 are remarks that do not generate instructions.

The asterisk can be used as the first character or *term* of an operand in an imperative statement and is interpreted by the program as the address of the high-order (leftmost) position of the address of the instruction. It may also be used as any *term* of the operand to indicate the high-order (leftmost) position of the address of the instruction.

When the asterisk is used in address adjustment as an *operator*, it indicates to the processor that a multiplication must be performed in order to adjust the address.

End-of-Line Character

An end-of-line character Ⓔ is required only on source statements that are to be processed on paper tape. Use of this character allows statements to be located on the tape immediately adjacent to each other, with no intervening blank characters. The statements are in "free" form; that is, they are not assigned a fixed number of positions.

Source statements that are to be processed in punched card form do not require an end-of-line character; the remainder of the line is left blank and this is recognized by the processor as the end of the statement.

When the end-of-line character is punched in a card for off-line conversion to paper tape, it is represented by a 12-5-8 punch combination.

Blank Character

A blank character in operands of the source statement is ignored by the processor except in DAC statements (alphameric constants), in which blanks are considered valid characters. In effect, the statement is condensed before it is processed.

Because blanks are ignored by the processor, the programmer, to achieve clarity on his coding sheets and output listing, may write his statements in modified "fixed" form (see Figure 8).

In this example, columns 16, 36, and 57 are arbitrary choices for the locations of the operands. The comma following or replacing the P operand may be in any column from 16 through 35.

Blanks are permitted in any position within a flag indicator operand except between the 1 and 0 in number 10, and between the 1 and 1 in number 11. A blank or blanks in the address operand of a *declarative* statement, when set off by commas, is interpreted by the processor as a zero address.

"At" Sign

When the "at" sign (@) is used as part of a constant being defined by a DC, DSC, or DAC statement, a record mark (⧧) is created by the processor and inserted into the constant in place of the @. Specific rules for use of the @ are covered under DECLARATIVE OPERATIONS.

Dollar Sign

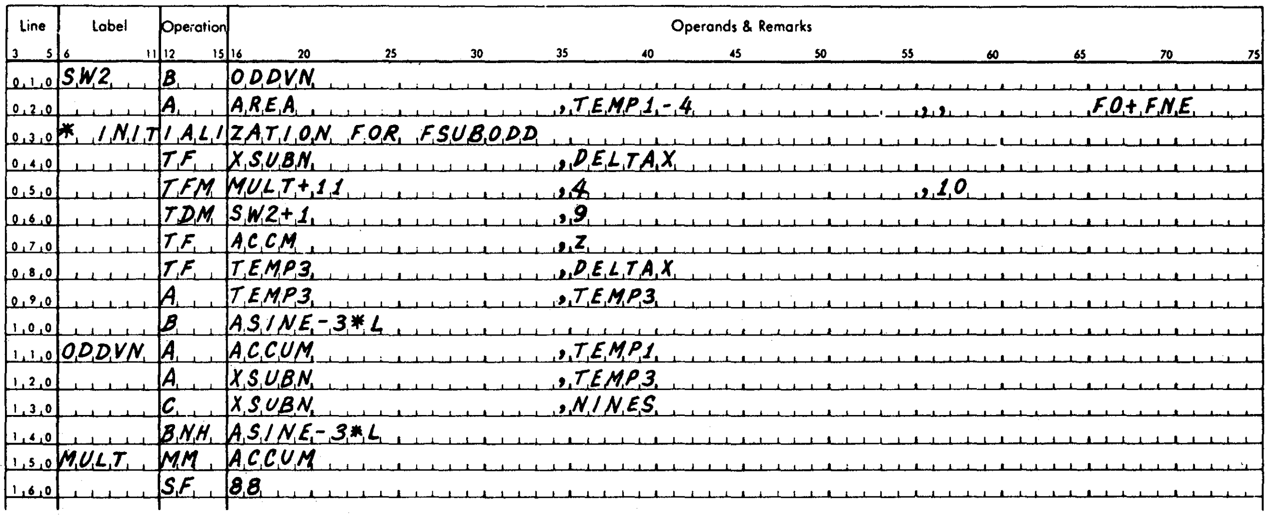
The dollar sign ($) is used in an operand to instruct the processor that the symbolic address in an operand has a specific heading character. The $ is written between the heading character and the symbol. For example, in an operand the heading character "5" and the symbol "SUM" appear as 5$SUM. For additional information on the use of the $, refer to HEAD-HEADING in the PROCESSOR CONTROL OPERATIONS section.

Operands

Flag Indicator Operand

The flag indicator operand specifies the positions that are to be flagged in the assembled instruction. These positions are numbered from left to right, 0 through 11, and must be listed sequentially. For example, if positions 2 7, and 10 are to be flagged, the flag indicator operand should be coded 2710, not 2107. All positions may be flagged, if desired. The operand then will be coded 01234567891011 and must be written in that order.

Normally, no flags are set when the flag indicator operand is omitted. However, if the flag indicator operand is omitted from immediate instructions (except TDM), a flag is automatically set in position



Line 3 5	Label 6 11	Operation 12 15	Operands & Remarks 16 … 35	36 … 56	57 … 75
010	SW2	B	ODDVN		
020		A	AREA	,TEMP1-4	,, F0+FNE
030	* INIT	IALI	ZATION FOR FSUBODD		
040		TF	XSUBN	,DELTAX	
050		TFM	MULT+11	,4	,10
060		TDM	SW2+1	,9	
070		TF	ACCM	,Z	
080		TF	TEMP3	,DELTAX	
090		A	TEMP3	,TEMP3	
100		B	ASINE-3*L		
110	ODDVN	A	ACCUM	,TEMP1	
120		A	XSUBN	,TEMP3	
130		C	XSUBN	,NINES	
140		BNH	ASINE-3*L		
150	MULT	MM	ACCUM		
160		SF	88		

Figure 8. SPS Statements in Modified "Fixed" Form

Q_7. If the operand is present, only the positions indicated are flagged.

The flag indicator operand can be used to insert a flag over the units position of the P and Q addresses, if the source program is written for a 1620 or 1710 that has Indirect Addressing (special feature).

Immediate (Q Operand)

With immediate-type instructions such as Add Immediate (AM), Subtract Immediate (SM), and with actual operation codes that begin with the digit 1, the Q operand represents the actual data to be used by the instruction. It may be absolute or symbolic as previously defined. High-order zeros of absolute data may be eliminated.

During assembly, the processor automatically places a flag over position Q_7 of an immediate instruction unless a flag indicator operand indicates otherwise. For example, the statement

Line (3-5)	Label (6-11)	Operation (12-15)	Operands & Remarks (16-50)
010		SM	TOTAL,10023

causes the number $\overline{1}0023$ to be subtracted from the field called TOTAL because the flag that terminates the field to be subtracted is automatically placed over position Q_7. However, the statement

Line (3-5)	Label (6-11)	Operation (12-15)	Operands & Remarks (16-50)
010		SM	TOTAL,10023,10

will cause only the number $\overline{2}3$ to be subtracted from the field called TOTAL because the flag indicator operand directs that the field-terminating flag be placed over position Q_{10} rather than Q_7. There is one exception to this rule: a transmit digit immediate instruction (TDM, code 15) does not require a flag; therefore, none is automatically set by the processor.

Types of Addresses Used as Operands

Operands assembled by the processor may be of three types: actual, symbolic, and asterisk. The individual applications for a particular type of address are described in the section PROGRAMMING THE 1620/1710 USING SPS II-D.

Actual Address

An actual address consists of five digits (00000-19999) for a standard capacity machine and is, as the name implies, the actual core storage address of a piece of data or an instruction. High-order zeros of an actual address may be eliminated. For example, the statement

Line (3-5)	Label (6-11)	Operation (12-15)	Operands & Remarks (16-50)
010		A	3684,12251

causes the data in storage location 12251 to be added to the data in storage location 03684.

Symbolic Address

A symbolic address is the name assigned by the programmer to the location of an instruction or a piece of data. A symbolic address is valid only if it is defined (given an actual numerical value) by a declarative statement somewhere in the source program, or if it is used as the label of an instruction. Symbolic addresses may contain from one to six characters (letters, digits, or special characters) with the following restrictions:

1. At least one character must be nonnumerical.
2. The only permissible special characters are: equal sign (=), slash symbol (/), "at" sign (@), and period (.).

Blanks are permitted within a symbol; however, they are ignored by the processor during assembly.

The example shown below contains both an actual address and a symbolic address.

Line (3-5)	Label (6-11)	Operation (12-15)	Operands & Remarks (16-50)
010		A	TOTAL,12251

In this example, the data in the field whose actual address is 12251 is added to a field whose address is the symbolic name TOTAL.

Asterisk Address

When the asterisk is used as the first character of an operand in an imperative operation, it is interpreted by the processor to be the address of the high-order (leftmost) position of the instruction. For example, the statement

Line	Label	Operation	Operands & Remarks
3 5	6 11	12 15	16 20 25 30 35 40 45 50
0,1,0		BNF	START,*

indicates to the processor that the Q portion of the instruction should contain the address of the instruction. This instruction is assembled as 44 01234 01876, where START equals 1234 and the address assigned to the instruction is 1876. Thus, when executed in the object program, this instruction examines its own leftmost position (1876) for a flag and either branches to the instruction at location 01234 or continues, on the basis of the examination, to the next instruction located at 01888.

When an asterisk (*) address is used with either declarative or control operations, it refers to the rightmost position of storage last assigned by the location assignment counter of the processor — not to the leftmost character of the instruction. For example, the statements

Line	Label	Operation	Operands & Remarks
3 5	6 11	12 15	16 20 25 30 35 40 45 50
0,1,0		TFM	12045,70000
0,2,0		DC	1,@,*

produce the instruction

16 12045 7000‡

Since record marks can be defined only in declarative operations, an imperative statement should be followed by a DC statement when a record mark is required in the instruction. The rightmost position of the instruction is the rightmost position of storage last assigned; therefore, it is also the position where the ‡ (constant) is stored.

Address Adjustment of Operands

Address adjustment is used to tell the processor to arithmetically adjust the addresses in operands. It is permitted with all types of addresses: actual, symbolic, and asterisk, and is used to refer to a location that is a given number of positions away from a specific address. Use of this feature of the language reduces the number of symbols necessary for a source program.

By writing a + (plus sign) for addition, – (minus sign) for subtraction, and * (asterisk) for multiplication, immediately after the first or subsequent term of an operand (an asterisk as a term of an operand does not represent multiplication but means the address of the instruction, as previously explained), the programmer indicates to the processor that the address is to be adjusted.

Arithmetically adjusted operands may take the form of $A \overset{*}{\pm} B \overset{*}{\pm} C \overset{*}{\pm} D$, where the terms A, B, C, and D may be numerical quantities. The number of terms in the operand is limited only by the size of the operand and remarks field. Thus the operand A + B * C – D may be further adjusted by writing after the last term another term, E, for example, A + B * C – D + E.

In arithmetically adjusted operands, the operation or operations of multiplication are always performed first, followed by the addition and subtraction required to calculate the adjusted address. Intermediate results that are greater than ten digits, or a final result (adjusted address) that is over five digits, cannot be calculated by the processor.

For the 1620 or 1710 with standard storage capacity (20,000 storage positions), addresses that exceed 19999 are considered errors; however, they will not be detected as such. Therefore it is possible, with a standard capacity machine, to assemble an object program for a machine with 40,000 or 60,000 positions of storage. For machines that have 40,000 or 60,000 positions of core storage, the processor is automatically modified to use the additional storage to enlarge the size of the symbol table. In that case, addresses that do not exceed 39999 or 59999, depending upon the storage capacity, are considered valid addresses.

In using address adjustment, the programmer should be careful that insertions or deletions do not affect the adjusted address. For example, if a P operand in a branch (B) instruction refers to an address as * +48 (i.e., branch to the instruction that follows the next three sequentially higher instructions), the programmer must ensure that no new instructions are introduced within the three instructions to make the * +48 incorrect. In this example, the asterisk (*) is the leftmost position of the instruction itself.

EXAMPLES

Line	Label	Operation		ADJUSTED ADDRESS	ARITHMETIC
3 5	6 11	12 15	16 20 25		
0,1,0			ALPHA+40	01040	= 1000+40
0,2,0			ALPHA-30	00970	= 1000-30
0,3,0			ALPHA+2*L	01008	= 1000+ (2X4)
0,4,0			ALPHA*3	03000	= 1000 X 3
0,5,0			ALPHA*L	04000	= 1000 X 4
0,6,0			500+20*3-11	00549	= 500+(20 X3) -11
0,7,0			100*5+20*3-11	00549	= (100 X5) + (20X3)-11
0,8,0			*+12	02012	= 2000 + 12
0,9,0			**3*2	12000	= (2000 X3) X2

The operands shown will produce the adjusted addresses, as indicated, provided the location 1000 has been assigned to the symbolic address ALPHA, the location 4 has been assigned the symbolic address L, and the instruction location (*) is equivalent to 2000.

Branch Operands

In some instructions such as Branch and Branch Back, the Q address is not used, although a zero (00000) address is generated. Thus the instruction uses twelve storage positions. By using an * address in the following statements,

Line	Label	Operation	Operands & Remarks
010		B	13668
020		DORG	*-3
030	NEXT	TFM	12045,70000

the instructions are condensed, to eliminate four positions of the unused (zero) Q address, and are stored as

49136680161204$\bar{5}$70000

whereas the statements

Line	Label	Operation	Operands & Remarks
010		B	13668
020	NEXT	TFM	12045,70000

are stored as

4913668000001612045$\bar{5}$70000

because the unused Q address is not eliminated. In the first example, only four positions of storage are saved; however, a considerable amount of storage can be saved in a program that contains many instructions where the Q or both the Q and P portions of instructions are unused. Because the * in the DORG statements (see PROCESSOR CONTROL OPERATIONS) refers to the rightmost position of storage last assigned (Q_{11} of the B instruction), * –3 is the address where the next instruction starts.

To automatically eliminate the unused storage assigned to Branch or Branch Back instructions, the following two imperative mnemonics are included in the SPS II-D language:

Mnemonic	Meaning
B7	Branch and adjust location assignment counter
BB2	Branch Back and adjust location assignment counter

These mnemonics are written left-justified in the operation field of the statements as shown in the following example.

Line	Label	Operation	Operands & Remarks
010		B7	ADDR
020		BB2	

The first statement is equivalent to the following symbolic instructions:

B	ADDR
DORG	* –4

where ADDR is the address used by the branch instruction (B). The second statement is equivalent to the following symbolic instructions:

BB	
DORG	* –9

For all imperative statements, except B7 and BB2, the address assignment counter is incremented by 12. For B7 and BB2 statements, the address assignment counter is incremented by 7 and 2, respectively. A label may be included with a B7 or BB2 statement.

Inserting Flags

By placing a minus sign in front of the first term of an operand, a flag (minus sign) can be inserted over the units position of the adjusted address. This feature of address adjustment can be used for inserting flags required for Indirect Addressing (special feature). However, an operand written as –0 (minus zero) does not insert the flag in the units position over the zero. When the minus sign is written in front of the first term of the operand in order to set a flag over the units position, other signs following the first term should be reversed so that the correct address is obtained.

Programming the 1620/1710 Using SPS II-D

This section describes in detail the various steps to be followed in writing a program for the 1620 or 1710 using SPS II-D. The material has been divided into three categories: Declarative Operations, Imperative Operations, and Processor Control Operations. The imperative operations that apply to the 1710 only are listed in Table 22 in the Appendix.

Declarative Operations

In programming the 1620 or 1710, all records, and any other data that is to be processed by the program, must be assigned storage areas. Normally, all records and data to be processed consist of fields of known length and arrangement. Unless otherwise specified, areas are automatically assigned core storage locations in the order in which they appear in the source statements.

The declarative statements provide the object program with the input/output areas, work areas, and constants it requires to accomplish its assigned task. These statements do not produce instructions that are executed in the object program. The entries, DS, DSS, DAS, and DSB assign storage. The entries, DC, DSC, DVLC, DAC, DSAC, DSA, DNB, DDA, DGM, and DMES usually assign storage, and also produce, in the object program, both the machine address of the area assigned and the constants that are to be stored in this area. Constants are then loaded with the object program.

Declarative statements may be entered at any point in the source program. However, these statements are normally placed by themselves, preferably at the beginning or end of the program – not within the instruction area. If not placed at the beginning or end, the programmer is required to branch around an area assigned to data so the program will not attempt to execute what is in a data area as an instruction.

The declarative mnemonic operation codes and their meanings follow:

Code	Meaning
DS	Define Symbol (Numerical)
DSS	Define Special Symbol (Numerical)
DAS	Define Alphameric Symbol
DC	Define Constant (Numerical)
DSC	Define Special Constant (Numerical)
DVLC	Define Variable Length Constant
DAC	Define Alphameric Constant
DSAC	Define Special Alphameric Constant
DSA	Define Symbolic Address
DSB	Define Symbolic Block
DNB	Define Numerical Blank
DDA	Define Disk Address
DGM	Define Group Mark
DMES	Define Message (1710 Only)

DS — Define Symbol (Numerical)

A DS statement may be used to define symbols used in the source program (i.e., to assign storage addresses or values to symbolic addresses or labels) and to assign storage for input, output, or working areas. A DS statement does not cause any data to be loaded with the object program.

The length of the field is defined by the first operand. This operand must be positive and may be an absolute value or a symbolic name. If a symbolic name is used, the symbol must previously have been defined as an absolute value, that is, it must have appeared in the label field in a statement of the source program preceding the one in which it is used. Address adjustment may be used with this operand.

The address in core storage of the field being defined may be assigned by the programmer or the programmer may let the processor assign the address. If the processor assigns the address, the statement is terminated after the first operand. If the programmer assigns the address, a second operand, which may be symbolic, asterisk, or actual, is used to establish the address of the field. Since data fields are addressed at their rightmost (low-order) digit, the processor assigns this position as the address of the field. Address adjustment may be used with the second operand. If the second operand is symbolic, it also must previously have been defined. Addresses assigned by the programmer do not disrupt the sequence of addresses assigned by the processor.

A DS statement may also be used to define a symbol without assigning any storage, ie., to define it as an absolute value. In this case, the first operand is omitted (or written as 0) and the second operand represents the value (may not exceed five digits in length). The second operand may be an actual value or a previously defined symbol. To define storage which will not be referred to symbolically, the label of the DS statement may be omitted.

The following statements define the field length only. When remarks are added to the statement, the field length must be followed by two commas.

Line	Label	Operation	Operands & Remarks
010	DELTAX	DS	7
020	DELTAX	DS	7,,TWO COMMAS REQUIRED FOR REMARKS

In the next example, the programmer assigns the address of the field and excludes the field length (the first operand) from the statement because it is without significance, and replaces it with a comma. The following statements cause the processor to associate the address 12930 with the label SUM:

Line	Label	Operation	Operands & Remarks
010	SUM	DS	,12930
020	SUM	DS	,12930,TWO COMMAS AGAIN REQUIRED

Again, in this example, two commas are required when remarks form part of the statement.

The following statement, which is similar to the one previously given, is assigned a value that is other than an address.

Line	Label	Operation	Operands & Remarks
010	FL	DS	,17,FLD LGTH FOR SUBSEQUENT STMTS

This statement defines the symbol FL as being equivalent to the value 17. Subsequent uses of this symbol are permitted because the symbol has been defined.

It should be noted that an area defined by the processor for a DS statement is always addressed at the rightmost position. However, to use this area for input/output, the leftmost digit must be addressed. This is done by using a DSS statement in place of a DS statement or by address adjustment with a DS statement, which subtracts a number that is one less than the length of the area from the address of the area. In a previous example, where DELTAX was defined as having a field length of 7, the operand of another instruction to read numerical data into the DELTAX field should be written as DELTAX –6.

DSS — Define Special Symbol (Numerical)

The DSS statement is similar to the DS statement with one exception: when the second operand is omitted, the processor assigns the leftmost position as the address of the field. If the second operand is assigned by the programmer, this address is assumed to be equivalent to the leftmost position of the field. A DSS statement is normally used to define a storage area for input/output. The data in such an area may be moved during execution of the object program by a Transmit Record instruction which requires that an address assigned to an area must be that of the leftmost position.

DAS — Define Alphameric Symbol

The DAS statement is similar to the DS statement with two exceptions:

1. The length specified by the first operand is automatically doubled by the processor to allow for alphameric data. Each alphameric character requires two storage positions.
2. The address of the field, if generated by the processor, is the leftmost position of the field plus one. The position is always odd-numbered, as it must be with any alphameric field.

The following example illustrates a DAS statement.

Line	Label	Operation	Operands & Remarks
010	TITLE	DAS	30
020	TITLE	DAS	30,,REMARKS REQUIRE TWO COMMAS

This statement defines an area for input/output that can contain 30 alphameric characters. The processor assigns 60 positions in core storage to accommodate alphameric coding. The output listing indicates this by typing 60 when this statement is assembled and listed. The omission of the second operand causes the processor to assign an address. During internal transmission of a field which utilizes an input-output area that is defined with a DAS, the area must be addressed at its rightmost position. In the example, the address may be achieved through address adjustment, i.e., TITLE + 2 * 30 – 2.

DC — Define Constant (Numerical)

The DC statement may be used to enter numerical constants into the object program, and, for ease of reference, to assign names to the constants. The label field contains the name by which the constant is known. DC statements consist of three operands. The first operand, which must be positive, indicates the length of the constant field; the second, the actual constant; the third, the storage address of the constant. The third operand is not used when the programmer prefers to let the processor assign the storage address. The assigned address is the rightmost storage position of the constant. The leftmost storage position is the position over which the processor places a flag.

Whenever remarks form a part of a DC statement, three commas must be included in the statement. The first and third operands may be symbolic or actual. They are subject to address adjustment. A symbolic address must previously have been defined to be valid.

If the first operand (length of constant) is smaller than the constant, an invalid condition results (see Error 10). If it is larger than the constant, zeros are inserted to the left of the constant so that the number of zeros plus the number of positions in the constant equals the length of the field (first operand).

A constant that is a positive number will be stored in the form of an unsigned integer; a negative number, in the form of a signed integer. A negative number has the minus sign written in front of the constant as part of the second operand. During assembly, a negative number produces a flag (minus sign) over the units position of the constant.

If the constant 0100000 and –0004337769 are required, they may be defined as follows:

Line	Label	Operation	Operands & Remarks
3 5	6 11	12 15	16 20 25 30 35 40 45 50
010	CONST1	DC	7,100000
020	CONST1	DC	7,100000,,3 COMMAS FOR REMARKS
030	CONST2	DC	10,-4337769
040	CONST2	DC	10,-4337769,,3 COMMAS FOR REMARKS

In both cases, constant 1 and constant 2, the length of the field is greater than the constant, and the address of these constants is assigned by the processor. These constants will appear in the object program as

$\bar{0}100000$
$\bar{0}00433776\bar{9}$

A record mark may be used in a constant but must be in the units position and must be written as the character @. The following example contains statements that:

1. Store a record mark by itself as a constant.
2. Store a constant 6 and record mark.
3. Store a minus 0773 and record mark.

Line	Label	Operation	Operands & Remarks
3 5	6 11	12 15	16 20 25 30 35 40 45 50
010	RMARK	DC	1,@,,STORE A RECORD MARK ONLY
020	CONSTX	DC	2,6@,,STORE A SIX AND RECORD MARK
030	CONSTY	DC	5,-773@,,STORE A MINUS 773 AND RM

These constants appear in the object program as:

‡
$\bar{6}$ ‡
$\bar{0}77\bar{3}$ ‡

A constant 7 with a flag ($\bar{7}$) is generated by either of the following statements:

Line	Label	Operation	Operands & Remarks
3 5	6 11	12 15	16 20 25 30 35 40 45 50
010	CONSTZ	DC	1,-7,,STORE A 7 WITH A FLAG
020	CONSTZ	DC	1,7,,STORE A 7 WITH A FLAG

A flag is always placed over a 1-digit constant (except a record mark) regardless of the sign (positive or negative). Therefore, the programmer must use two positions to define a positive 1-digit constant.

Constants may not exceed 50 characters. The following statement generates a constant containing 50 zeros.

Line	Label	Operation	Operands & Remarks
3 5	6 11	12 15	16 20 25 30 35 40 45 50
010	ZERO	DC	50,0,,STORE FIFTY ZEROS

To store a zero with a flag at location 401, the following statement can be used:

Line	Label	Operation	Operands & Remarks
3 5	6 11	12 15	16 20 25 30 35 40 45 50
010		DC	1,-0,401,STORE A ZERO WITH A FLAG

DSC — Define Special Constant (Numerical)

This statement is similar to the DC statement in that the first, third, and fourth operands are written in the same manner as a DC statement. The second operand (the constant itself) is assigned an address that corresponds to the leftmost digit of that constant. The constant may contain digits, flagged digits, and the @ character. The @ character is translated by the processor into a record mark and may appear only as the low-order character of the constant. The high-order digit of the constant is not automatically flagged by the processor. Flagged digits in the constant are specified by J-R for 1-9. A flagged zero is indicated by an 11-0 punch (card only). A flagged zero may not be used when input is from the paper tape reader or from the typewriter.

DVLC — Define Variable Length Constant

The DVLC statement permits the programmer to specify one or more constants of the same or different lengths with one statement. The statement requires a minimum of three operands; the first specifies the address of the low-order position of the first constant; the second, which must be positive, specifies the length of the first constant; the third specifies the constant itself. Each constant after the first will require two operands, one for the length and one for the constant. All operands may be actual or symbolic and may be arithmetically adjusted.

The first operand is used only if the programmer wishes to specify an address. If it is omitted, the processor will assign the address of the first constant. The total length of all constants must not exceed 50 digits.

Constants are flagged in the high-order position. The address of the low-order position of the first constant (third operand) is assigned to the label of the statement. Negative constants should be preceded by a minus sign. Symbols used in the constant operand need not have been previously defined, but may have appeared as labels in any part of the program. No remarks or @ characters are permitted within this statement.

The constant operands are treated as normal DSA operands. This means that the constant, if specified as an actual number, may not exceed five digits. However, in a DVLC statement, address adjustment may be used within the constant operand to obtain an output of up to ten significant digits. If the address adjustment results in a number greater than ten digits in length, only the ten *right-most* digits will be retained.

The following statement defines a constant of 10000 which can be referred to by two different labels:

Line (3–5)	Label (6–11)	Operation (12–15)	Operands & Remarks (16–50)
010	CONST	DVLC	NAME,5,10000

DAC — Define Alphameric Constant

To define a constant consisting of alphameric data, the operation code DAC is used. The DAC statement is similar to the DC statement with three exceptions:

1. The first operand (length) is automatically doubled by the processor to allow two storage positions for each alphameric character.
2. The storage address of the constant is the address of the leftmost position plus one. This address must be an odd-numbered address to comply with the requirements for alphameric data storage. An odd-numbered address will automatically be assigned, if it is assigned by the processor. If it is specified by the programmer (as in line 020 of the following example), the processor assigns the specified address and provides that the constant is stored beginning one position to the left of the specified address. In the latter case, the processor makes no test of whether or not the address is odd-numbered or whether the address (or the position to the left) has been previously assigned.
3. High-order zeros are not automatically inserted in the constant by the processor, as is the case with a DC statement when the field length exceeds the number of characters. The number of characters, including blank characters, should not be greater or less than the specified length (first operand). When the rightmost position or positions of the constant are blank characters, they should be followed by a comma or end-of-line character. For card input, the rightmost position must be followed by a comma or a record mark.

NOTE: Only DAC and DNB instructions may be used to insert blank characters into storage.

Line (3–5)	Label (6–11)	Operation (12–15)	Operands & Remarks (16–50)
010	NOTE1	DAC	17,DECK 3478 PUNCHED,,END MESSAGE
020		DAC	6, ,901,STORE 6 ALPHA BLKS
030	RMARK	DAC	1,@,,ALPHA RM FOR OUTPUT AREA
040	CONST	DAC	13,DELTAX=0.000@,,STORE CNST,RM

In the example shown:

1. Statement 010 uses 34 storage positions to store the 17-position constant (deck 3478 punched).
2. Statement 020 places 6 alphameric blanks into storage locations 900 through 911. Also, a flag is set in location 900.
3. Statement 030 records an alphameric record mark in storage.
4. Statement 040 places a 13-position constant, including a record mark, in storage.

A 50-character alphameric constant (maximum allowable) occupies 100 positions of storage. A flag is set over the leftmost position of the field. Addressing this constant for internal field transmission requires the address OUTPUT +50 * 2–2, where OUTPUT is the symbol (label) which represents the leftmost address plus one.

DSAC — Define Special Alphameric Constant

This statement is similar to the DAC statement with one exception. The constant in a DSAC statement is addressed by the low-order digit of the field. The high-order digit of the field is flagged as in a DAC statement.

DSA — Define Symbolic Address

The DSA statement may be used to store a series of up to ten addresses as constants, as part of the object program. These addresses can be used for instruction modification or for setting up a table of addresses through which the programmer may index to modify a routine.

Each entry (symbolic or actual) in the operands field, with the exception of the last entry, is followed by a comma. The equivalent machine address of each entry is stored as a 5-digit constant. The constants are stored adjacent to each other with a flag over the high-order position of each. The label field of this statement must contain the symbolic name by which the table of constants may be referred to. An address at which this table is stored in core storage may not be assigned by the programmer nor may any remarks be included in the DSA statement. The address assigned by the processor is the address at which the rightmost digit of the first constant will be located.

NOTE: If the last operand is followed by a comma, an additional zero address ($\bar{0}0000$) is assembled in the table.

In the example that follows, symbols ALPHA, ORIGIN, and OUTPUT are equivalent to addresses 3200, 3600, and 15000, respectively.

Line 3 5	Label 6 11	Operation 12 15	Operands & Remarks 16 20 25 30 35 40 45 50
010	TABLE	DSA	ALPHA,ORIGIN,3234,OUTPUT-50

The constants are stored as

$$\bar{0}3200\bar{0}3600\bar{0}3234\bar{1}4950$$

(06200)

(06204)

If the leftmost digit of these four constants is located at 06200, then the address equivalent to TABLE will be 06204, the location of the rightmost digit of ALPHA.

DSB — Define Symbolic Block

A DSB statement is used to define an area of storage for storing a numerical array. A DSB statement does not cause any data to be loaded with the object program. The label of this statement is converted to the address at which the first element of the array is stored (i.e., the rightmost position of the first element). The first operand indicates the size of each element; the second, the number of elements. Both operands must be positive.

Either or both operands may be symbolic or actual. If symbolic, the symbol must have been previously defined. A third operand is required if the programmer wishes to assign the address. For example, to store an array of 75 elements, with each element containing 15 digits, the statement used would be:

Line 3 5	Label 6 11	Operation 12 15	Operands & Remarks 16 20 25 30 35 40 45 50
010	ARRAY	DSB	15,75,7514

In this example, the array begins at location 07500 (leftmost position of the first 15-digit element). ARRAY is equivalent to 07514 (address of the first element).

DNB — Define Numerical Blank

A DNB statement is used to define a field of numerical blanks. (The 8-4 card code denotes a numerical blank.) Up to 99 blanks may be specified in each DNB statement. In addition to a label, two operands can be assigned by the programmer. The first of these specifies the number of blank characters desired (field length). This number must be positive. The second operand specifies the rightmost address of the field where the blanks are stored in the object program.

If the second operand is omitted and the statement is labeled, the address assigned to the label by the processor is the rightmost storage position of the blank field. The blank field does not contain a flag in its leftmost position.

If the programmer wishes to move a blank field in core storage, he must either define a single-digit constant with a flag bit in the position in front of the leftmost position of the blank field, or a record mark in the position following the rightmost position of the blank field.

If six numerical blanks are required, they may be defined as follows:

Line	Label	Operation	Operands & Remarks
010	BLANKS	DNB	6

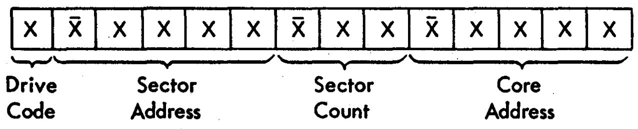
The processor assigns the storage address of the six blank positions to the label BLANKS. In the example that follows, the programmer assigns the storage address as 08625.

Line	Label	Operation	Operands & Remarks
010	BLANKS	DNB	6,8625,STORE SIX NUMERICAL BLANKS

In a DNB statement, two commas are required whenever remarks are included in the statement; the first after the length operand and the second after, or in place of, the address operand.

DDA — Define Disk Address

The DDA statement allows the programmer to define the disk control field of a seek, read, write, or check operation. This field is assembled as a 14-digit constant divided in the following manner:



The statement requires five operands, each of which may be actual or symbolic and may use address adjustment.

1. The first operand specifies the address of the high-order position (drive code) of the disk control field. This address must be even. If the operand is omitted, the processor assigns the next valid address. A comma indicating the end of the operand must be present.
2. The second operand specifies the module (drive) to be acted upon by the input/output or control instruction that addresses the disk control field. During assembly, this operand becomes the first digit of the 14-digit constant. If the digit is even or zero, the module to be used by the instruction is determined by the *second* digit (high-order digit of the sector address) of the constant; if it is odd, the module is determined by the digit itself. When the operand is in symbolic form, the low-order digit of the equivalent numerical value of the symbol becomes the first digit of the assembled 14-digit constant. See Example 1 (EX1).
3. The third operand specifies the 5-digit sector address (00000-79999) where reading, writing, or checking begins. From this address the computer determines the correct cylinder, head, and sector. It also determines the drive module unless overridden by the second operand.
4. The fourth operand indicates the number of sectors (1 to 200) to be read, written, or checked.
5. The last operand specifies the 5-digit core storage address used for data transfers to and from disk storage. Reading or writing begins at the specified address and extends into successively higher-numbered core storage locations. This address must be even.

Each of the last three operands assembles with a flag over the high-order digit. If a label is used with this mnemonic, it is assigned the address of the drive code.

EXAMPLES

Assume: M=02
DISK=14540
SECT=150
CORE=10000

Line	Label	Operation	Operands & Remarks
010	EX1	DDA	,M,DISK,SECT,CORE
020	FIELD	DDA	CYLD,M,16250,SECT+2,CORE

For these examples, the assembled output (disk control field) would appear as follows:

$$2\bar{1}4540\bar{1}50\bar{1}0000$$
$$2\bar{1}6250\bar{1}52\bar{1}0000$$
$$\uparrow$$

Address of FIELD and CYLD

DGM — Define Group Mark

A DGM statement is used to place a group mark at some specific address in core storage. It needs only one operand; the address where a group mark is desired. The operand can be actual, symbolic, or arithmetically adjusted. If no operand is specified, a group mark is placed in the next available core storage location. Only one group mark may be defined with each DGM statement.

DMES — Define Message (1710 Only)

The DMES instruction is designed to aid the 1710 user in programming the output control codes of the Serial Input/Output Channel (SIOC). With one DMES statement, a complete message of alphabetic, numerical, or mixed-mode data can be specified. Although this mnemonic is oriented to the SIOC output printer, it may be used to program other devices which are wired to accept the control codes of SIOC.

The DMES statement requires three operands; address, starting mode, and message. Remarks are not permitted.

1. The address operand may be used by the programmer to assign an address to a message, in which case he assumes the responsibility of correctly positioning the message if alphabetic output is indicated. If no particular address is desired, a comma must be placed in lieu of the address operand. The processor will then assign a valid address.
2. The second operand specifies the starting mode; an A indicates the alphabetic mode; the omission of the operand indicates the numerical mode. Even though this operand may specify an alphabetic starting mode, the correct positioning of the message is still dependent upon the address operand.
3. The message operand consists of alphabetic, numerical, or mixed-mode data, and special characters which have been assigned to the control functions for use in a DMES statement (see Table 2).

The special characters are enclosed in parentheses and inserted into the message wherever they are needed. (NOTE: Parentheses are *not* permitted in the "data" portion of a message operand.)

Table 2. DMES Representation of Output Printer Control Codes

DMES CHARACTER	ASSEMBLED CONTROL CODE		FUNCTION
	ALPHAMERIC	NUMERICAL	
(P)	0‡71	‡1	Type numerical period
(M)	0‡72	‡2	Change mode
(C)	0‡73	‡3	Type numerical comma
(B)	0‡74	‡4	Shift printer ribbon to black
(A)	0‡75	‡5	Shift printer ribbon to red (Alert)
(T)	0‡76	‡6	Tabulate printer carriage
(S)	0‡77	‡7	Space printer carriage one position
(R)	0‡78	‡8	Return printer carriage and advance one line
(F)	0‡79	‡9	Advance printing form to next form feed stop
(E)	0‡0‡	‡‡	End of message

If information is to be placed into a message sometime after it is stored, the space must be reserved by the programmer when the statement is written. This is normally done by writing zeros in the message in place of the data which is to be inserted later.

The length of the message need not be specified by the programmer; however, if the number of core storage locations needed to contain the message exceeds 100, an error is indicated. NOTE: Control characters require two core locations when in numerical mode and four locations when in alphabetic mode.)

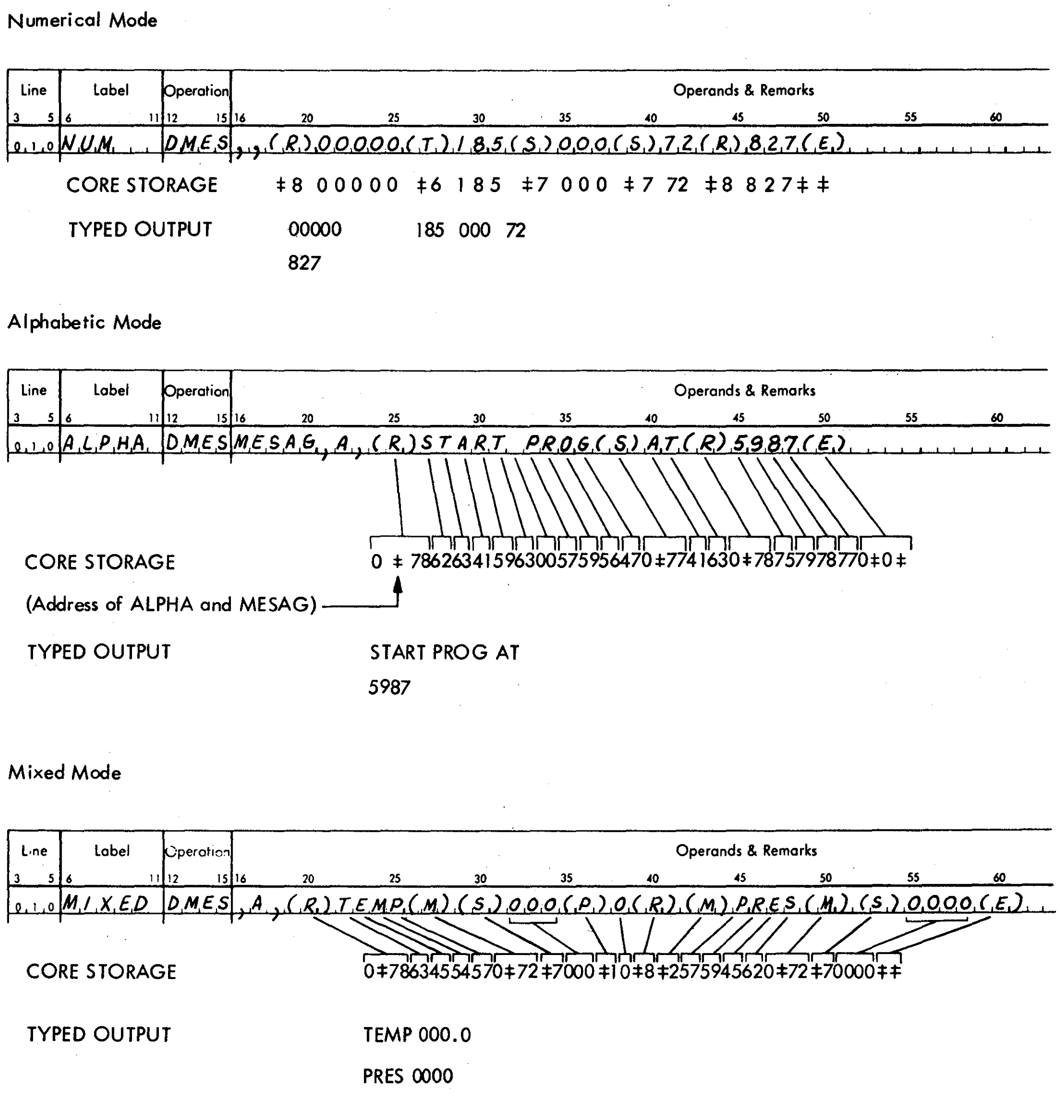
Flagged digits can be placed into a numerical message by substituting the letter J-R for 1-9. A flagged zero is indicated by a $\bar{0}$ (card only).

Some examples of DMES statements are shown in Figure 9. In the example of alphabetic mode, notice that the programmer has assigned the symbolic address MESAG and therefore is responsible for the correct positioning of the entire message. Since the message begins in the alphabetic mode, the address assigned to MESAG corresponds to the second digit of the stored data.

In the example of a mixed-mode message, zeros were placed in the message to reserve space for temperature and pressure readings. These are normally inserted in the message before it is typed out.

Invalid Mode Changes

Since the IBM 1717 Output Printer and the IBM 1620 console typewriter both require alpha messages to



Numerical Mode

Line	Label	Operation	Operands & Remarks
010	NUM	DMES	,,(R)00000(T)185(S)000(S)72(R)827(E)

```
CORE STORAGE      ‡8 00000 ‡6 185 ‡7 000 ‡7 72 ‡8 827‡ ‡

TYPED OUTPUT      00000      185 000 72
                  827
```

Alphabetic Mode

Line	Label	Operation	Operands & Remarks
010	ALPHA	DMES	MESAG,A,(R)START PROG(S)AT(R)5987(E)

```
CORE STORAGE                        0 ‡ 78626341596300575956470‡7741630‡78757978770‡0‡
(Address of ALPHA and MESAG)

TYPED OUTPUT                        START PROG AT
                                    5987
```

Mixed Mode

Line	Label	Operation	Operands & Remarks
010	MIXED	DMES	,A,(R)TEMP(M)(S)000(P)0(R)(M)PRES(M)(S)0000(E)

```
CORE STORAGE                        0‡78634554570‡72‡7000‡10‡8‡2575945620‡72‡70000‡‡

TYPED OUTPUT                        TEMP 000.0
                                    PRES 0000
```

Figure 9. Examples of SPS DMES Statements

start at *even* core addresses, an invalid message can be created when programming a change from numerical to alphabetic mode. The invalid mode change occurs when the alphabetic characters following the change are assigned addresses starting at an *odd* core location instead of an even core location. Such an assignment is incorrect and is indicated as an error. The user may either enter the *correct* DMES statement manually from the console typewriter, or he may allow the processor to:

1. Assemble the mode change (‡2).
2. Place a $\bar{0}$ in the location following the mode change (i.e., ‡2$\bar{0}$).
3. Assemble the remainder of the message.

Summary of Declarative Operations

As stated earlier, areas being defined by the processor are assigned core storage locations in the order in which they are processed. To do this, the processor program uses a location assignment counter to keep track of the address of the last assigned storage location. Table 15 in the Appendix shows the amount added to the location assignment counter for each instruction and summarizes the coding and operation of each declarative mnemonic.

1620/1710 Imperative Operations

Imperative operations may be divided into five classes:

1. Arithmetic
2. Internal data transmission
3. Branch
4. Input/Output and control
5. Miscellaneous

This section describes the five classes of imperative operations and gives some examples of statements written in symbolic language. For a detailed description of the function of each machine language instruction, refer to the appropriate machine reference manuals.

Arithmetic

Arithmetic operations are those that involve adding, subtracting, multiplying, or dividing. Table 16 in the Appendix is a list of all arithmetic mnemonics and a brief description of their P and Q address functions.

Some examples of arithmetic statements written in symbolic language follow:

Line (3–5)	Label (6–11)	Operation (12–15)	Operands & Remarks (16–50)
010		A	COST,LABOR
020		A	COST,LABOR,,ADD LABOR TO COST
030		SM	STORE+4,2,10
040		AM	88,05,10,HALF-ADJUST POSITIVE AMT
050		LD	97,DDND
060		D	86,DVR

These statements cause the following operations to be performed:

Line 010 – Add labor amount to cost amount.

020 – Same as line 010 except three commas are required for remarks.

030 – Subtract a constant 02 from the field located at STORE PLUS 4.

040 – Add a constant 05 to the field at storage location 00088.

050 – Move DDND (dividend) to the product area (storage location 00097).

060 – Divide the dividend by successive subtractions of the DVR (divisor) starting in storage location 00086.

Internal Data Transmission

Internal data transmission operations are those that cause the movement of data from one core storage location to another. They require both a P and a Q address. Table 17 in the Appendix lists all internal data transmission mnemonics and their P and Q functions.

Some examples of internal data transmission statements written in symbolic language follow:

Line (3–5)	Label (6–11)	Operation (12–15)	Operands & Remarks (16–50)
010		TD	FIELD,DIGIT
020		TDM	FIELD,3
030		TF	STORE,RATE1,,MOVE RATE 1 TO STORE
040		TFM	STORE,3525,,MOVE 03525 TO STORE
050		TFM	*-11,41,10,CHGE PREV OP CODE TO NOP
060		TNS	A,B,,CONVERT FLD A TO NUMER CODING
070		TNF	C,D,,CONVERT FLD D TO ALPHA CODING

These statements cause the following instructions to be executed:

Line 010 – A numerical digit at the location called DIGIT is moved to the location called FIELD.

020 – A digit 3 is moved to the location called FIELD.

030 – Rate 1 is moved to the field called STORE.

040 – A constant 3525 is moved to the location called STORE.

050 – A constant 41 is moved to the O_0 and O_1 positions of the preceding instruction in the object program.

060 – Field A is moved to field B and converted from alphameric coding (2 digits per character) to numerical coding (1 digit per character).

070 – Field D is moved to field C and converted from numerical coding to alphameric coding.

Branch

Branch operations are used to alter the normal sequence of instruction execution. They may be conditional or unconditional. Table 18 in the Appendix lists the branch mnemonics and their P and Q address functions. Also listed in this table are the two compare mnemonics. Compare operations, though arithmetic in nature, perform a distinctly logical function.

Some examples of branch and compare statements written in symbolic language follow:

Line	Label	Operation	Operands & Remarks
3 5	6 11	12 15	16 20 25 30 35 40 45 50
010		C	B,A,,COMPARE FIELD A WITH FIELD B
020		B	START,,,BRANCH TO LABEL START
030		BI	START,100,,IF SW1 ON, BR TO START
040		BC1	START,,,SAME AS LINE 030
050		BNC1	START+3*12,,,
060		BB	

These statements cause the following operations to be performed in the object program.

Line 010 — Compare field A with field B.

020 — Branch to an instruction labeled START.

030 — If Program Switch 1 is on, branch to the instruction labeled START.

040 — Same as line 030 with the exception that the unique mnemonic operation code used does not require a Q address.

050 = If Program Switch 1 is not on, branch to the third instruction following the one labeled START.

060 — Branch unconditionally to an instruction whose address is saved in IR-2 or PR-1.

Input/Output and Control

Input/Output operations enable the transfer of data between core storage and various I/O units; control operations do not affect data, but rather pertain to electro-mechanical operations of I/O units. Table 19 in the Appendix lists the input/output and control mnemonics and their P and Q functions.

Some examples of input/output and control statements written in symbolic language follow:

Line	Label	Operation	Operands & Remarks
3 5	6 11	12 15	16 20 25 30 35 40 45 50
010		WA	OUTPUT,100
020		WATY	OUTPUT,,,SAME AS LINE 010
030		K	,101,,SAME AS LINE 040
040		SPTY	

These statements cause the following operations to be performed in the object program:

Line 010 – Type out alphameric data from a storage location called OUTPUT.

020 – Same as line 010; however, a unique mnemonic is used.

030 – Single space on the typewriter.

040 – Same as line 030; however, a unique mnemonic is used.

Miscellaneous

Miscellaneous operations are those that do not fall into any of the operations described previously. Table 20 in the Appendix lists the miscellaneous mnemonics and their P and Q functions.

Some examples of miscellaneous statements written in symbolic language follow:

Line	Label	Operation	Operands & Remarks
3 5	6 11	12 15	16 20 25 30 35 40 45 50
010		CF	OUTPUT-5
020		MF	3352,2694,,
030		H	,,,HALT 1
040		NOP	

These statements cause four different operations to be performed in the object program as follows:

Line 010 – Clear a flag at the storage location, OUTPUT minus 5.

020 – Move a flag from storage location 2694 to storage location 3352.

030 – Cause the program to halt.

040 – Perform no operation but proceed to the next sequential instruction.

Processor Control Operations

The SPS language includes the following five control operations:

DORG	Define Origin
DEND	Define End
HEAD	Heading
TCD	Transfer Control and Load
TRA	Transfer to Return Address

These operation codes are orders to the processor that give the programmer control over portions of the assembly process. Specifically, DORG gives the programmer control over the placement of his program in storage. DEND, TCD, and TRA order the processor to produce unconditional branches to locations specified by the programmer. HEAD assigns unique characters to labels or symbols used within a source program.

With the exception of TRA and DORG, none of the preceding operations may be labeled.

DORG — Define ORiGin

The DORG statement instructs the processor to override its automatic assignment of storage and to begin the assignment of succeeding entries at the particular location specified by the programmer. In this way, the programmer is able to control assignment of storage to instructions, constants, and data.

A define origin statement is coded as follows:

Line	Label	Operation	Operands & Remarks
3 5	6 11	12 15	16 20 25 30 35 40 45 50
0 1 0		DORG	7820

This statement directs the processor to reset its location assignment counter to the particular address specified in the operand (actual or symbolic). This causes the assignment of succeeding entries to begin at this address. When an actual address is entered by the programmer, care must be taken to avoid inadvertent overlapping of areas assigned by the processor.

If the operand is left blank, assignment of storage starts with an address of 00000. Since the Monitor I System occupies locations 00000 through 02401, constants and instructions at object time cannot occupy these storage locations.

If a symbolic address is entered, it must appear as a label earlier in the program sequence. An * address refers to the current contents of the location assignment counter. A define origin statement can take any of the following individual forms:

Line	Label	Operation	Operands & Remarks
3 5	6 11	12 15	16 20 25 30 35 40 45 50
0 1 0		DORG	XYZ
0 2 0	ORIGIN	DORG	XYZ+50
0 3 0	ORIGIN	DORG	*+50, LOCATION ASSIGN COUNTER+50

If XYZ (label) is previously defined as 7002, the first entry directs the processor to begin the assignment of succeeding entries at location 7002. The second entry directs the processor to begin the assignment of succeeding entries at the location that has been assigned to the symbol XYZ plus 50. The symbol ORIGIN can be used at any point in the program to refer to that address. The third entry directs the processor to begin the assignment of succeeding entries at the address specified by the current contents of the location assignment counter plus 50. A comma must follow the operand when remarks are included in the DORG statement.

DEND — Define END

The DEND statement is the last statement entered in the source program; it informs the processor that all statements of the source program have been processed. The DEND statement requires the presence of an operand representing the starting address of the program. The operand may be actual or symbolic.

The following statement illustrates a DEND statement.

Line	Label	Operation	Operands & Remarks
3 5	6 11	12 15	16 20 25 30 35 40 45 50
0 1 0		DEND	START

HEAD — HEADing

It is frequently convenient, and sometimes necessary, to write a source program piecemeal and to assemble these pieces into the total program. Parts of the program may be written by different programmers, or by the same programmer at different times with considerable time lapses between.

Suppose, in such a situation, that a program block, say B_1, has been written; that another program block, B_2, is in the course of being written; and that B_1 and B_2 eventually are to be joined to compose a single program. Certain symbols may already have been used to write block B_1, and certain symbols, varying from the symbols used in B_1, may be used to write block B_2. To avoid duplication of symbols in each block, the programmer writing block B_2 must be concerned with the symbols used in B_1.

Symbols used in block B_2 can duplicate those in B_1, provided they are less than six characters in length and have been prefaced by a HEAD statement. The programmer can completely ignore the symbols in B_1 by prefacing B_2 with the following control statement:

Line	Label	Operation	Operands & Remarks
3 5	6 11	12 15	16 20 25 30 35 40 45 50
0 1 0		HEAD	X

where the single character X may be any one of the characters A to Z, 0 to 9, or blank.

The control statement, HEAD X, generates no instructions or data in the object program. When the processor encounters a HEAD statement, it treats the symbols in the label or operands fields of the following statements, provided the symbols are less than six characters in length, as though they were headed by the character X. The processor continues to do this until it encounters another head statement.

Thus, the symbols used in block B_1 which contain less than six characters cannot possibly conflict with the symbols used in block B_2. Six-character symbols are not affected, that is, a 6-character label, COMMON, following the control statement HEAD 9 is not treated as 9 COMMON, for it would be a 7-character symbol, and only a maximum of six characters can be handled by the symbol table.

A symbol is said to be "unheaded" if, and only if, its representation uses exactly six characters. The symbol COMMON, for example, is unheaded. The symbol ALPHA whose length is less than six characters is considered to be headed, whether under a HEAD control statement or not. If ALPHA is under control of HEAD X, then ALPHA is said to be "headed by X." If ALPHA is not under control of any HEAD statement, then ALPHA is said to be "headed by blank."

A symbol, ALPHA, headed by the character X, is not identical to the symbol XALPHA. The heading character is essentially on a different level from the characters which make up the symbol. However, ALPHA headed by a blank should be regarded as identical to the symbol ALPHA used without a heading statement.

If a HEAD statement with a nonblank character does not occur in the entire source program, all considerations of heading can be ignored. This is the reason for not introducing the concept of headed symbols earlier.

A HEAD statement with a blank character must be used if the programmer desires to modify the heading process in the example. Note that the statement HEAD and the statement HEAD 0 are quite different. For example, if blocks B_1 and B_2 are to be joined in one program, and B_2 must be nested somewhere in the middle of B_1, as follows:

```
Operation      Operands
    .          .
    .          .  } first part of block B1
    .          .
  HEAD         X
    .          .
    .          .  } block B2
    .          .
  HEAD
    .          .
    .          .  } second part of block B1
               .
```

the entire program might have been prefaced by a HEAD statement with a blank character operand. As implied previously, however, such a HEAD instruction is superfluous, since the symbols in the first part of block B_1 are automatically headed by blank, being under the control of no HEAD instruction at all.

Often it is inconvenient to refer to a symbol that is defined in another headed region because of the requirement that the symbol be six characters in length. To facilitate cross referencing between headed blocks, the following convention can be used:

Suppose that a symbol, say SUM, under HEAD 1, has been defined by some instruction. Suppose further that this symbol is to be referred to in an instruction under the control of the instruction HEAD 2. Then the desired reference can be made by writing 1$SUM as it appears in the following instruction.

Line 3 5	Label 6 11	Operation 12 15	Operands & Remarks 16 20 25 30 35 40 45 50
0,1,0		A	TOTAL,,1$SUM

In general, if the two characters "C$," where C is any allowable heading character, are placed in front of the headed reference symbol SUM, then the result is SUM headed by C. To specify SUM headed by blank, one simply writes $SUM, with no character preceding the $ character.

If the processor finds an operand containing a 6-character symbol plus a head character, such as 9COMMON, the processor will produce an error message indicating that the symbolic address contains more than six characters.

If a label is used in a HEAD statement, it is ignored.

TCD — Transfer Control and loaD

The TCD statement may be used to cause the loader to execute an unconditional branch instruction. When this statement is encountered during the loading of the object (machine language) program, it causes the loader to break the normal loading process and to branch to the location (ADDR) specified in the operand.

Line 3 5	Label 6 11	Operation 12 15	Operands & Remarks 16 20 25 30 35 40 45 50
0,1,0		TCD	ADDR

ADDR may be actual or symbolic.

This statement allows programs which are too large to fit into core storage to be loaded and executed piecemeal, by terminating each piece with a TRA statement. In effect, a TCD instruction can be used in conjunction with a DORG statement to execute portions of the program that have already been loaded into storage and to overlap these with other instructions.

During assembly, the TCD instruction does not affect the location assignment counter or alter the symbol table.

TRA — Transfer to Return Address

The TRA statement causes the normal loading sequence of an object program to be resumed once it has been broken by a TCD statement.

This processor control operation increments the location assignment counter by 42. The last statement of that part of a source program that is executed, when loading is interrupted by a TCD statement, must be a TRA statement. When the TRA instruction equivalences are encountered in the object program, the normal loading process is resumed. The TRA statement, which takes the following form, uses no operands.

Line	Label	Operation	Operands & Remarks
3 5	6 11	12 15	16 20 25 30 35 40 45 50
010		TRA	

The following example illustrates the use of the TCD and TRA mnemonics.

Line	Label	Operation	Operands & Remarks
3 5	6 11	12 15	16 20 25 30 35 40 45 50
010	START	(First	instruction)
020		•	• • •
030		•	• • •
040		•	• • •
050		•	• • •
060		TRA	
070		TCD	START
080		DORG	START
090		(Remaining	Instructions)
100		•	• • •
110		•	• • •
120		•	• • •

The TCD statement causes a branch to the location assigned to the symbol START, followed by the execution of instructions from START through the TRA statement. The TRA statement causes a branch to the load program, which resumes loading the remainder of the object program beginning with the location labeled START.

Product-Area Mnemonics

Two mnemonics, SAVE and RSTR, are provided to allow 1710 interrupt programs to make use of the product area (locations 00080 through 00099) during their operation, even though that area was being used by the main program when the interrupt occurred. The SAVE mnemonic should be inserted at the beginning of the interrupt routine where it will generate in-line instructions to store the contents of the product area in a specified location. The interrupt routine may then use the area but must restore the original contents (RSTR) before returning control to the main program. The P and Q operands of these statements may be actual or symbolic and may be arithmetically adjusted.

SAVE

EXAMPLE

```
        XFER     SAVE     A, B
```

where XFER is the address of the first instruction generated by the use of the SAVE mnemonic.

A is the symbolic address of the leftmost position in the temporary storage area. This area can be reserved with a DSS statement. The length of the area must be N + 1 digits where N is the number of digits in the product area to be saved.

If LOCA is the label of the first digit of a 26-digit area in core storage, the statement:

```
        A10      SAVE     LOCA, 75
```

will cause the contents of the product area from address 00075 to 00099 to be stored in the first 25 locations reserved at LOCA. A record mark will be placed in the 26th location.

If LOCB is the label of the first digit of a 21-digit area in core storage, the statement:

```
        A100     SAVE     LOCB
```

will store the contents of the product area from address 00080 through 00099 and a record mark in the 21 locations reserved at LOCB. The SAVE mnemonic generates the following instructions:

```
        TDM           00100,0
        DC            1,@,*
        TR            LOCB, 00080
        TDM           00100, 0
```

RSTR

EXAMPLE

```
        XFER     RSTR     A, B
```

where XFER is the address of the first instruction generated by the RSTR mnemonic. A represents the location in the product area into which the leftmost digit of the temporarily stored information is placed. This address should always be less than 99. If it is left blank, the product area between 00080 and 00099 will be filled in with the stored data.

B represents the address of the leftmost digit in the record of information to be returned to the product area.

If LOCA is the label of the first digit of an area where locations 75-99 have been stored by a SAVE instruction, the statement:

```
        RSTR     75, LOCA
```

will cause the product area from addresses 00075 through 00099 to be restored with the information in the first 25 locations reserved at LOCA.

If LOCB is the label of the first digit of an area where locations 80-99 have been stored by a SAVE instruction, the statement:

```
        RSTR       , LOCB
```

will return the information which was stored in the 21 locations reserved at LOCB to product area addresses 00080 through 00099.

The RSTR mnemonic generates the following instructions:

```
        TR        00080, LOCB
        TDM       00100, 0
```

Since each of these two mnemonics generate in-line instructions, care should be taken when using address adjustment in the same areas where these mnemonics are used.

1620 Subroutines

A program or routine consists of a set of coded instructions arranged in logical sequence; it is used to direct the 1620, 1710, or any IBM data processing system to perform a desired operation or series of operations. Generally, programs contain one or more short sequences of instructions that are parts or subsets of the entire program, and that are used to solve a particular part of a problem. These parts of the program or routine are called *subroutines.*

Usually, a subroutine performs a specific function, is common to a number of programs, and may be executed several times during the course of the program of which it is a part (main program). For example, a subroutine that extracts the square root of a number may be required during the execution of a pipe stress analysis program. The same subroutine may be used to extract a square root in a bridge and truss design program.

Classification of Subroutines

An efficient programming procedure is obviously one in which all necessary subroutines are coded only once, are retained on file, and are incorporated into a program whenever the operation performed by the subroutine is required. IBM Programming Systems has developed, for the SPS II-D Symbolic Programming System, a group of subroutines that are more frequently required because of their general applicability. Seventeen subroutines are available; they fall into three general catgories: arithmetic, data transmission, and functional.

Arithmetic subroutines
- Floating-Point Add
- Floating-Point Subtract
- Floating-Point Multiply
- Floating-Point Divide
- Fixed-Point Divide

Data Transmission subroutines
- Floating Shift Right
- Floating Shift Left
- Transmit Floating
- Branch and Transmit Floating

Functional subroutines (those that evaluate)
- Floating-Point Square Root
- Floating-Point Sine
- Floating-Point Cosine
- Floating-Point Arctangent
- Floating-Point Exponential (natural)
- Floating-Point Exponential (base 10)
- Floating-Point Logarithm (natural)
- Floating-Point Logarithm (base 10)

The methods used by the functional floating-point subroutines to evaluate the functions of arguments are shown in Table 3.

The subroutines are written in machine language and are provided in card or paper tape form for floating-point numbers with either a fixed-length or

Table 3 SPS Subroutine Method of Evaluating Arguments

SUBROUTINE	METHOD	
	FIXED LENGTH	VARIABLE LENGTH
Square Root	Odd integer	Odd integer
Sine and Cosine	Based on Hastings' approximation*	Series approximation
Arctangent	Truncated series	Series approximation for arctangent
Exponential (natural and base 10)	Hastings' approximation 10^B. 10^B is converted to e^B	Series approximations of 10^B and convert to e^B
Logarithm (natural and base 10)	Truncated series for ln B. ln B is converted to log 10^B	Series approximation of ln B and convert to log B

*Hastings, Cecil Jr., Approximations for Digital Computers, Princeton University Press, Princeton, New Jersey. The Rand Corporation, 1955.

variable-length mantissa. The terms "variable length" and "fixed length," as applied to subroutines in this manual, refers to the number of digits (L) in the mantissa, not to the length of the subroutine itself.

The four sets of subroutine card decks or paper tapes with their identifying set numbers follow.

Subroutine Set	Set Number
Divide subroutine for machines *not* equipped with the Automatic Divide feature. Does not call out or utilize the PICK subroutine	00
Fixed-length subroutine for machines equipped with the Automatic Divide feature.	01
Variable-length subroutines for machines equipped with the Automatic Divide feature.	02
Variable-length subroutines for machines equipped with the Automatic Floating-Point feature (The Automatic Divide feature is a prerequisite).	03

A PICK subroutine is included at object time when any of the seventeen subroutines previously mentioned have been called by the object program. This subroutine performs the function of obtaining the data specified for a subroutine, storing the result produced by that subroutine, and furnishing a return address to the mainline program.

In addition to the Library subroutines, the user may include up to twelve subroutines of his own. The method used to incorporate these routines into the proper subroutine set on disk is explained under ADDING SUBROUTINES. Subroutines appear with the object program only at execution time.

Subroutine Macro-Instructions

All linkages for the 1620 subroutines are generated automatically through the use of certain macro-instructions. The programmer places the macro-instruction, related to a particular subroutine, in the source program at the point where the subroutine is desired. This causes the SPS II-D processor, during assembly, to generate linkage to the desired subroutine. In addition, the processor arranges for the subroutine to be added to the object program at object time.

The data and addresses required by the subroutine and supplied in the macro-instruction are incorporated into the linkage instructions where they are made available for use. In this way, the subroutine obtains the information it requires to perform its given task and also to compute a return address to the main program. Control is returned to the main program at the completion of the subroutine by transferring to the return address (first instruction after the macro-instruction). The macro-instruction statement related to each subroutine is as follows:

Arithmetic Subroutines

Line 3 5	Label 6 11	Operation 12 15	Operands & Remarks 16 20 25 45 50
010		FA	A, B (Floating Add)
020		FS	A, B (Floating Subtract)
030		FM	A, B (Floating Multiply)
040		FD	A, B (Floating Divide)
050		DIV	A, B, A1, B1 (Divide)

Data Transmission Subroutines

Line 3 5	Label 6 11	Operation 12 15	Operands & Remarks 16 20 45 50
010		FSRS	A, B (Floating Shift Right)
020		FSLS	A, B (Floating Shift Left)
030		TFLS	A, B (Transmit Floating)
040		BTFS	A, B (Branch and Transmit Floating)

Functional Subroutines

Line 3 5	Label 6 11	Operation 12 15	Operands & Remarks 16 20 40 45 50
010		FSQR	A, B (Floating Square Root)
020		FSIN	A, B (Floating Sine)
030		FCOS	A, B (Floating Cosine)
040		FATN	A, B (Floating Arctangent)
050		FEX	A, B (Floating Exponential, Natural)
060		FEXT	A, B (Floating Exponential, Base 10)
070		FLN	A, B (Floating Logarithm, Natural)
080		FLOG	A, B (Floating Logarithm, Base 10)

In the arithmetic statements, the B operands represent the addresses of quantities to be added, subtracted, etc. For the fixed-point divide routine, two additional operands, A1 and B1 are required. These operands, as well as the A and B operands for data transmission statements, are explained in greater detail under each macro-instruction as it is described.

In the case of functional subroutine macro-instructions, the B operand represents the address of the argument to be evaluated, while the A operand represents the address where the result is placed in storage.

When using a macro-instruction, the programmer must code the exact number of operands required for that macro-instruction. Every macro-instruction used with subroutines supplied by IBM Library Services has at least two operands. Added subroutines may have macro-instructions with up to nine operands. Remarks and flag operands are not permitted in macro-instructions. Omitted operands require the insertion of commas as in imperative statements.

All operands in macro-instructions may be symbolic or actual; all are subject to address adjustment. If an * is used as an operand, its address is that of the leftmost position of the first linkage instruction.

Many subroutines have been paired (i.e., add and subtract, sine and cosine, natural and base 10 exponential, natural and base 10 logarithm) into single subroutines to conserve storage by sharing those program steps common to both. The individual subroutines within each pair are distinguished from each other solely by the point at which they are entered. The correct entry point is obtained through the use of the macro-instruction pertaining to the particular subroutine desired.

All subroutines are identified by a 5-digit code. This code identifies the subroutine as follows:

```
                     |X X|   |X X|   |X|
Set Number-------------┘        |      |
Subroutine Number---------------┘      |
Number of Entries----------------------┘
```

Table 4 shows the pairing arrangements of the subroutines together with their respective identification numbers and their sequence in core when used by an object program.

NOTE: The location of the PICK routine is determined by the last address assigned by the SPS processor during assembly. All other required subroutines will follow PICK (in core) in the sequence of Table 4. Subroutines not required are omitted.

Table 4. SPS Subroutine Group and Identification Numbers

SUBROUTINE	IDENTIFICATION NUMBERS			
	FIXED LENGTH		VARIABLE LENGTH	
	Without Automatic Divide	With Automatic Divide	With Automatic Divide	With Automatic Floating Point
PICK		01001	02001	03001
DIV	00011	01011	02011	03011
FA		01022	02024	03024
FS				
FM		01041		
FD		01051		
FSQR		01061	02061	03061
FCOS		01072	02072	03072
FSIN				
FATN		01091	02091	03091
FEXT		01102	02102	03102
FEX				
FLOG		01122	02122	03122
FLN				
FSRS		01141	02141	03144
FSLS		01151	02151	
TFLS		01161	02161	
BTFS		01171	02171	

The assigned address may be determined by inserting a DORG statement, with the desired address, immediately preceeding the DEND statement.

Linkage

For each macro-instruction statement in a source program, two machine language linkage instructions, a 5-digit address for each operand, and a record mark are generated by the processor in the object program. These linkage instructions replace the macro-instruction which never appears in the object program. A label written with a macro-instruction references the leftmost position of the first linkage instruction generated. If the programmer wishes to use this label in address adjustment, he must remember that the location of the instruction following a macro-instruction is not LABEL + 12.

The linkage instructions generated by the processor for a macro-instruction are *equivalent* to the following series of symbolic instructions:

Line	Label	Operation	Operands & Remarks
010		TFM	PCK+10,*+19
020		B7	SUBR,,6
030		DSA	A,B
040		DC	1,@

In this sequence of instructions,

PCK is the address of a fixed work area that is used to contain the operands of the macro-instruction.

SUBR is a fixed location that contains the address of the desired subroutine.

A, B . . . are the 5-digit addresses that are equivalent to the operands specified in the macro-instruction.

Floating-Point Arithmetic

Scientific and engineering computations frequently involve lengthy and complex calculations necessitating the manipulation of numbers that may vary widely in magnitude. To obtain a meaningful answer, problems of this type usually require retention of as many significant digits as possible during calculation, and correct positioning of the decimal point at all times. When the computer is used for such problems, several factors must be considered, the most important of which is the location of the decimal point.

In general, a computer does not recognize the presence of a decimal point in any quantity during calculation. A product of 414154 results whether the factors are 9.37 x 44.2, 93.7 x .442, or 937 x 4.42, etc. The programmer must be cognizant of the location of the decimal point during and after the calculation and arrange the program accordingly. In adding, the decimal points of all numbers must be lined up to obtain the correct sum. The programmer facilitates this arrangement by shifting the quantities as they are added. In the manipulation of numbers that vary greatly in magnitude, it is conceivable that the resulting quantity could exceed allowable working limits.

Processing numbers which are expressed in ordinary form, e.g., 427.93456, 0.0009762, 5382, –623.147, 3.1415927, etc., can be accomplished on a computer only with extensive analysis to determine the size and range of intermediate and final results. The percentage of time required for this analysis and subsequent number scaling is frequently much larger than the percentage of time required to perform the actual calculation. Moreover, number scaling requires complete and accurate information regarding the bounds on the magnitude of all numbers that come into the computation (input, intermediate, output). Since prediction of the size of all numbers in a given calculation is not always possible, analysis and number scaling are sometimes impractical.

To alleviate this programming problem, a system must be employed which provides information regarding the magnitude of all numbers in the calculation along with the quantities in the calculation. Thus, if all numbers are represented in some standard predetermined format that instructs the computer in an orderly and simple fashion as to the location of the decimal point, and if this representation is acceptable to the routine that performs the calculation, then quantities that range from minute fractions having many decimal places to large whole numbers having many integer places can be handled. The arithmetic system most commonly used, in which all numbers are expressed in a format that has these characteristics, is called "floating-point arithmetic."

The notation used in floating-point arithmetic is basically an adaptation of the scientific notation that is widely used today. In scientific work very large or very small numbers are expressed as a number, between one and ten, times a power of ten. Thus,

427.93456 is written as 4.2793456 x 10^2

and

0.0009762 is written as 9.762 x 10^{-4}

In the 1620 floating-point arithmetic system, the range of the fractional part of the number is modified to extend between .10000000 and .99999999, that is, the decimal point of all numbers is placed to the left of the high-order (leftmost) nonzero digit. Hence, all quantities may be thought of as a decimal fraction times a power of ten. For example,

427.93456 becomes .42793456 x 10^3

and

0.0009762 becomes .97620000 x 10^{-3}

where the fraction is called the *mantissa,* and the power of ten, indicating the number of places the decimal point was shifted, is called the *exponent.* The use of floating-point numbers during processing, besides offering advantages inherent in scientific notation, eliminates the need for analyzing operations in order to determine the positioning of the decimal point in intermediate and final results, since the decimal point is always immediately to the left of the high-order, nonzero digit in the mantissa.

Format

In 1620 floating-point operations, a floating-point number is a field consisting of a variable-length or fixed-length mantissa and a 2-digit exponent. The exponent is in the two low-order positions of the field, and the mantissa is in the remaining high-order positions, as shown:

$\bar{M}$ .... $M\bar{E}E$

For the subroutines, the variable-length mantissa may have a minimum of two digits and a maximum of 45 digits. Two operand fields that are added together must have mantissas of the same length. A flag over the high-order digit marks the extremity of the field. A fixed-length mantissa must have eight digits.

The exponent is established on the premise that the mantissa is less than 1.0 and equal to or greater than 0.1. The exponent always consists of two digits ranging between −99 and +99. A flag over the high-order (tens) digit defines the exponent.

The high-order digit of the mantissa and the high-order digit of the exponent *must* contain flag bits to operate properly with floating-point subroutines.

The mantissa and the exponent, if negative, must have an algebraic sign, represented by a flag, over the units position of the respective fields; if they are positive, they are not flagged. A floating-point number with a negative mantissa and a negative exponent is represented as follows:

$\bar{M}$ .... $\bar{M}\bar{E}\bar{E}$

Sign control of the results of all computations is maintained according to the standard rules of arithmetic operations.

Normalizing

In all floating-point numbers, the decimal point is assumed to be at the left of the high-order digit, which must be a nonzero digit. Such a number is referred to as *normalized.* When a number has one or more high-order zeros, it is considered to be *unnormalized,* unless the number itself is zero. An unnormalized number resulting from a floating-point subroutine computation is normalized automatically, but unnormalized terms are not recognized as such when entered as data. Therefore, it is necessary for all data to be entered in normalized form. Although unnormalized numbers will be processed, correct results cannot be assured. For example, the number $\bar{0}682349\,4\bar{0}5$ should be entered as $\bar{6}823494\,0\bar{0}4$, assuming the fixed-point number is 6823.494 and an 8-digit mantissa is required.

The following examples demonstrate the conversion of numbers in ordinary form to 1620 floating-point notation for an 8-digit mantissa.

Number	Normalized	1620 Floating Point
123.45678	.12345678 x 10^{3}	$\bar{1}2345678\bar{0}3$
.00765438	.76543800 x 10^{-2}	$\bar{7}6543800\bar{0}\bar{2}$
−.12348693	−.12348693 x 10^{0}	$\bar{1}234869\bar{3}\bar{0}0$
−.00000070	−.70000000 x 10^{-6}	$\bar{7}000000\bar{0}\bar{0}\bar{6}$
.00000000	.00000000 x 10^{-99}	$\bar{0}0000000\bar{9}\bar{9}$

NOTE: A zero mantissa is associated with a $\bar{9}\bar{9}$ exponent. With any other representation of zero, accuracy cannot be assured.

The result of a floating-point operation is normalized automatically. For example, the result .00123456 when normalized becomes $\bar{1}23456NN\bar{0}\bar{2}$, where N is an inserted digit (0 through 9) and $\bar{0}\bar{2}$ is the exponent. The value of the N digit (sometimes referred to as a noise digit) is determined by the programmer, who in most cases will choose to use zero. At object time the noise digit can be found at location 02401.

Effects of Normalizing

In normalizing, certain low-order digits in the mantissa may lose significance. To recognize these digits, the floating-point arithmetic can be performed twice, using a different N digit for each run, e.g., zero for the first run and nine for the second run. The significance of these digits can be readily distinguished by comparing the two results. For example, if the programmer compares the following:

	Mantissa	Exponent
Result, 1st run	.12345000	04
Result, 2nd run	.12345099	04

he will see that the two low-order positions of the mantissa have lost significance because they are significantly different.

When intermediate floating-point results enter into additional floating-point calculations, inserted digits may become a part of the result of the additional calculation.

In the case of lengthy computations using floating-point results, precision gradually decreases because of truncation. The magnitude of the truncation error depends on the individual computation process and cannot be predicted without a knowledge of the process in question. However, the truncation error in such cases is usually no greater than the degree of error present in a rounded amount. Results in floating-point subroutines are not rounded. The maximum truncation error for a fixed-length mantissa will not exceed 10^{-8} or, for a variable-length mantissa, 10^{-L}, except under certain conditions described in the explanation of floating-point functional subroutines.

Exponent Overflow and Underflow

In the 1620 floating-point subroutines, numbers with a magnitude equal to or greater than 10^{99} create a condition called *exponent overflow;* those with a magnitude of less than 10^{-99} create a condition called *exponent underflow.*

If either of these conditions is generated as a result of an arithmetic operation, the resultant field is set to the most reasonable value under the circumstances, and operation is resumed (see Table 5). The programmer is not given a visual indication when an error occurs; however, a core storage location (00401) is set to reflect the type of error. The programmer can interrogate this location for the following digits:

‡ — no error
$\bar{1}$ — exponent overflow
1 — exponent underflow
$\bar{0}$ — value cannot be calculated
0 — loss of accuracy in FSIN or FCOS, or negative input argument to FSQR or FLN

Location 00401 is reset to the "no error condition" (‡) by the pick subroutine; therefore if it is to be interrogated, the interrogation must be done before a new subroutine is entered.

When the digit in location 00401 indicates that an error has occurred, the user will most likely initiate some corrective action. The information that follows should be of some assistance if this situation arises.

Table 5. SPS Subroutine Errors

DESCRIPTION OF ERROR	CONTENTS OF 00401	RESULTANT FIELD
FA or FS, exponent overflow	$\bar{1}$	$\bar{9}9. . .9\bar{9}9$
FA or FS, exponent underflow	1	$00. . .0\bar{9}\bar{9}$
FM, exponent overflow	$\bar{1}$	$\bar{9}9. . .9\bar{9}9$
FM, exponent underflow	1	$\bar{0}0. . .0\bar{9}\bar{9}$
FD, exponent overflow	$\bar{1}$	$\bar{9}9. . .9\bar{9}9$
FD, exponent underflow	1	$\bar{0}0. . .0\bar{9}\bar{9}$
FD, attempt to divide by zero	$\bar{0}$	If $0\overline{)n}$: mantissa of dividend unchanged, exponent of dividend = E + 99. If $0\overline{)0}$: mantissa of dividend unchanged, exponent of dividend = -99
FSQR, input argument is negative	0	$\sqrt{\lvert x \rvert}$
FSIN or FCOS, input argument has exponent value greater than the mantissa length	$\bar{0}$	$\bar{0}0. . .0\bar{9}\bar{9}$
FSIN or FCOS, input argument has exponent value (X) such that $03 \leq X \leq L$, where L is the mantissa length	0	SIN (X) or COS (X)
FEX or FEXT, exponent overflow	$\bar{1}$	$\bar{9}9. . .9\bar{9}9$
FEX or FEXT, exponent underflow	1	$\bar{0}0. . .0\bar{9}\bar{9}$
FLN or FLOG, attempt to take log of zero	$\bar{0}$	$\bar{9}9. . .\bar{9}\bar{9}9$
FLN or FLOG, input argument is negative	0	LN ($\lvert x \rvert$) or LOG ($\lvert x \rvert$)

Symbolic Addresses	Description of Data
PCK + 10	Return address of mainline program
PCK + 15	Address of A operand data (result)
PCK + 20	Address of B operand data
PCK + 33	Mantissa length in use
PCK + 36	Noise digit in use

NOTE: PCK = 02365, a fixed location in core storage at object time.

Overflow and/or underflow conditions can arise in only six of the floating point subroutines presented in this manual, namely, the four arithmetic subroutines and the two exponential functional subroutines.

Arithmetic Indicators

During the execution of *arithmetic* subroutines, the overflow, high/positive, and equal/zero indicators are used. The overflow indicator is always reset at the beginning of each arithmetic subroutine. If it is desired to determine its status prior to the execution of an arithmetic subroutine, the indicator must be tested and its condition stored before the linkage instructions are executed. The high/positive and equal/zero indicators are set according to the mantissa of the result. Whenever a zero mantissa results ($\bar{0} \ldots\ldots 0\bar{9}\bar{9}$), the equal/zero indicator is turned on.

At the conclusion of a functional subroutine, the status of the high/positive, equal/zero, and overflow indicators does not necessarily reflect the result of the operation, because the indicators are disturbed during the execution of a functional subroutine. Therefore, their status at the conclusion of a functional subroutine should not be assumed to be the same as it was prior to the execution of the subroutine.

Description of 1620 Subroutines

In this section the various subroutines are described together with examples of how the associated macro-instructions are written. For average execution times of all subroutine macros, refer to Table 23 in the Appendix.

PICK

This subroutine is common to all fixed-length and variable-length subroutines. The pick subroutine, during execution of the object program:

1. Sets up A and B operands to be operated upon, calculates the return address to the mainline program, and then returns to the user's subroutine.
2. Resets location 00401 to "no error condition" ($\ddagger$).
3. Stores the calculated result in the proper storage area and branches back to the mainline program. This function is used as required by the individual subroutine.
4. Provides constants and working storage for the other subroutines.

The average execution time for the pick subroutine can be determined by the formula:

$$\text{Average time (in } \mu\text{sec)} = 40\text{ L} + 4640$$

where L = the length of the mantissa. The numbers are expressed in microseconds. Therefore, an 8-digit mantissa (same as fixed-length mantissa) requires 4960 μsec.

$$\begin{array}{rl} 40 \times 8 = & 320 \\ & \underline{4640} \\ & 4960\ \mu\text{sec} \end{array}$$

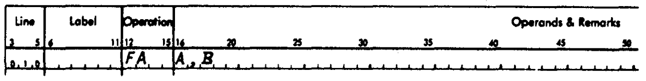
or approximately five milliseconds (ms).

NOTE: For the variable length subroutines used with the Automatic Floating-Point feature:

$$\text{Average time (in } \mu\text{sec)} = 40\text{ L} + 2080$$

Floating Add

Macro-instruction



Line 3–5	Label 6–11	Operation 12–15	Operands & Remarks 16–50
010		FA	A,B

The A and B addresses refer to the units position of the exponent of the fields:

$$\bar{\text{M}}\text{MMMMMMM}\bar{\text{E}}\text{E}$$

address of field

where the E's represent digits of the exponent and the M's represent digits of the mantissa. Neither A nor B should reference any location within the Product Area (i.e., the area used to contain products and quotients).

Operation. Field B is added to field A. The floating-point sum replaces field A; field B remains unchanged.

Floating Subtract

Macro-instruction

Line	Label	Operation	Operands & Remarks
010		FS	A,B

The A and B addresses refer to the units position of the exponents of the fields. Neither A nor B should reference any location within the Product Area.

Operation. Field B is subtracted from field A. The floating-point difference replaces field A; field B remains unchanged.

Floating Multiply

Macro-instruction

Line	Label	Operation	Operands & Remarks
010		FM	A,B

The A and B addresses refer to the units position of the exponents of the fields. Neither A nor B should reference any location within the Product Area.

Operation. Field A is multiplied by field B. The floating-point product replaces field A; field B remains unchanged.

Floating Divide

Macro-instruction

Line	Label	Operation	Operands & Remarks
010		FD	A,B

Operation. Field A is divided by field B. The floating-point quotient replaces field A; field B remains unchanged. Neither A nor B should reference any location within the Product Area.

Fixed-Point Divide

Macro-instruction

Line	Label	Operation	Operands & Remarks
010		DIV	A,B,A1,B1

In addition to the A and B operands, which represent the addresses of the dividend and divisor, the divide macro-instruction requires two additional operands; one specifies the number of zeros to be inserted to the right of the dividend (A1 operand) and the other, the shift factor needed by the subroutine (B1 operand). Specifically,

A operand is core storage address of dividend (must not reference Product Area).

B operand is core storage address of divisor (must not reference Product Area).

A1 operand is 00099 minus the number of zeros desired to the right of the units position of the dividend.

B1 operand is 00100 minus the length of the quotient. The quotient must be at least two digits in length.

NOTE: The quotient address after the division is executed will be equal to 00099 minus the length of the divisor.

Prior to the divide operation, the divide subroutine always resets to zeros (clears) positions 00080 through 00099, the product area where the 20-digit quotient and remainder are developed. For variable-length mantissa subroutines, the divide subroutine clears core storage positions 00001-00099 to zeros. When the quotient plus the remainder exceeds the number of positions cleared to zeros, positions lower than the last position cleared must be reset to zeros by *programming*. One additional position should also be cleared to allow for a possible overdraw. For example, if 25 positions are required for the quotient and remainder in a fixed-length mantissa subroutine, 00074-00079 should be reset to zeros before the divide macro-instruction is given.

The fixed-point divide macro-instruction may be used with any of the subroutine sets. Whenever it is used, the fixed-point divide subroutine will be incorporated into the object program at object time. For the subroutine sets that are designed to work with automatic divide, the fixed-point divide subroutine uses automatic divide in performing its operation. For the subroutine set that is designed to work *without* automatic divide, the fixed-point divide subroutine performs its operation as instructed by the routine without the aid of the Automatic Divide feature. Coding of the macro-instruction is the same for all of the subroutine sets.

Operation. The area to be cleared is automatically reset to zeros. The dividend (A address) is transmitted to the product area (A1 address), beginning at the low-order dividend digit and terminating at the flag bit marking the high-order position of the dividend field. The A1 address is 00099 minus the number of zero positions desired to the right of the dividend.

The algebraic sign of the dividend is automatically placed in location 00099, regardless of where the rightmost dividend digit is placed by the A1 address. A flag bit automatically marks the high-order digit of the dividend.

The divisor (B address) is successively subtracted from the dividend. The B1 address of the divide macro-instruction positions the divisor for the first subtraction from the high-order position(s) of the dividend, as in manual division. The B1 address is determined by subtracting the number of digits in the quotient from 100. For subroutines using program divide, the value of B1 must be between 0 and 99. For subroutines using automatic divide, the value of B1 is not restricted.

The quotient and remainder replace the dividend in the product area. The address of the quotient is 00099 minus the length of the divisor. The algebraic sign of the quotient (determined by the signs of the dividend and divisor) is automatically placed in the low-order position of the quotient. The address of the remainder is 00099; a flag bit is automatically placed in the high-order position of the remainder. The remainder has the sign of the dividend and the same number of digits as the divisor.

The high/positive indicator is on if the quotient is positive and not zero; the equal/zero indicator is on if the quotient is zero. Neither indicator is on if the quotient is negative.

The quotient must be at least two digits in length. This is the minimum field length that the 1620 will accept.

EXAMPLES

1. The macro-instruction

   DIV A, B, 99, 96

   will perform the division for $\bar{0}273\overline{)\bar{3}972}$ and store the result $\bar{0}014$ in storage locations 00092 through 00095.

2. The macro-instruction

   DIV A, B, 96, 93

   will perform the division for $\bar{0}273\overline{)\bar{3}972.000}$ and store the result $\bar{0}014.549$ in storage locations 00089 through 00095.

NOTE: In examples 1 and 2, A represents the address of the dividend $\bar{3}972$, and B represents the address of the divisor $\bar{0}273$.

Incorrect Positioning of Divisor. The following error conditions are caused by an incorrect B1 address.

1. An incorrectly positioned divisor can cause more than nine successful subtractions and an incorrect quotient. The divide operation is terminated, the Overflow indicator and Overflow Arithmetic Check light are turned on, but processing will not stop unless the Overflow Check switch is set to STOP. A divide by zero (K/0) causes the same error conditions just described.
2. The high-order digit of the dividend is assumed by the 1620 to be one position to the left of the high-order digit of the divisor. The high-order digits of the dividend are lost if the divisor is positioned too far to the right. Processing continues with no indication of an incorrect quotient.
3. If the B address is less than 10000, i.e., between 00100 and 00999, the divide operations will terminate when a subtraction occurs at 0XX99. This, in effect, restricts the size of the dividend to 10,020 digits if only 20,000 positions of core storage are installed.

Floating Shift Right

Macro-instruction

Line 3 5	Label 6 11	Operation 12 15	Operands & Remarks 16 20 25 30 35 40 45 50
0,1,0		FSRS	A,B

The effect of this macro-instruction is to shrink the mantissa by shifting it to the right and truncating the low-order digits. The A address is normally the units position of the mantissa.

$\bar{M}MMMMMMM\bar{E}E$

$\uparrow$ units position of mantissa

The B address specifies the digit of the mantissa which will become the low-order digit of the mantissa.

Operation. The field at the B address (the portion of the mantissa to be retained) is shifted right to the location specified by the A address. The exponent is not moved or altered. For example, the macro-instruction

FSRS 00097,00093

causes the mantissa

$\bar{3}0590011325\bar{7}\bar{0}1$

Storage Address	Storage Address
00093	00097

to be shifted, producing the following result

$0000\bar{3}059001\bar{1}\bar{0}1$

Storage Address	Storage Address
00093	00097

Vacated high-order positions are set to zeros, an existing flag at the A address is retained for algebraic sign, and the field flag bit is transmitted with the high-order digit of the B field.

Floating Shift Left

Macro-instruction

Line	Label	Operation	Operands & Remarks
010		FSLS	A,B

The effect of this macro-instruction is to expand the mantissa by shifting it to the left and filling the vacated positions with zeros. It is important to note that the B address is the low-order position of the field moved, and the A address is the high-order position of the resultant field.

Operation. The field at the B address, which is the *low-order* digit of the mantissa, is shifted left so that the *high-order* digit is moved to the location specified by the A address. The exponent is not moved or altered. For example, the macro-instruction:

FSLS 00090,00097

causes the mantissa

$00\bar{1}1325\bar{7}\bar{0}1$

Storage Address	Storage Address
00090	00097

to be shifted, producing the following result

$\bar{1}132570\bar{0}\bar{0}1$

Storage Address	Storage Address
00090	00097

An existing flag bit at the Q address is retained for algebraic sign; the field flag bit is transmitted with the high-order digit of the Q field.

Transmit Floating

Macro-instruction

Line	Label	Operation	Operands & Remarks
010		TFLS	A,B

The B address refers to the low-order digit of the floating-point field exponent, whereas the A address refers to the low-order position to which the field is transmitted.

Operation. The field at the B address is transmitted to the location specified by the A address. The B field remains unchanged in storage. Flag bits in the three low-order positions of the B field are also transmitted. Starting with the fourth low-order position, only one additional flag bit is transmitted, and it stops transmission.

Branch and Transmit Floating

Macro-instruction

Line	Label	Operation	Operands & Remarks
010		BTFS	A,B

The B address is normally the low-order position of the floating-point field exponent, whereas the A address is the leftmost position of the next instruction to be executed.

Operation. The address of the instruction following the macro-instruction is saved at a storage location in the BTFS subroutine, and the field at the B address is transmitted to the A address minus one. The normal exit of a routine which is entered by a BTFS is a Branch Back (BB) instruction. The instruction at the A address is the next one executed. The B field remains unchanged in core storage. Any flag bits in the three low-order positions of the B field are transmitted. Starting with the fourth low-order position, only one additional flag is transmitted, and it stops transmission.

Floating Square Root

Macro-instruction

Line	Label	Operation	Operands & Remarks
3 5	6 11	12 15	16 20 25 30 35 40 45 50
0 1 0		FSQR	A, B

The A and B addresses refer to the units position of the exponents of the fields.

Operation. The square root of argument B is extracted and the result, in floating point form, is stored at A. The argument, which must be in floating-point form, is unchanged by the operation.

The floating-point square root subroutine accepts all numbers within the floating-point range that are greater than or equal to zero. If the argument is less than zero, a 0 is placed in location 00401 and A is set equal to SQR |A|.

Floating Sine

Macro-instruction

Line	Label	Operation	Operands & Remarks
3 5	6 11	12 15	16 20 25 30 35 40 45 50
0 1 0		FSIN	A, B

The A and B addresses refer to the units position of the exponents of the fields.

Operation. The sine of argument B is computed and the result, in floating-point form, is stored at A. the argument must be in radians and in floating-point form. The computation does not disturb the original value of the argument.

The floating-point sine subroutine accepts all numbers of floating-point range up to and including exponent $\overline{0}8$ (fixed length mantissa) or L (variable length mantissa).

For arguments with exponents less than $\overline{0}3$, the magnitude of the maximum truncation error in the mantissa of the result does not exceed 10^{-L}. Accuracy in the mantissa of the result decreases as the size of the argument (exponent of $\overline{0}3$ or greater) increases. For error codes, see Table 5.

Floating Cosine

Macro-instruction

Line	Label	Operation	Operands & Remarks
3 5	6 11	12 15	16 20 25 30 35 40 45 50
0 1 0		FCOS	A, B

The A and B addresses refer to the units position of the exponents of the fields.

Operation. The cosine of argument B is computed and the result, in floating-point form, is stored at A. The argument must be in radians and in floating-point form. The computation does not disturb the original value of the argument.

The allowable range of the argument, maximum accuracy, etc., for the cosine subroutine is the same as that previously described for the sine subroutine.

Floating Arctangent

Macro-instruction

Line	Label	Operation	Operands & Remarks
3 5	6 11	12 15	16 20 25 30 35 40 45 50
0 1 0		FATN	A, B

The A and B addresses refer to the units position of the exponents of the fields.

Operation. The floating-point value of the arctangent of B is computed and the result is stored at A. The argument must be in floating-point form; the result in radians will also be in floating-point form.

The arctangent subroutine accepts any number within the floating-point range. During the evaluation of the arctangent of B, use will be made of the divide subroutine.

The maximum truncation error in the mantissa of the result is $\pm 10^{-L}$, except for results having an exponent less than or equal to $\overline{02}$ ($E \leq \overline{02}$). The maximum error for these results is $\pm$ 1 in the $(L+1)$th decimal place. L = 08 for the fixed length mantissa.

Floating Exponential (Natural)

Macro-instruction

Line	Label	Operation	Operands & Remarks
3 5	6 11	12 15	16 20 25 30 35 40 45 50
0 1 0		FEX	A, B

Operation. The A and B addresses refer to the units position of the exponents of the fields. The value of the e^B, where B is in floating-point form, is computed and the result, also in floating-point form, is stored at A. An input value that exceeds

227.955924206n ...... n($\bar{2}$27955924206n ...... n$\bar{0}$3)

causes an exponent overflow, and one which is less than

−227.955924206n ...... n($\bar{2}$27955924206n ...... $\overline{n0}$3)

causes an exponent underflow. Depending on the type of error, a 1 or $\bar{1}$ is placed in location 00401.

For negative arguments, the subroutine uses the absolute value of the argument to evaluate the function and then finds the reciprocal value.

For positive and negative arguments, the maximum truncation error in the mantissa of the result is $\pm 10^{-L}$.

Floating Exponential (Base 10)

Macro-instruction

Line	Label	Operation	Operands & Remarks
0,1,0		FEXT	A,B

The A and B addresses refer to the units position of the exponents of the fields.

Operation. The value of 10^B is in floating-point form; it is computed and the result, also in floating-point form, is stored at A. An input value that exceeds

98.9n . . . n($\bar{9}$89n . . . n$\bar{0}$2)

causes an exponent overflow, and one which is less than

−98.9n . . . n($\bar{9}$89n . . . $\overline{n0}$2)

causes an exponent underflow.

This subroutine handles negative arguments in the same manner as they are handled by the natural exponential subroutine. Maximum accuracy is the same.

Floating Logarithm (Natural)

Macro-instruction

Line	Label	Operation	Operands & Remarks
0,1,0		FLN	A,B

The A and B addresses refer to the units position of the exponents of the fields.

Operation. The floating-point value of the ln B is computed and stored at A. Input arguments must be in floating-point form.

This subroutine accepts all arguments greater than zero within the floating-point range. For error codes, see Table 5.

Floating Logarithm (Base 10)

Macro-instruction

Line	Label	Operation	Operands & Remarks
0,1,0		FLOG	A,B

The A and B addresses refer to the units position of the exponents of the fields.

Operation. The floating-point value of the log 10^B is computed and stored at A. Input arguments must be in floating-point form.

This subroutine accepts all arguments greater than zero within the floating-point range. For error codes see Table 5.

Adding Subroutines

The user may add from one to twelve subroutine macros to subroutine sets 01, 02, and 03. Each new subroutine may use from two to nine operands. Although the minimum number of operands allowed is two, both the A and B operands may be the same.

To add a subroutine, it is necessary to:

1. Modify the Op Code table to include the new mnemonic (see SPS II-D MODIFICATION PROGRAM).
2. Write the subroutine in SPS language, keeping in mind certain factors regarding PICK, mantissa length (L), and modifications with regard to the subroutine itself.
3. Assemble the subroutine in relocatable form, and store it on the disk.

Modifying the Op Code Table

The Op Code table is modified by executing the SPS II-D Modification Program. This program is part of the Monitor system; it is described later in the manual.

Writing a Subroutine

When writing a subroutine, the programmer should be aware of certain information concerning PICK, namely: the functions of PICK, PCK area, linkages, common work areas in PICK, and the means of signifying operands that are relative to PICK and/or are a function of mantissa length.

FUNCTIONS OF PICK

PICK is common to all subroutines in the subroutine set. Therefore, it is to the advantage of the subroutine writer to make use of PICK. The listing of the appropriate PICK subroutine (furnished with the Library package) should be studied. Briefly, PICK performs the following operations:

1. Resets location 00401 (subroutine error digit).
2. Moves A and B operands into PCK + 15 and PCK + 20, respectively. If more than two operands are used, all should be handled by the *user's* subroutine.
3. Calculates the return address to the mainline program. However, if a subroutine uses more than two operands, the return address must be calculated by the subroutine itself. To calculate the return address, use the following formula:

   (PCK +10) + 5n + 1 if n is even, or
   (PCK +10) + 5n + 2 if n is odd

   where PCK + 10 (before entering the PICK subroutine) is the address of the high-order digit of the first operand, and n is the number of operands.

   If PICK calculates the return address (two operands), eleven is added to PCK + 10.
4. Moves the B operand (mantissa and exponent) into working area beta. If the A operand is also used by the subroutine for calculation, it (the A operand) must be moved by the subroutine itself. This requires two instructions:

```
TF  ALPHA, PCK + 15, 11
TF  ALPHA —2, PCK + 25, 11
```

   or one instruction if TFL is available on the machine.

```
TFL  ALPHA, PICK + 15, 11
```

5. Sets error indicator 00401 for overflow and underflow.
6. Stores the computed floating-point result in the location specified by the A operand.

The functions of PICK are not mandatory, but are under control of the user. When a subroutine is called, the object program branches directly to the subroutine. If the functions of PICK are desired, the user may branch to PICK via the following linkage:

```
TFM    PCK + 5, * + 20
B7     PCK, , 6
```

OPERANDS THAT ARE FUNCTIONS OF PICK AND/OR MANTISSA LENGTH

Whenever PICK is used, the programmer must use instructions in his subroutine which make reference to the PICK subroutine (All references to PICK must be written as relocatable quantities.) The operands of these instructions are then adjusted to make them correspond to the actual addresses of PICK in the object program. This is done by using a pseudo constant (DC statement). The constant does not become a part of the object program; its only function is to indicate that the instructions that follow are to be modified.

One DC statement can modify up to 25 instructions. Each instruction, whether it is to be modified or not, requires two digits in the pseudo constant, one for the P operand and one for the Q operand. The statement itself consists of three operands: the first specifies the length of the constant which may not be greater than 50 nor less than 2; the second, the actual constant; the third, the storage address of the constant. This address must be specified as an absolute address of 00350. The P and Q operand modifier constants follow:

P and Q Operand Modifiers	Modification
0	No Modification
1	Add L
2	Subtract L
3	Add 2L
4	Subtract 2L
5	Modify with respect to PICK, no L modification
6	Modify with respect to PICK, add L
7	Modify with respect to PICK, subtract L
8	Modify with respect to PICK, add 2L
9	Modify with respect to PICK, subtract 2L

The following example shows how a variable-length mantissa subroutine may be modified by the use of modifier constants.

```
DC      6, 527005, 350
TF      SAVE, 98
SF      SAVE
TFL     PCK + 15, SAVE , 6
B7      PCK + 10, , 6
```

NOTE: (1) Intervening DORG statements and constants between instructions are never modified in this manner.
(2) SAVE is a relocatable quantity.

Assembling a Subroutine

When a subroutine is assembled, it must be "Assembled Relocatable," stored in relocatable format, and must have the following items defined:

1. Disk Identification Map (DIM) entry
2. Subroutine identification number
3. Desired entry points

Subroutine DIM Entry

A subroutine DIM entry is in the same general format as that described in the Supervisor section of this manual. However, the last two 5-digit fields specify the length of the subroutine and the subroutine identification number instead of a core address and a starting or entry address.

Subroutine DIM entries occupy fixed locations in the DIM table. There are 30 entries reserved for subroutine sets 01, 02, and 03 (this includes an entry for PICK in each set); only one entry is needed for set 00. The 30 entries are needed for the 17 subroutines (plus PICK) provided by IBM plus 12 subroutines that may be added by the user.

To calculate the DIM entry number for a new subroutine, add the new subroutine number (18-29) to the base DIM number for the applicable subroutine set. The base numbers for each set are as follows:

Base Number	Subroutine Set
130	Fixed Point Divide - 00
100	Fixed Length - 01
70	Variable Length - 02
40	Automatic Floating Point - 03

For example, the DIM entry number for the first user-written subroutine to be added to the variable length set would be $70 + 18 = 88$.

A DIM entry number is assigned to a subroutine by including an "ID NUMBER" control record when the subroutine is assembled (see SPS CONTROL RECORDS).

SUBROUTINE LENGTH

The length of a particular subroutine is automatically placed in its respective DIM entry when an LIBR control record (see SPS CONTROL RECORDS) precedes the subroutine source program.

SUBROUTINE IDENTIFICATION NUMBER

As stated previously, all subroutines are identified by a 5-digit code number. When a subroutine is being added, this number must be supplied by the user. The number is composed of two digits for "subroutine set number" (00, 01, 02, 03), two digits for "subroutine number" (01-29), and one digit for "number of entry points" (1-9). The number will automatically be loaded to its proper position in the DIM table if it is used as the address operand of the DEND statement that terminates the subroutine source program.

```
Example        DEND        02182
```

This code number identifies subroutine number 18, a "2-entry point subroutine" that is to be placed in the variable-length subroutine set.

SUBROUTINE ENTRY POINTS

Each subroutine requires at least one entry point but may have as many as nine entry points. These must be specified at the beginning of the user's source program. Two records of data are needed:

```
ORIGIN  DSA    ENTRY1, ENTRY2, etc.
        DORG   ORIGIN-4
```

In the preceding statements, ORIGIN is any label not otherwise used in the subroutine; ENTRY1 is the label of the first entry point; ENTRY2 is the label of the second entry point. The DORG is needed to ensure that the subroutine will begin at relocatable 00000.

SUMMARY OF ASSEMBLY PROCEDURE

By inserting a DSA for entry points and placing the subroutine identification number in the DEND statement of the source program, it is possible through the use of proper control records to assemble a subroutine and have it loaded to disk and ready to use all in one operation. Of course, the mnemonic of the added subroutine must be defined in the Op code table prior to assembly of a program that uses it.

Example

This example illustrates how a subroutine is added to the subroutine library of the Monitor I System. In this example, the new subroutine is assembled along with a mainline program and both are then executed (Figure 10). The example assumes that the Supervisor program is already in core storage.

GIVEN:

Macro – EXCH A, B

Function– to exchange floating-point numbers between A and B

Identity – Subroutine No. 18, Set No. 02

PROCEDURE:

1. Modify Op Code table by loading the following records;

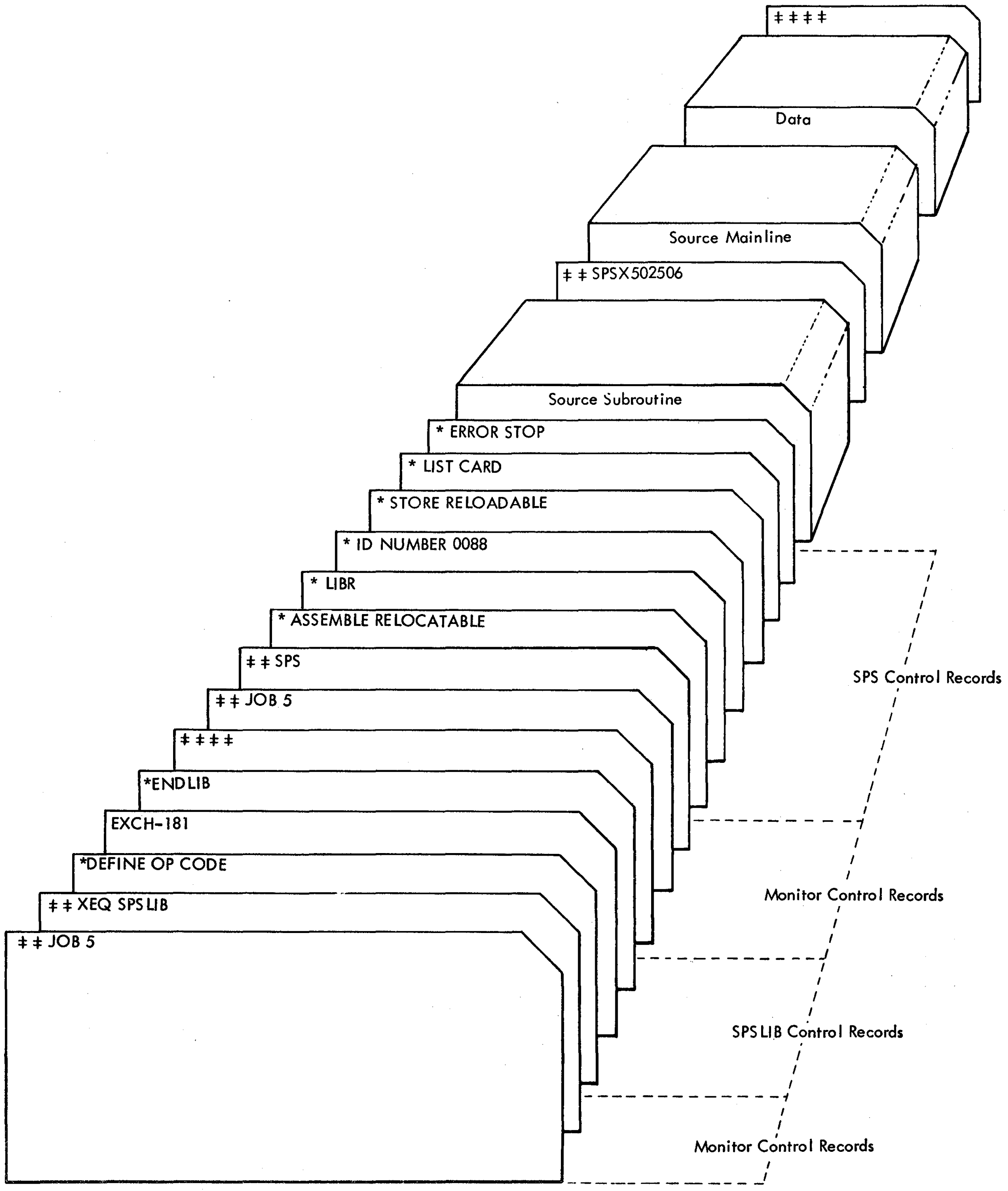


Figure 10. Stacked Input for "Adding SPS Subroutine" Example

Line	Label	Operation	Operands & Remarks
0,1,0		DSA	ENTRY
0,2,0	*		DSA IS USED TO DEFINE SUBROUTINE ENTRY POINT
0,3,0	ZERO	DORG	*-4 , DEFINES RELOCATABLE ZERO
0,4,0	BETA	DS	,ZERO+196 ,, DEFINES RELOCATABLE SYMBOL
0,5,0	PCK	DS	,2365 ,, DEFINES ABSOLUTE SYMBOL
0,6,0	ENTRY	TFM	PCK+5 ,*+20
0,7,0		B7	PCK , ,6, LINKAGE TO PICK
0,8,0		TF	PCK+20 ,PCK+15 ,611, MOVE A EXP TO B OPERAND EXP POSITN
0,9,0		TF	PCK+30 ,PCK+25 ,611, MOVE A CHAR TO B OPRAND CHAR POSITN
1,0,0		DC	4 ,0505 ,350, PSEUDO CONSTANT FOR PICK AND L REF
1,1,0		TF	PCK+15 ,BETA ,6, MOVE B TO A
1,2,0		TF	PCK+25 ,BETA-2 ,6, NOTE THAT PICK PUT B IN BETA
1,3,0		B7	PCK+10 , ,6, RETURN TO MAIN LINE
1,4,0		DEND	02 18 1 , ID FOR SUBR

Figure 11. Source Program for "Adding SPS Subroutine" Example

Monitor Control —	XEQ specifying SPSLIB
SPSLIB Control	DEFINE OP CODE
	EXCH — 181 (Col. 12-19)
	ENDLIB

NOTE: The above four records must be entered from the same input unit.

2. Precede the source subroutine (Figure 11) in the stacked input with the following control records:

Monitor Control —	JOB
	SPS
SPS Control —	ASSEMBLE RELOCATABLE
	LIBR
	ID NUMBER 0088
	STORE RELOADABLE
	LIST CARD
	ERROR STOP

I/O Macro-Statements

The I/O macro-statements are provided to relieve the programmer of the task of writing his own I/O subroutine in a source program each time he wants to read or punch cards or tape, read or write disk storage, or read or write typewriter. I/O macro-statements may be inserted anywhere in the user's program. When an I/O macro-statement is encountered in the source program during assembly, linkage instructions to the I/O routine (part of the Supervisor program) are generated in the object program.

The macro-instruction mnemonic operation codes and their meanings follow:

Mnemonic	Meaning
GET	Read card, paper tape, typewriter, or disk.
PUT	Punch card, paper tape, or write typewriter, or write disk records with or without read-back check.
SEEK	Seek disk record.
CALL	Read disk stored subprogram and execute (requires LINK operand), or read disk stored subprogram or data without execution (requires LOAD operand), or return control to Supervisor (requires EXIT operand).

The six types of macro-statements that use the I/O routine are written as follows:

	Label	Operation	Operands & Remarks
1.		GET	DEF
2.		PUT	DEF
		PUT	DEF,RBC
3.		SEEK	DEF
4.		CALL	LINK,DIMENT,RELOC
		CALL	LINK,NAME,RELOC
5.		CALL	LOAD,DIMENT,RELOC
		CALL	LOAD,NAME,RELOC
6.		CALL	EXIT

A label and remarks may be included with each of these statements. The address associated with the label will be that of the first instruction in the generated linkage. The operands RBC, LINK, LOAD, and EXIT are fixed symbols, i.e., they must be written exactly as given. The symbol DEF is the address of the associated I/O constant generated by an I/O declarative statement. DIMENT is the DIM entry number of a program to be called from disk storage; RELOC is the relocation core storage address of a program to be called; and NAME is the name of a program (as it appears in the Equivalence table) to be called. DEF, DIMENT, and RELOC may be in actual or symbolic form.

PUT statements may be specified with or without read-back check (RBC), as shown. Either a DIM entry number or the program name can be given with CALL LINK or CALL LOAD statements. Also, the relocation address is optional with these statements, i.e., the programmer does not have to include a relocation address.

A CALL EXIT statement should be included in the user's source program. This statement, when executed at the end of the object program, causes control to be returned to the Supervisor program.

All linkages for I/O routines are generated automatically through the use of macro-instructions. The data and addresses supplied in the macro-instruction are incorporated into the linkage instructions where they are made available for use by the I/O routine. The linkage instructions generated in an object program for macro-instructions by the processor are equivalent to the following series of symbolic instructions.

Linkage for GET, PUT, and SEEK

```
    TFM     IORT, * + 23
    B       ENTRY, DEF, 7
```

IORT is the address of a 5-position storage area in the I/O routine.

ENTRY is one of the four possible entry points in the I/O routine (see I/O ROUTINE LINKAGES).

DEF is the address of the I/O constant.

Linkages for CALL LINK and CALL LOAD

Without a relocation address

```
    TFM     IORT,   * + 19
    B       IOCAL
    DORG    * - 4
    DC      1, M0
    DSC     1, M1
    DSC     1, 0
    DC      5, IIII@
```

With a relocation address

```
    TFM     IORT,   * + 19
    B       IOCAL
    DORG    * - 4
    DC      1, M0
    DSC     1, M1
    DSC     1, 0
    DC      4, IIII
    DSA     LLLLL
    DSC     1, @
```

IOCAL is an entry to the I/O routine (core storage address 00716).

M_0 M_1 is a constant $\bar{3}2$ for call linkages.

IIII is the DIM entry number of the program to be called.

LLLLL is the relocation core storage address where the program is to be loaded.

Linkage for CALL EXIT

```
    B       MONCAL
    DORG    * - 4
```

MONCAL is an entry (core storage address 00796) to the I/O routine which will call in the Monitor Control Record Analyzer routine.

Input/Output Declarative Statements

Input/output core storage areas to be used by a program for reading or punching cards, reading or punching tape, reading or writing disk storage, and reading or writing typewriter must be allotted by the programmer. These I/O areas may be allotted and defined using DSS or DAS declarative statements. Each I/O area used by an I/O macro-statement (GET or PUT) must be identified by an I/O declarative statement.

The I/O declarative mnemonic operation codes and their meanings are as follows:

Code	Meaning
DTN	Define Typewriter Numerical
DTA	Define Typewriter Alphameric
DCN	Define Card Numerical
DCA	Define Card Alphameric
DPTN	Define Paper Tape Numerical
DPTA	Define Paper Tape Alphameric
DDW	Define Disk with WLRC
DD	Define Disk without WLRC

DTN, DTA, DCN, DCA, DPTN, and DPTA Statements

These statements may be used in the source program to identify numerical or alphameric typewriter, card, or paper tape I/O areas. Each statement causes an 8-digit I/O constant to be generated in the object program. Two operands are required for each statement. The first operand is used to specify the address where the I/O constant is to be loaded into core storage. The operand may be an absolute value or a symbolic name. If a symbolic name is used, the symbol must have been previously defined as an absolute value; that is, it must have appeared in the label field of a statement preceding the statement in which it is used. If the first operand is omitted, the processor assigns the address to which the constant will be loaded in core storage.

The second operand, which may be symbolic or actual, is the leftmost core storage address of an assigned I/O area. This address will be included in the constant in the object program. Remarks are permitted following the second operand. Items (operands or remarks) of a statement must be separated by commas. Address adjustment may be used with either operand.

If a label is included with the statement, the storage address assigned to it will be that of the leftmost position of the 8-position I/O constant.

The following example illustrates a DCN statement.

Line	Label	Operation	Operands & Remarks
3 5	6 11	12 15	16 20 25 30 35 40 45 50
010	CARDIO	DSS	80,,DEFINE CARD I/O AREA
020	DEF	DCN	,CARDIO,IDENTIFY CARD I/O AREA

The first statement defines an 80-position numerical card I/O area and the second statement identifies it. The I/O constant generated in the object program for the DCN statement is equivalent to the following symbolic instructions,

```
DEF     DS     , * + 1
        DSA    CARDIO
        DC     3, 0̄4@
```

where the label DEF is the leftmost address of this 8-position constant and CARDIO is the address of the associated I/O area. The code $\bar{0}4$ is generated for a DCN statement only. A complete list of the codes generated for each type of statement follows:

DTN	$\bar{0}0$	(Typewriter Numerical)
DTA	$\bar{0}6$	(Typewriter Alphameric)
DCN	$\bar{0}4$	(Card Numerical)
DCA	$\bar{1}0$	(Card Alphameric)
DPTN	$\bar{0}2$	(Paper Tape Numerical)
DPTA	$\bar{0}8$	(Paper Tape Alphameric)

DDW and DD Statements

These statements are used to specify the disk control field (defined by a DDA statement) to be used for a disk operation (SEEK, GET, or PUT) and to specify certain options. Each statement (DDW or DD) is assembled as an 8- or 13-digit I/O constant depending upon the number of given operands.

A minimum of two operands is required with a DDW or DD declarative statement. The first operand, which may be symbolic or actual, is used to specify the address where the generated I/O constant is to be loaded into core storage. If this operand is omitted, the processor will assign the address.

The second operand is the address of the leftmost position of the disk control field (defined by a DDA statement). The operand may be in actual or symbolic form.

The third operand, which is optional, specifies a relocation core storage address of a program or data to be read into core storage from disk storage (when the object program is executed). It may be in actual or symbolic form. If the programmer does not want to include a relocation address, and another operand or remark is to follow, a comma must be present.

The fourth operand, also optional, can cause the read/write heads to be repositioned to an assigned cylinder (specified in the System Communications Area) after a disk read or write instruction is executed as a result of a GET or PUT input/output macro-instruction. The operand letter "R" causes repositioning. If the operand is a blank or a letter "N," no repositioning takes place. If the programmer does not enter a letter N or R, and another operand or remark is to follow, a comma must be present.

The fifth operand, also optional, may be the letter A or a blank. If it is the letter A, the sector address in the disk control field will be used for the disk operation initiated by a SEEK, GET, or PUT macro-instruction. If the operand is blank, an effective sector address, produced by adding the sector address to the address of the beginning work cylinder, is used for the disk operation.

If this operand does not contain the letter A and remarks are to follow, the operand must be a comma.

Address adjustment may be used with any of the first three operands. If a label is given with a DDW or DD statement, the associated address in the assembled object program will be that of the leftmost position of the resulting I/O constant.

The following example illustrates a DDW statement.

Line	Label	Operation	Operands & Remarks
3 5	6 11	12 15	16 20 25 30 35 40 45 50
010	DEF	DDW	,DCTRL,,R,A,IDENTIFY DISK CTRL FLD

This declarative statement:

1. Identifies the address of the disk control field by the symbol DCTRL.
2. Indicates by the letter "R" that the read/write heads are to be repositioned to a previously assigned cylinder after a GET or PUT is executed.
3. Indicates by the letter "A" that the sector address in the disk control field is to be used for the disk operation.

The I/O constant generated in the object program for a DDW or DD statement is equivalent to the following symbolic instructions:

For statements *without* a relocation address (third operand)

```
DEF     DSC     1, M0
        DSC     1, M1
        DSA     DCTRL
        DC      1,@
```

For statements *with* a relocation address

```
        DSC     1, M0
        DSC     1, M1
        DSA     DCTRL
        DSA     RELOC
        DC      1, @
```

where M_0 and M_1 are codes generated by the processor for use by the I/O routine when the object program is executed. DCTRL is the address of the disk control field. RELOC is the relocation core storage address.

All disk I/O options are specified by the codes in positions M_0 M_1. Only certain options are utilized by the DD and DDW statements. A complete list of options available to the user by hand coding M_0 and M_1 are given under I/O CONSTANTS in the Supervisor Program section.

To define a disk control field for a seek, read, or write operation, a DDA statement may be used. The memory address in the DDA statement specifies the core location where reading or writing of data is to begin. Any DDA statement used by the I/O routine must be followed by a statement that defines a record mark (e.g., DC 1, @). Data or programs which occupy a read area are replaced by data read from disk storage. Provision should be made to save the replaced data or program, if necessary.

SPS II-D Processor

The 1620/1710 SPS II-D processor, available in either card or paper tape form, is designed to use the IBM 1311 Disk Storage Drive as an integral unit in the assembling of user-written source programs. By using the large storage capacity of the disk drive, program assembly can be accomplished quickly and efficiently.

A typical assembly procedure might proceed as follows:

1. The source program is loaded together with the applicable Monitor Control records and SPS II-D Control records (see SPS II-D CONTROL RECORDS).
2. The source input is read into core storage, one statement at a time. During assembly, the work cylinder area of disk storage is used for intermediate output.
3. If a program listing (typewriter) or a list deck (cards) has been requested, the output will appear after all statements have been read.
4. After assembly, the object program is stored or outputted as directed by the SPS II-D control statements that were loaded with the source input. Subroutines used by an object program and subprograms are not stored or outputted as part of the object program.

Operating Procedures

These operating procedures are written assuming that the SPS II-D processor and all subroutine sets are already on disk storage.

Assembly Set-Up

To assemble an SPS II-D source program, proceed as follows: Load the source program preceded by the applicable Monitor Control record (SPS or SPSX) and the desired SPS II-D control records. The source program must end with a DEND statement. Following the DEND statement is either the first card of the next "job," or data if the program is to be executed immediately after assembly.

The loading procedure for SPS programs is more fully described in the Supervisor section of this manual.

SPS II-D Control Records

SPS control records must be provided to control the assembly of SPS programs. These records may be in card, paper tape, or typewritten form, and are inserted in the stacked input behind the Monitor Control record (SPS or SPSX) to control the specified SPS assembly. SPS control records are typed out when they are encountered in the stacked input. The format of an SPS control record in terms of cards is as follows:

Columns	1	*
	2-75	Control statement

Only one control statement may be entered in each control card. The 24 control statements must be written exactly as given, except for blanks which are permitted anywhere in the control statement (TWO PASS MODE, OBJECT CORE n, etc.). Control statements may be followed by remarks. Any statement, other than those listed below (e.g., an identification statement) will be typed, but will have no effect on the assembly. The processor will indicate an identification statement by typing (ID) to the right of such a statement.

Intervening blanks between the letters of a control statement do not invalidate the statement.

TWO PASS MODE. This control statement causes the object program to be produced by entering the source program twice (two passes). Two passes are required when the space allotted for work storage is too small to contain the intermediate output from the assembly. (The space required for one-pass operation is approximately one sector per source statement.) If one-pass assembly is attempted with too large a source program, an error message is typed out.

OBJECT CORE n. This statement specifies the core storage capacity (20,000, 40,000, or 60,000) of the object machine (machine on which the object program will be run). If the storage capacity of the assembly machine (machine on which the object program will be assembled) and the object machine are the same, this statement is not needed. The n digit of the statement is one of the coded digits 2, 4, or 6 which represent 20,000, 40,000, and 60,000, respectively.

SUBROUTINE SET nn. This statement specifies a subroutine set number, 00, 01, 02, or 03. When the program being assembled is to be executed, this set number will be used to load the proper subroutines into core storage. This subroutine set specification can be overridden, however, by specifying a different number in the XEQS Monitor Control card (see SUPERVISOR PROGRAM) when the program is executed. If no subroutine set control statement is present at *assembly* time or at execute time, the assembler will use the set number that was previously stored in the System Communication Area (see section on Supervisor Program).

MANTISSA LENGTH nn. This statement specifies the mantissa length (02 to 45) for subroutines sets 02 or 03. The mantissa length that will ultimately be used for execution purposes is determined in the same manner as the subroutine set number described above.

NOISE DIGIT n. This statement specifies the noise digit (0-9) to be used by the subroutines. The noise digit that will ultimately be used for execution purposes is determined in the same manner as the subroutine set number described above.

ERROR STOP. This statement instructs the processor to stop whenever a source statement containing an error is encountered. When this occurs, an error message will be typed (see ERROR MESSAGES). The operator can then enter a corrected source statement and continue assembly (see ON-LINE ERROR CORRECTION). If an Error Stop control statement is omitted, the processor will not stop for erroneous source statements; however, an error message will still be typed.

ASSEMBLE RELOCATABLE. This statement causes the processor to assemble a relocatable object program in System Output format. If this statement is omitted, the processor will produce an "absolute," nonrelocatable program.

BEGIN CARD INPUT, BEGIN PAPER TAPE INPUT, BEGIN TYPEWRITER INPUT. These three statements cause the loading program to begin reading input from the newly designated unit. These statements can be used as "last" control statements when the source program is to be entered from a different input medium than were the control statements.

TYPE SYMBOL TABLE. This statement causes the symbol table to be typed after all source statements have been read (see TYPEOUT OF SYMBOL TABLE).

PUNCH SYMBOL TABLE. This statement causes the symbol table to be punched into cards after all source statements have been read. These cards may be listed 80-80 on an IBM 407 to obtain a printed symbol table.

LIST TYPEWRITER. This statement causes a program listing (containing both source and object data) to be typed as the program is being assembled.

LIST CARD. This statement causes the program to be punched into cards which may be used to make a program listing (IBM 407 80-80). If desired, both the *List Typewriter* and *List Card* statements may be used in one assembly.

OUTPUT CARD. This statement causes the object program to be punched into cards in a reloadable format (see SYSTEM OUTPUT FORMAT in the Supervisor section). This output will occur after any symbol table and listing outputs.

OUTPUT PAPER TAPE. This statement causes the object program to be punched into paper tape in a reloadable format (see SYSTEM OUTPUT FORMAT in the Supervisor section). Either an *Output Card* or an *Output Paper Tape* statement may be used for an assembly, but not both.

STORE CORE IMAGE. This statement causes an assembled program to be permanently stored on the disk in a format that is identical to the format of an executable program in core storage. Subroutines required by the program, however, remain in relocatable format until the program is executed. For a method of storing subroutines in core image format with the main program, while overcoming other limitations inherent in this statement, refer to the section entitled, CONVERTING SPS OBJECT PROGRAMS TO CORE IMAGE. The *Store Core Image* statement may not be used in an assembly that also contains an *Assemble Relocatable* statement.

STORE RELOADABLE. This statement causes the assembled program to be stored on the disk in a reloadable format. This format is identical to that described under SYSTEM OUTPUT FORMAT. If neither Store Core Image nor Store Reloadable is specified, the assembled program will not be permanently stored on the disk. However, the program will remain in the work cylinders until destroyed by another job.

SYSTEM SYMBOL TABLE. This statement allows the source program to use symbols stored in the System Symbol table without defining them in the source program itself. There is a provision in SPS II-D for defining user symbols in the System Symbol table (see SPS II-D MODIFICATION PROGRAM).

NO SUBROUTINES. This statement is used, when assembling subprograms for a mainline program, to prevent subroutines from being called with subprograms. When the mainline program is assembled, it must specify or call all the subroutines used by it, as well as those used by its subprograms.

ID NUMBER dddd. This statement assigns a 4-digit DIM entry number (dddd) to a program being assembled. Exactly four digits, including leading zeros, must be entered.

NAME aaaaaa. This statement can be used to assign a Name in the Equivalence table for an assembled program which is to be stored in disk storage. aaaaaa is a 6-character alphameric name. At least one of these characters must be alphabetic.

LIBR. This statement must be entered when assembling a user-written subroutine that is to be added to the library subroutines.

PUNCH RESEQUENCED SOURCE DECK. This statement causes the processor to punch a new source deck in sequence by page and line number. The page and line field will contain a 5-digit number starting with 00010 and will increase by ten for each successive card, e.g., 00010, 00020, etc. The resequenced deck is punched while the old source cards are being read. The output appears in the punch stacker ahead of any other punched output. When operating in two-pass mode, the resequenced source deck should be used for the second pass. Corrections to source statements made from the typewriter will not appear in a resequenced source deck.

Error Messages

The error message codes that might be typed out on the typewriter during an assembly are listed in Table 6. Error messages take the following general form:

```
PPPPP     ALABEL + CCCC  ERn
```

where PPPPP is the page and the line number of the statement in error, ALABEL is the last label used, and CCCC is the number of statements from that label to the statement in error.

When ER5 (see Table 6) is typed out, the erroneous symbol is also typed.

Table 7 shows what the processor will do about each error if no Error Stop control statement has been included in the assembly.

On-Line Error Correction

One-Pass Mode. If the operator wishes to correct source program errors during the assembly process,

Table 6. Description of SPS Error Codes

ERROR CODE	CAUSE OF ERROR
ER1	The capacity of the machine on which the object program is to be executed has been exceeded. The processor does not take subroutines into account when determining this error.
ER2	Invalid label or record mark is in a label field.
ER3	Invalid OP code or record mark is in an OP code field.
ER4	A label is defined more than once.
ER5	1. A symbolic address contains more than six characters. 2. An actual address contains more than five digits. 3. An undefined symbolic address is used in an operand. 4. A HEAD character ($) is improperly specified.
ER6	A DSA statement has more than ten operands.
ER7	A DSB statement has the second operand missing.
ER8	1. A DC, DSC, or DAC has a specified length greater than 50. 2. A DVLC has a length greater than 50. 3. A DMES has a length greater than 100. 4. A DNB has a length greater than 99.
ER9	A DC, DSC, DAC, DVLC, or DMES statement has no constant specified.
ER10	1. A DC or DSC statement has a specified length which is less than the number of digits in the constant itself. 2. A DAC statement has a specified length which is less than or greater than the number of digits in the constant itself.
ER11	An invalid character is used as a HEAD character in a HEAD statement.
ER12	A HEAD operand contains more than one character.
ER13	A DMES statement contains an invalid starting mode character.
ER14	1. A DMES statement contains a control character which is incorrectly specified. 2. A DMES statement has an invalid format, i.e., stray parenthesis, etc.
ER15	A DMES statement contains an alpha character in a numerical field.
ER16	A DMES statement contains an invalid mode change.
ER17	1. A relocatable assembly contains either a relocation error (see Rules of Relocatability) or, 2. A DORG with an absolute operand.
ER18	A symbolic name used in a CALL LINK or CALL LOAD statement is not in the Equivalence table.
ER19	The storage area allotted for the symbol table has been exceeded.
ER20	Intermediate output has exceeded disk storage work area (program requires two passes).
ER21	Object output has exceeded disk storage work area.
ER22	Improper "select" operand is in a CALL statement; i.e., neither LINK, LOAD, nor EXIT is specified.

Table 7. Disposition of SPS Errors When no Error Stop Statement is used

NOTE: Assembly and outputting continues in all cases except ER 19, 20, and 21.

ERROR CODE	DISPOSITION
ER1	No disposition.
ER2	The label is ignored.
ER3	A NOP is assembled.
ER4	The second definition of the label is ignored; the first definition of the label is used in the assembly.
ER5	The operand is assembled as an absolute 00000.
ER6	The first ten operands are assembled; any remaining operands are ignored.
ER7	The number of elements is set to 1.
ER8	1. Length is set to 50. 2. Length is set to 50. 3. Length is set to 100. 4. Length is set to 99.
ER9	A field of zeros is generated, equal to the size of the length operand for the DC, DSC, DAC, or DVLC constant. In the case of a DMES, an end of message (ǂ ǂ) is assembled and the address counter is increased by 100.
ER10	For a DC or DSC, the length of the constant is used as the length operand; for a DAC, the specified length is used, and the programmer-assigned address, if present, is ignored.
ER11	The HEAD character is set to blank.
ER12	The first character of the operand is used as the HEAD character.
ER13	The starting mode is assembled as the alphabetic mode.
ER14	An end of message (ǂ ǂ) is inserted into the constant.
ER15	An end of message (ǂ ǂ) is inserted into the constant.
ER16	A $\bar{0}$ is placed in the next available location following the mode change.
ER17	1. The operand is assembled as an absolute 00000. 2. The DORG is ignored.
ER18	A DIM number of 0000 is assembled.
ER19	Processing continues but no more labels are stored. After completion of the intermediate phase, processing stops, the following message is typed, and control returns to the Supervisor Program. DISK AREA TOO SMALL. ASSEMBLY DELETED
ER20	Processing continues, but no more intermediate data is sent to disk storage. After completion of the intermediate phase, processing stops, the following message is typed, and control returns to the Supervisor Program. DISK AREA TOO SMALL. ASSEMBLY DELETED
ER21	Processing stops immediately and control is returned to the Supervisor Program.
ER22	The statement is ignored.

he must use the Error Stop control statement. When an error occurs, the appropriate error message is typed out along with one of the following instructions to the operator:

RE-ENTER STATEMENT or
RE-ENTER OPERANDS

At this point, the processor returns the typewriter carriage and types the full erroneous source statement. If only the operands are to be re-entered, the processor will then retype the source statement up to the operand field. The processor at this point requires that the operator enter either an entire corrected source statement or a corrected operand field. The operator should use the previously typed original statement as a guide to the positions of the Page, Line, Label, Op code, and Operand fields.

Two-Pass Mode. The error correction procedure in two-pass mode is identical with that of the one-pass mode, with one exception. During the second pass, the processor might type an error message containing "ER XX." This message always refers to a statement corrected during the first pass. The operator should scan the typewritten record of the corrections made during the first pass to find the one identical in page and line number, label, and increment. When the processor types RE-ENTER STMT and returns the carriage, the operator must re-enter the entire corrected statement, exactly duplicating the statement entered during the first pass.

Post Assembly Phase

After assembly is completed and listings, if desired, have been outputted, the following messages are typed:

END OF ASSEMBLY
XXXXX CORE POSITIONS REQUIRED
XXXXX STATEMENTS PROCESSED

In the above typeouts, XXXXX is a 5-digit number. In the case of CORE POSITIONS REQUIRED, any needed subroutines are included in the count of core positions.

SYMBOL TABLE OUTPUT

If either of the statements *Type Symbol Table* or *Punch Symbol Table* are present in an assembly, the symbol table will be typed or punched during assembly. This output, if punched, will precede the list deck in the punch stacker.

All 6-character labels are listed first in reverse alphameric order, i.e., 9 to 0, Z to A. All other labels follow in normal alphameric order with their head characters. In the case of an assembly in which the number of symbols exceeds 235 (some symbols will then have to be stored on disk), the listing is broken into two or more blocks, each of which is sorted as described above.

The format of the symbol table output is as follows:

Typewriter. The typed output lists all labels and their numerical equivalences, five to a line. The format is as shown.

Label	Equivalence
LLLLLL	AAAAA(−)

Here LLLLLL refers to a 6-character label or a 5 or fewer character label with a head character.

AAAAA refers to the numerical equivalence of the symbol. The minus sign, if present, denotes a negative quantity. If the program is being "assembled relocatable," the minus sign is replaced by an R to denote a relocatable quantity.

Card. The card output format of the symbol table is as follows:

Columns	1-13	1st label plus equivalence
	17-29	2nd label plus equivalence
	33-45	3rd label plus equivalence
	49-61	4th label plus equivalence
	65-77	5th label plus equivalence

FORMATS OF TYPEWRITER LISTING AND PUNCHED DECK

If desired, the operator can obtain a typewriter listing and/or a punched list deck of an assembled program. The formats of each type of output are described here.

Typewriter. A typewriter listing consists of a source statement together with its associated assembled machine language instruction.

Card. A card list deck usually consists of one card for each source statement. The format is as follows:

Columns	1-5	Page and line number.
	6	Blank.
	7-12	Label as on source card.
	13	Blank.
	14-17	Op mnemonic as on source card.

18	Blank.
19-78	Operand fields as on source card. If the fields extend beyond column 59, the object information (normally found in columns 61-80 of first card) is placed on a subsequent card or cards.
61-65	Actual address of assembled instruction or constant.
66	Blank.

NOTE: The data in columns 67-80 is peculiar to the type of statement assembled.

Imperative Statements.

Columns	67-68	Op code in machine language.
	69	Blank.
	70-74	P operand in machine language.
	75	Blank.
	76-80	Q operand in machine language.

Non-imperative Statements.

Columns	67-71	Length of assembled data.
	72	Blank.
	73-80	If these columns are punched, they will contain actual assembled data.

ERROR MESSAGES AFTER ASSEMBLY

The following error messages are applicable after assembly.

EXCEEDED SPECIFIED CAPACITY BY XXXXX

The above message indicates that the object program together with applicable subroutines would exceed the available core storage if the program were to be executed. The available core storage is determind by the user's *Object Core* control card.

This error does not invalidate the assembly, however, since a different set of subroutines may be specified at execution time. A different subroutine set might occupy less core storage; therefore, the error may no longer apply.

SUBROUTINES OTHER THAN PGM DIV USED

The above message is typed when subroutine set 00 (programmed divide) is specified and the mainline source program calls a subroutine in another set. This error does not invalidate the assembly since the user may specify a different subroutine set at execution time.

NO DIM ENTRY FOR SUBROUTINE

The above message is typed out when the DIM entry which corresponds to a called subroutine cannot be found in the map. This would indicate that the entry was either deliberately deleted from the map or otherwise destroyed. The assembly will continue, however, and the object program will be stored. But if execution was planned immediately after assembly (SPSX card), assembly would be deferred and control would return to the Supervisor program.

MORE THAN 5 CYLINDERS OF
RELOADABLE OUTPUT SSW4
ON TO DUMP OUTPUT OFF
TO CONTINUE, NO OUTPUT

The above message is typed when the reloadable object output would occupy more than 999 sectors on disk storage (approximately 5 cylinders). This situation is an error because programs greater than 999 sectors cannot be specified in the Disk Identification Map.

After the message is typed out, the computer halts. At this time the user can either turn Program Switch 4 on and depress START to have the program outputted on a pre-chosen output unit, or turn Program Switch 4 off and depress START to continue, in which case the program is not outputted. In either case the program is not stored on disk storage.

Execution of SPS II-D Object Programs

When SPS II-D object programs are to be executed, they are read into core storage from disk storage, paper tape, or cards by the use of Monitor Control records. The subroutines which are called for in the program are loaded from disk storage at execution time. Neither subroutines nor subprograms are ever a part of the mainline object program. They are stored on disk in relocatable form and brought into core storage if needed. The selection of proper subroutines at execution time is made by referring to an "indicator record" which is stored with the mainline object program. This record, generated at assembly time, contains a 1-digit location for each of the subroutines in a set. At assembly time, as the individual subroutine macro-instructions are encountered in the source program, a 1 is placed in the 1-digit location that corresponds to the subroutine being called. The record is then a "map" of the subroutines needed for the particular mainline object program.

ment. Thus, an address assigned by the programmer, e.g., DC 1, @, 19999, would not be included unless it fell within the addresses assigned by the processor.

- No TRA-TCD sequences are allowed.
- A program which requires subroutines and is assembled with an *ASSEMBLE RELOCATABLE control record cannot be converted to core image using the methods mentioned above.

A procedure which will overcome all these limitations is outlined below.

1. End the source program with a sequence of instructions which will dump the program into the work cylinders when it is called for execution. When the program is called the first time, only the special instructions will be executed. An assembly of this type is shown in the following example:

```
##JOB
##SPS
*NAME TEMP
*STORE RELOADABLE
*Other control records – listings, symbol table,
  etc.
START XX etc.
.
.
.
.
.
.
.
end of regular source program
STORE  PUT     X, RBC
       CALL    EXIT
X      DD      , XDDA
XDDA   DDA     , 1,00000, SSS, CCCCC
       DC      1, @
       DEND    STORE
```

   In the special sequence of instructions, sss is a sector count sufficient to store the entire core image program with applicable subroutines. The transfer into the work cylinders will start from core address ccccc. If subroutines are used, ccccc should be 02276 so as to include the subroutine transfer vector area.
2. Assemble the program.
3. Call the program as if it were to be executed. This will cause the special sequence of instructions to be executed, thereby dumping the program and subroutines into the work cylinders in core image format.
4. Using the disk utility routine, DLOAD or DREPL, load the program from the work cylinders into permanent disk storage. If DLOAD is used, give the program a different name and delete the old name, so as not to duplicate names in the DIM table. The parameters for the DLOAD or DREPL operation can be obtained either from the listing or from the messages typed out after the assembly; e.g., XXXXX CORE POSITIONS REQUIRED gives the highest processor-assigned address, including subroutines. An example of how to implement steps 3 and 4 is shown below:

```
##JOB
##XEQS NAME
##JOB
##DUP
*DLOAD or *DREPL
##DUP        }
*DELET NAME  }not needed if DREPL is used
```

   To call and execute a core image program which has been converted by this method, use an XEQ control record.

SPS II-D Modification Program

This program allows the user to modify the SPS II-D assembler by: (1) Adding or deleting operation codes from the System Op code table, and (2) Adding or deleting symbols from the System Symbol table. The SPS II-D Modification program is loaded into disk storage as part of the Monitor I system. It is identified in the Equivalence table by the name "SPSLIB."

Error Messages at Execution Time

The following error messages are applicable at execution time. Their occurrence terminates loading and returns control to the Supervisor program.

```
CORE CAPACITY EXCEEDED
BY XXXXX LOCATIONS
PROGRAM IS TERMINATED
```

The above message is typed out when the total core storage required for the object program and all applicable subroutines exceeds the available core storage. Note that there is a similar message at assembly time if available core storage is exceeded. However, it is possible to get the message at execution time without having gotten it at assembly time. This could happen if a different subroutine set is specified at execution time (in XEQS card) than that which was assembled with the source program.

```
SUBR NOT LOCATED IN SUBROUTINE MAP
```

The above message occurs if a subroutine that is specified in the "indicator record" of the object program cannot be found in the subroutine section of the Disk Identification Map.

```
IMPROPER IND CODE IN SUBR XXXX
```

The above message occurs when an invalid "reloadable indicator code" (see SYSTEM OUTPUT FORMAT in Supervisor section) is found in the object output of a subroutine. In this message, xxxx are the first four digits of the subroutine identification number; two digits are for the set number, and two digits are for the subroutine number.

Rules of Relocatability

When a program is relocated, as specified by an *Assemble Relocatable* statement, certain addresses within the program are adjusted relative to the relocation (starting) address. Only relocatable quantities are adjusted. Absolute quantities are not adjusted. Examples of both relocatable and absolute quantities follow:

$$\text{B}\qquad \overbrace{* + 24}^{\text{relocatable}}$$

$$\text{AM}\quad \text{X, } \overbrace{12345}^{\text{absolute}}$$

The processor recognizes relocatable and absolute quantities by applying the following rules:

1. An integer (e.g., 1, 12345, etc.) is an absolute value.
2. A processor-assigned address, which is associated with a label (i.e., the address of an instruction or constant with an associated label), is a relocatable quantity. An asterisk address (*) is also relocatable.
3. A symbol defined as equal to some quantity has the same relocation property as the associated quantity. An example follows:

   ```
   SYMBOL   DS      ,QUAN
   ```

4. The product of two absolute quantities is an absolute quantity.
5. The sum or difference of two absolute quantities is an absolute quantity.
6. The sum or difference of a relocatable quantity and an absolute quantity is a relocatable quantity.
7. The difference between two relocatable quantities is an absolute quantity.

The processor will recognize any of the following situations as "relocation errors."

1. The sum of two relocatable quantities.
2. The product of a relocatable quantity and any other quantity.
3. An operand below the relocatable address of 00000. For example, RELOC −10000, where RELOC is a relocatable quantity of less than 10000.

   NOTE: The exact negative of a valid relocatable quantity is a valid relocatable quantity.

Although the quantity defined by an operand may be either positive or negative, a symbol may be equivalent to a positive quantity only. If a symbol is defined equal to a negative quantity, any reference to that symbol by the assembler will produce the absolute value of the quantity.

Converting SPS Object Programs to Core Image

Two methods of storing object programs in core image have been described in this manual. One is by using the disk utility routines DLOAD and DREPL (see DISK UTILITY PROGRAM), and the other is by using a STORE CORE IMAGE control card when the program is assembled. Both of these methods have the following limitations:

- Subroutines cannot be converted and stored with with the main program.
- The core storage limits of the program image do not extend beyond the last *processor-assigned* state-

An XEQ Monitor Control record with the assigned name SPSLIB punched in columns 7-12 is used to call the modification program for execution. To specify the type of modification desired, the user places modification control records following the XEQ record. These records and any other input data to the Modification program must be entered from the same input device that was used to enter the XEQ record.

Modification program control records, in terms of cards, use the same format as that used for SPS control records. The five modification control statements must be written exactly as given (DEFINE OP CODE, DELETE OP CODE, DEFINE SYSTEM SYMBOL TABLE, LIST OP CODE, ENDLIB. Only one statement may be included in a control record. These statements are typed when they are read. A description of the five control statements follows.

DEFINE OP CODE. This statement causes user-assigned Op (operation) codes, specified in Op code definitions cards, to be added to the SPS II-D System Op code table. The Op code definition card(s) must follow the control record in the stacked input. The format of the Op code definition card follows:

Columns 12-15	New mnemonic Op code (left justified).
16-75	A 3-digit code which determines the instruction generated by the Op code. (The code may be preceded by a minus sign.)

The allowable 3-digit codes that may be entered in columns 16-75 are shown in Table 8.

The digits X and Y may be any number 0-9. A separate Op code definition card should be entered for each Op code that is to be defined. If an attempt is made to define an Op code that is already present in the Op code table, the message

ALREADY DEFINED

will be typed and the new Op code will be ignored. If space is unavailable in the Op code table for a new Op code, the message

NO ROOM IN TABLE

will be typed and the new Op code will be ignored.

DELETE OP CODE. This statement causes Op codes, specified in Op definition cards, to be deleted from the SPS II-D System Op code table. The Op code definition card(s), which must follow the control record in the stacked input, specifies in columns 12-15 the code to be deleted; columns 16-75 may be blank. Only one Op code may be specified per card. If an attempt is made to delete an Op code that is not in the Op code table, the message

NOT IN TABLE

will be typed and no change will be made to the table.

DEFINE SYSTEM SYMBOL TABLE. This statement is used to modify the System Symbol table. The System Symbol table consists of certain symbols that were defined when the Monitor System was assembled plus any symbols the user adds by means of the *Define System Symbol Table* statement. Any symbol that is in the System Symbol table may be used in any assembly without defining the symbol within the program being assembled. When used, the Define System Symbol statement first causes all user-defined symbols to be deleted from the table. Then all symbols which follow the Define System Symbol statement are added to the System Symbol table. Symbols to be added are defined in the Symbol Definition record. The format of this record in terms of cards is as follows:

Columns 6-11	Symbol to be defined (left justified).
16-75	An operand, symbolic or actual, but not asterisk. (If a symbolic operand is used, it must have been previously defined in the System Symbol table.)

If a symbolic operand, contained in the operand field (columns 16-75) of a Symbol Definition card, cannot be matched with a previously defined symbol in the System Symbol table, the message

UNDEFINED SYMBOL XXXXX

is typed out, where xxxxx is the undefined symbolic operand; no change is made to the System Symbol table. Up to 150 user-defined symbols may be added to the System Symbol table. Any attempt to insert more symbols causes an error message to be typed and control to be returned to the Supervisor program, thus terminating the add-to-symbol-table function. Symbols that have less than six characters will be defined with a blank "heading character" in the System Symbol table. Symbols defined as *positive* quantities will be treated as positive-absolute quantities in both absolute and relocatable assemblies. *Negative* quantities will be treated as negative-absolute quantities in an absolute assembly and positive-relocatable quantities in a relocatable assembly.

LIST OP CODE. This statement causes the processor to type a listing of the Op code table. All Op codes are listed in tabular form with their associated 3-digit codes.

ENDLIB. This statement causes control to be returned to the Supervisor program. In the stacked input, it must follow other control statements which utilize the modification program.

IBM Defined System Symbols

The following symbols will be available to the user in the System Symbol table.

Symbol	Equivalence	Description
9RCYL0 9RCYL1 9RCYL2 9RCYL3	00513 00515 00517 00519	These are the low-order positions of four 2-digit fields which contain the numbers of cylinders (00-99), where the disk access arm is repositioned *after* a disk operation in which a reposition has been requested. The four fields refer to drives 0, 1, 2, and 3, respectively.
9CCYL0 9CCYL1 9CCYL2 9CCYL3	02111 02113 02115 02117	These are the low-order positions of four 2-digit fields, similar to the previous four. However, these positions contain the cylinder numbers of the *current* access arm positions (the position of the arm after the last disk IORT operation).

Table 8. Codes and Assembled Data for SPS Modification Program

CODE	ASSEMBLED DATA	USE
XY0	XY PPPPP QQQQQ	Any instruction
-XY1	Macro-instruction subroutine linkage to subroutine XY	Subroutine Macros
-XY2*	8X PPPPP YQQQQ	SIOC instructions
-XY4	4X PPPPP Q00QY	Mask and Unmask instructions
XY2	3X PPPPP Q07QY	Disk instructions
XY3	3X PPPPP Q0YQQ	I/O instructions
XY4	34 PPPPP Q0XQY	Control instructions
XY6	46 PPPPP QXYQQ	Branch Indicator instructions
XY7	47 PPPPP QXYQQ	Branch No Indicator instructions

*If the first character of the Op code contained in columns 12-15 is the letter "R," position O_1 of the assembled instruction will be flagged.

FORTRAN II-D

The FORTRAN II-D programming system consists of the FORTRAN II-D *language* and the *processor*. The FORTRAN II-D language is comprised of a number of types of statements that the programmer may use in defining the problem to be solved. The FORTRAN II-D processor is a program that accepts the source program statements as input and produces, as output, a machine language program, known as the object program. The FORTRAN II-D processor can operate only under control of the Monitor System and the object programs it produces can be reloaded only by the Monitor Input/Output routine. It is possible for the user to remove the FORTRAN portion of Monitor I and still utilize the remainder of the system. It is also possible to remove some of the FORTRAN library subroutines that are supplied and still utilize the remainder of the FORTRAN system (see DISK STORAGE LOCATION OF THE FORTRAN COMPILER).

FORTRAN II-D Language

The FORTRAN II-D source program consists of a number of *statements*. Each statement deals with one aspect of the problem; that is, it may cause data to be fed into the computer, calculations to be performed, decisions to be made, results to be printed, etc.

Some statements do not cause specific computer action, but rather provide information to the processor program.

FORTRAN II-D statements are arranged in five groups:
Arithmetic statements which specify the mathematical calculations to be performed.
Control statements which govern the sequence in which the statements will be followed.
Subprogram statements that enable the programmer to define and use subprograms.
Input/Output statements that read data into the program or print or punch the results of the program.
Specification statements that provide information about the data that the object program is to process.

The above statement types are explained in detail later in this manual.

FORTRAN II-D statements are written on a standard FORTRAN Coding Form which is designed to organize the statements into the special format required by the processor program. All statements and comments of the source program are written on this form.

The function of each portion of the coding form shown in Figure 12 is explained below.

Space is provided at the top of each page for the name of the program, date, etc. This information does not constitute part of the source program and is not punched into cards.

The series of numbers (1, 5, 6, 7, 10, ..., 72) across the top of the form indicates the card column that the information is punched into.

Comments to explain the program are written in columns 2-72 of a line with a C in column 1. A comment line is not processed by the FORTRAN II-D program but is listed when the source program cards are listed.

Columns 2 through 5 are used for the *statement number*. Any number from 1 through 9999 may be used as a statement number. Statement numbers are used for cross reference within a program (see explanations of DO and GO TO statements) or may be used merely as a means of identifying statements. Statements should be numbered only when they are referenced by another statement and no two statements can have the same number. Also, there is no requirement that every statement must have a number, nor that statements must be numbered in sequence.

Column 6 of the initial line of a statement must be blank or zero. If a statement is too long to be written on one line it can be continued on as many as four "continuation lines." Continuation lines are written by placing in column 6 any character or any number from 1 through 9 (zero allowed only for initial line). The normal method is to number the initial line zero, the second line one (first continuation line), the third line two, etc. A statement other than a comment statement may not consist of more than 330 characters (i.e., 5 lines).

The body of the statements themselves are written in columns 7 through 72. Blank columns for the most part are ignored by the processor and may be used freely to improve the readability of the source program listing.

Columns 73 through 80 are not processed and therefore may contain any identifying information.

The information on each written line in the statement section of a coding form is punched into a card.

A standard FORTRAN card is shown in Figure 13.

After the cards are punched, they should be verified to lessen the chances of clerical errors causing source and object program errors.

Arithmetic Mode

Quantities used in the FORTRAN statements may be expressed in either *fixed-point* or *floating-point* form. Numbers expressed as integers (whole numbers) are considered fixed-point. Thus, the integers 3, 57, and 1008 are *fixed-point* numbers.

Floating-point arithmetic is a technique used to eliminate the complex programming required for correct placement of the decimal point in arithmetic operations. Floating-point numbers are represented in a standard format which specifies the location of the decimal point. With this method, quantities which range from minute fractions to large numbers may be handled by the computer. Floating-point numbers are expressed as decimal fractions times a power of ten. For example:

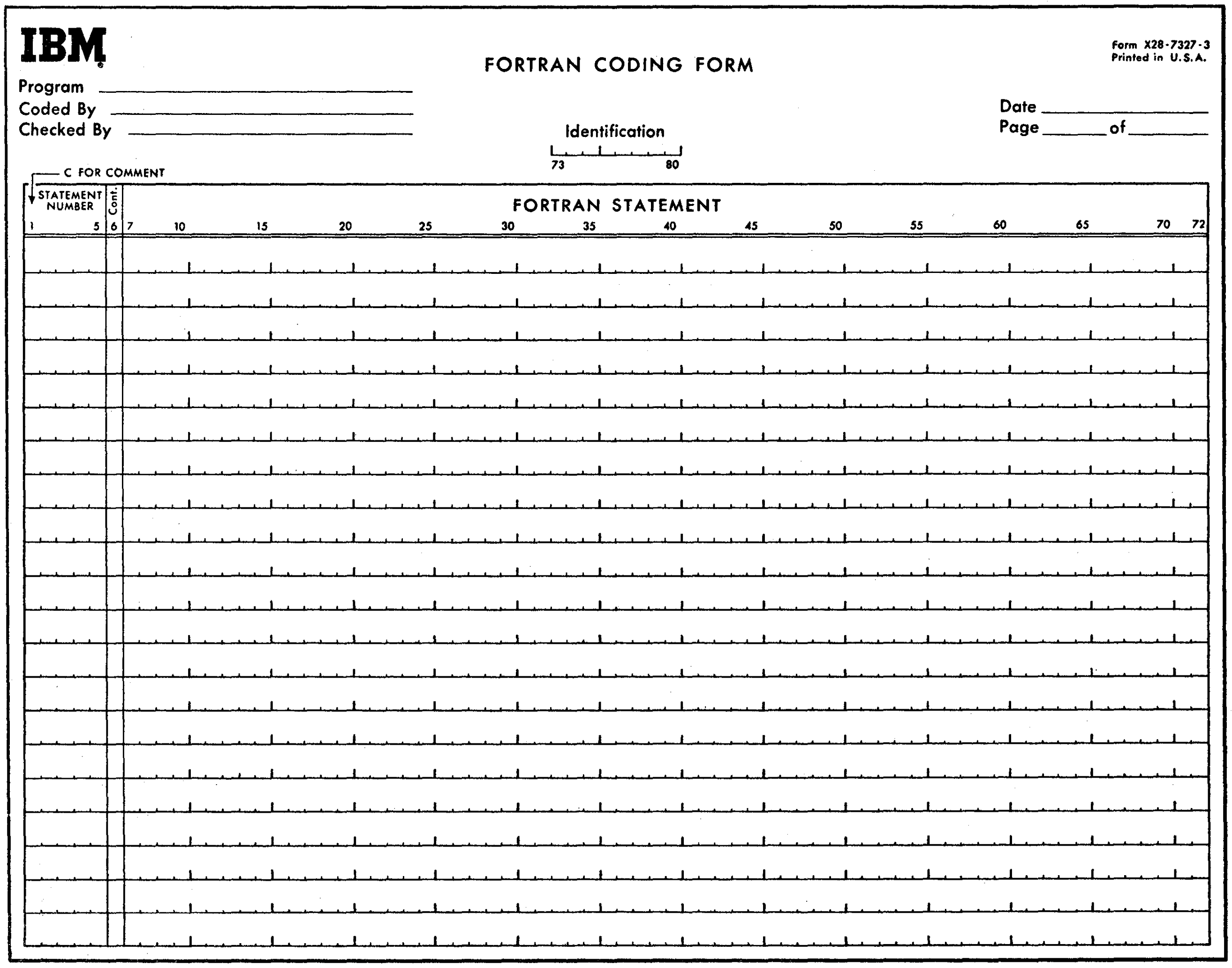
3.14159 is expressed as $.314159 \times 10^1$

4800.0 is expressed as $.48 \times 10^4$

0.0187 is expressed as $.187 \times 10^{-1}$

The numerical part of the floating-point number is called the *mantissa* and the power of ten is called the *exponent*. For a floating-point number, the decimal is always moved to the left of the high-order nonzero digit. This is called normalizing the number.

In FORTRAN II-D, fixed-point or floating-point numbers can be used, subject to the rules described under ARITHMETIC STATEMENTS.



IBM

FORTRAN CODING FORM

Form X28-7327-3
Printed in U.S.A.

Program ____________
Coded By ____________
Checked By ____________

Identification
73 80

Date ____________
Page ______ of ______

C FOR COMMENT

STATEMENT NUMBER 1 5 | Cont. 6 | FORTRAN STATEMENT 7 10 15 20 25 30 35 40 45 50 55 60 65 70 72

Figure 12. FORTRAN Coding Form

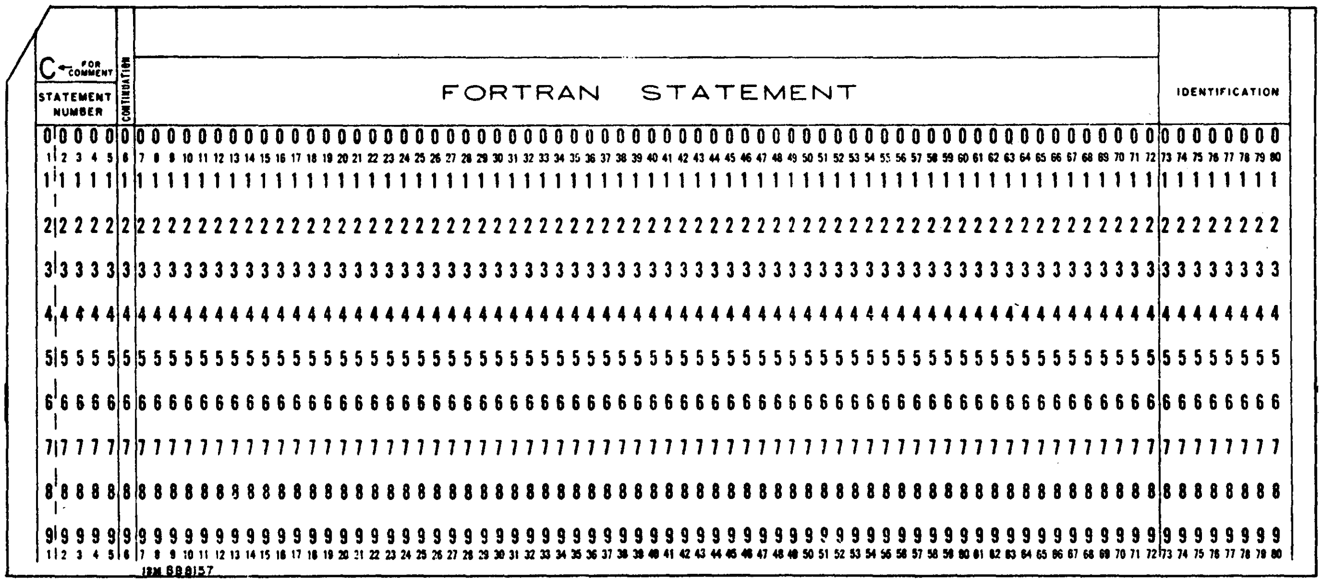


Figure 13. FORTRAN Source Program Card

Constants, Variables, Subscripts, and Expressions

Mathematical problems usually contain some data that does not change throughout the entire problem, and other data that may change many times during calculation. These two kinds of data are referred to as *constants* and *variables*, respectively. Both constants and variables can be used in FORTRAN II-D, but they must be written so the processor can distinguish one from the other.

Constants

A constant is any number which is used in computation without change from one execution of the program to the next. A constant appears in numerical form in the source statement. For example, in the statement

$$J = 3 + K$$

3 is a constant, since it appears in actual numerical form. Two types of constants may be written in FORTRAN II-D: fixed-point (restricted to integers), and floating-point (characterized by being written with a decimal point).

FIXED-POINT CONSTANT

A fixed-point constant is an integer consisting of 1 to 10 numerical characters (see ARITHMETIC PRECISION). A preceding plus sign is optional for positive numbers. An unsigned constant is assumed to be positive.

EXAMPLES

```
3
+1
-28987
```

FLOATING-POINT CONSTANT

A floating-point constant may be in either of two forms:

1. Any number consisting of 1 to 28 decimal digits with a decimal point at the beginning, at the end, or between two digits (see ARITHMETIC PRECISION). A preceding plus sign is optional for positive numbers. Zeros to the left of the decimal point are permissible.

   EXAMPLES

   ```
   17.
    5.0
   -.0003
    0.0
   ```

2. An integral decimal exponent preceded by an E may follow a floating-point constant. The magnitude thus expressed must be between the limits of 10^{-100} and 10^{99} or must be zero (see ARITHMETIC PRECISION).

   EXAMPLES

5.0E3	$= (5.0 \times 10^3)$
5.0E+3	$= (5.0 \times 10^3)$
3.14E	$= (3.14 \times 10^0)$

Variables

A FORTRAN variable is a symbolic name which will assume a value during execution of a program. This value may change either for different executions of the program or at different times within the program.

For example, in the statement

$$A = 3.0 + B$$

both A and B are variables. The value of B will be assigned by a preceding statement and may change from time to time, and A will change whenever this computation is performed with a new value of B.

As with constants, a variable may be in fixed-point or floating-point form.

FIXED-POINT VARIABLES

A fixed-point variable is named by using 1 to 6 alphabetic or numerical characters (not special characters) of which the first must be I, J, K, L, M, or N.

EXAMPLES

```
I
M2
JOBNO1
```

A fixed-point variable can assume any integral value provided the magnitude is less than the maximum size as defined through the use of a control record as stated under ARITHMETIC PRECISION. (If not defined, the maximum size will be 4 decimal positions for fixed-point numbers.)

FLOATING-POINT VARIABLES

A floating-point variable is named by using 1 to 6 alphabetic or numerical characters (not special characters), of which the first is alphabetic but *not* I, J, K, L, M, or N.

EXAMPLES

```
A
B7
DELTA
```

A floating-point variable may assume any value expressible as a normalized floating-point number, i.e., zero or any number between 10^{-100} and 10^{99}. The number of mantissa characters may be from 2 to 28 (see ARITHMETIC PRECISION). If not defined, the maximum size will be 8 characters for the mantissa.

Arithmetic Precision

The precision of the quantities used in the calculation is an important consideration in most types of scientific computation. For example, the computation of 7.19 x 3.14 would not be as precise as 7.19286 x 3.14159.

In the FORTRAN II-D system, the variable-field length capacity of the 1620 is used to allow varying the degree of precision from one program to another. The user has the ability to define the precision to which fixed-point and floating-point values should be carried. Floating-point precision, denoted in this publication as *f*, may be varied from 2 to 28 places; fixed-point precision, denoted by *k*, may be varied from 4 to 10 places.

The precision of the values may be changed by the use of a control record which must precede the source program (see FORTRAN II-D CONTROL RECORDS, FANDK) or the values may be changed by use of the Disk Utility Program DEFINE control record. *If not redefined (by either of these methods), the value of f is 8 and k is 4.*

Values for *f* and *k* must be the same for subprograms and link programs called by the main program.

Subscripts

An *array* is a group of quantities. It is often advantageous to be able to refer to this group by one name and to refer to each individual quantity in this group in terms of its place within the group. For example, assume the following is an array named NEXT:

```
15
12
18
42
19
```

If it were desired to refer to the second quantity in the group, the ordinary mathematical notation would be $NEXT_2$. In FORTRAN this becomes

NEXT (2)

The quantity in parentheses is called a subscript. Thus

NEXT (2) has the value of 12
NEXT (4) has the value of 42

The ordinary mathematical notation might be $NEXT_i$, to represent any element of the array NEXT. In FORTRAN, this is written

NEXT (I)

where I equals 1, 2, 3, 4, or 5. A program may also use two or three-dimensional arrays. For example, the following is a two-dimensional array named MRATE.

	Column 1	Column 2	Column 3
Row 1	14	12	8
Row 2	48	88	4
Row 3	29	25	17
Row 4	1	3	43

To refer to the quantity in Row 4, Column 2, the FORTRAN statement would be written as MRATE (4,2).

The value of MRATE (4, 2) is 3.

The value of MRATE (3, 3) is 17.

Thus, subscripts are positive fixed-point quantities whose values determine the member of the array to which reference is made.

GENERAL FORM

Let v represent any fixed-point variable and c (or c′) any fixed-point constant. Then, a subscript is an expression in one of the following forms.

```
v, c, v+c, v−c, c*v,
c*v+c′ or c*v−c′
```

(The symbol * denotes multiplication.)

EXAMPLES

```
  I
  3
MU + 2
5 * J
5 * J −2
```

The variable in a subscript must *not* itself be subscripted.

Subscripted Variables

A fixed or floating-point variable may be subscripted by enclosing up to three fixed-point subscripts in parentheses to the right of the variable.

EXAMPLES

```
A(I)
K(3)
BETA (5*J−2, K+2, L)
```

The commas separating the subscripts are required punctuation. Note that subscript arithmetic may take place as shown in the third example above. For instance, if J is equal to 20, the first subscript will be 98. (The symbol * denotes multiplication.)

The value of a subscript (including the added or subtracted constant, if any) must be greater than zero and not greater than the corresponding array dimension. Each subscripted variable must have the size of its array (i.e., the maximum values which its subscripts can attain) specified in a DIMENSION statement preceding the first appearance of the variable in the source program.

Expressions

An expression in FORTRAN language is any sequence of constants, variables (subscripted or not subscripted), and functions (explained later), separated by operation symbols, commas, and parentheses, which comply with the rules for constructing expressions. *Expressions appear on the right-hand side of arithmetic statements.*

In arithmetic-type operations, the following operation symbols are used:

+	addition
−	subtraction
*	multiplication
/	division
**	exponentiation (i.e., raising to a power)

RULES FOR CONSTRUCTING EXPRESSIONS

Since constants, variables, and subscripted variables may be fixed-point or floating-point quantities, expressions may contain either fixed-point or floating-point quantities; however, the two types may appear in the same expression only in certain ways. (In the following description, no mention is made of the rules for using fixed-point and floating-point quantities in *functions.* These rules will be stated when functions are discussed and will be considered as addenda to the following rules.)

1. The simplest expression consists of a single constant, variable, or subscripted variable. If the quantity is an integer quantity, the expression is said to be in the fixed-point mode. If the quantity is a floating-point quantity, the expression is said to be in the floating-point mode.

EXAMPLES

Expression	Type of Quantity	Mode of Expression
3	Fixed-Point constant	Fixed Point
3.0	Floating-Point constant	Floating Point
I	Fixed-Point variable	Fixed Point
A	Floating-Point variable	Floating Point
I(J)	Fixed-Point subscripted variable	Fixed Point
A(J)	Floating-Point subscripted variable	Floating Point

In the last example, note that the subscript, which must be a fixed-point quantity, does not affect the mode of the expression. *The mode of the expression is determined solely by the mode of the quantity itself.*

2. Exponentiation of a quantity does not affect the mode of the quantity; however, a fixed-point quantity may not be given a floating-point exponent. The following are valid:

I**J	Fixed Point
A**I	Floating Point
A**B	Floating Point

The following is *not* valid:

I**A	(Violates the rule that a fixed-point quantity must not have a floating-point exponent)

NOTE: The expression A**B**C is not permitted. It must be written A**(B**C) or (A**B)**C, whichever is intended.

3. Quantities may be preceded by a + or a – or connected by any of the operators (+, –, *, /, **) to form expressions, provided:
   a. No two operators appear consecutively.
   b. Quantities so connected are all of the same mode. (Exception: floating-point quantities may have fixed-point exponents.)

   The following are valid:

   –A+B
   B+C–D
   I/J
   K*L

   The following are *not* valid expressions:

A+–B	(must be written as A+(–B)
A+I	(variables are of different modes)
3J	(must be written as 3 * J if multiplication is intended)

4. The use of parentheses in forming expressions does not affect the mode of the expression. Thus, A, (A), and (((A))) are all floating-point expressions.
5. Parentheses may be used to specify the order of operations in an expression. Where parentheses are omitted, the order is taken to be from *left to right* as follows:

Order	Symbol	Operation
1	**	Exponentiation
2	* and /	Multiplication and Division
3	+ and –	Addition and Subtraction

For example, the expression

A+B*C/D+E**F–G

will be taken to mean

$$A+\frac{B*C}{D} + E^F - G$$

Using parentheses, the expression could be written

(A+B)*C/D+E**F–G

which would be taken to mean

$$\frac{(A+B)*C}{D} + E^F - G$$

A valid expression will be evaluated when the object program is executed. An invalid expression may result in an error message from the FORTRAN II-D processor or may result in inaccurate object program results.

Arithmetic Statements

GENERAL FORM

A = B

where A is a *variable* (subscripted or not subscript ed) and B represents an *expression.*

EXAMPLES

Q = K+1
A(I) = 2(I) + SINF (C(I))

The numerical calculations to be performed in the object program are defined by arithmetic statements. FORTRAN arithmetic statements closely resemble conventional arithmetic formulas. They contain a variable to be computed, followed by an equal (=) sign, followed by an arithmetic expression. In FORTRAN language, the equals sign means *"is to be replaced by"* rather than "is equivalent to." For example, the arithmetic statement

Y = N – LIMIT (J-2)

means that the value in the storage area assigned to Y is to be replaced by the value of N–LIMIT (J-2). The equal sign description can be emphasized more with the example of

I=I+1

which means that the variable I *is to be replaced* with its old value plus one.

The result of the expression is stored in fixed-point form if the variable on the left of the equals sign is a fixed-point variable, or in floating-point form if it is a floating-point variable.

If the variable on the left is in fixed-point form and the expression on the right is in floating-point form, the result is first computed in floating-point, then truncated (the fractional value is dropped) and converted to a fixed-point number. Thus, if the result of an expression is 3.872, the fixed-point number stored is 3, not 4. Likewise, the statement

$$J = A/B$$

where the value of A is 7,
and the value of B is 4,
produces a result of 1.

If the variable on the left is in floating-point form and the expression on the right is in fixed-point form, the expression will be computed in fixed-point and then converted to floating-point before it is stored as the new value of the variable.

Example	Meaning
A = B	Store the value of B in A.
I = B	Truncate B to an integer, convert to fixed point, and store in I.
A = I	Convert I to floating-point, and store in A.
A = 3.0*B	Replace A with 3 times B.
A = I*B	Not permitted. The expression is *mixed,* i.e., contains both fixed-point and floating-point variables.
A = 3 * B	Not permitted. The expression is *mixed.*

Control Statements

The second class of FORTRAN II statements is comprised of control statements that enable the programmer to state the flow of the program. Normally, statements may be thought of as being executed sequentially. That is, after one statement has been executed, the statement immediately following is executed. However, it is often undesirable to proceed in this manner. The following descriptions discuss statements which may be used to alter the sequence of a program.

GO TO Statement (Unconditional)

This statement interrupts the sequential execution of statements, and specifies the number of the next statement to be performed.

GENERAL FORM

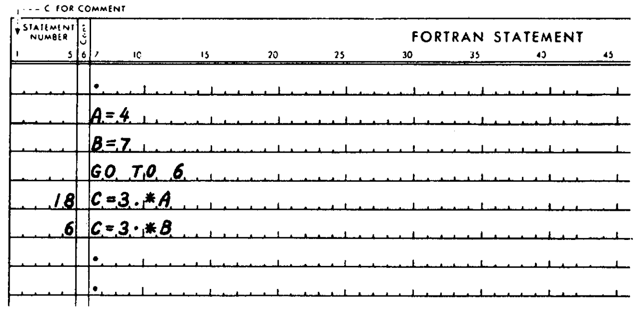
GO TO n

where n is a statement number.

EXAMPLES

GO TO 1009
GO TO 3

A coding sample is shown below:


C FOR COMMENT
STATEMENT NUMBER
FORTRAN STATEMENT

```
      .
      A=4
      B=7
      GO TO 6
   18 C=3.*A
    6 C=3.*B
      .
      .
```

The GO TO statement transfers the program to statement 6 where the result 21 is obtained.

Computed GO TO

This statement also indicates the statement that is to be executed next. However, the statement number that the program is transferred to can be altered during the program.

GENERAL FORM

GO TO $(n_1, n_2, \ldots, n_m)$, i

where $n_1, n_2 \ldots, n_m$ are statement numbers and i is a non-subscripted fixed-point variable.

The parentheses enclosing the statement numbers, the commas separating the statement numbers, and the comma following the right parenthesis are all required punctuation.

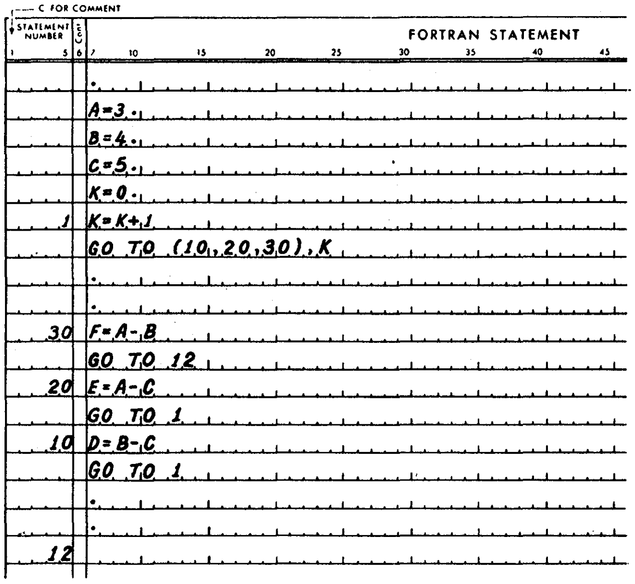
This command causes transfer of control to the 1st, 2nd, 3rd, etc., statement in the list depending on whether the value of i is 1, 2, 3, etc.

The variable i must never have a value greater than the number of items in the list.

EXAMPLES

	Meaning
GO TO (3, 4, 5), L	If L is 1, transfer to statement 3. If L is 2, transfer to statement 4. If L is 3, transfer to statement 5.
GO TO (4, 4, 5, 2), J	If J is *1 or 2,* transfer to statement 4. If J is 3, transfer to statement 5. If J is 4, transfer to statement 2.

Further examples of the Computed GO TO and the Unconditional GO TO statements are illustrated below:



```
C FOR COMMENT
STATEMENT NUMBER | FORTRAN STATEMENT
      .
      A=3.
      B=4.
      C=5.
      K=0.
    1 K=K+1
      GO TO (10,20,30),K
      .
      .
   30 F=A-B
      GO TO 12
   20 E=A-C
      GO TO 1
   10 D=B-C
      GO TO 1
      .
      .
   12
```

In the example, D, E, F are computed in that order, and the program is transferred to statement 12. This is a simplified example and if these were the only computations in the program, the programmer would simply list the arithmetic statements to compute D, E, and F in any desired order without using the Computed GO TO statement.

IF Statement

This statement permits the programmer to change the sequence of the statement execution, depending upon the value of the arithmetic expression.

GENERAL FORM

IF $(a)n_1, n_2, n_3$

where a is an expression and n_1, n_2, n_3 are statement numbers.

The expression must be enclosed in parentheses and the statement numbers must be separated by commas. The expression may be in either fixed or floating mode.

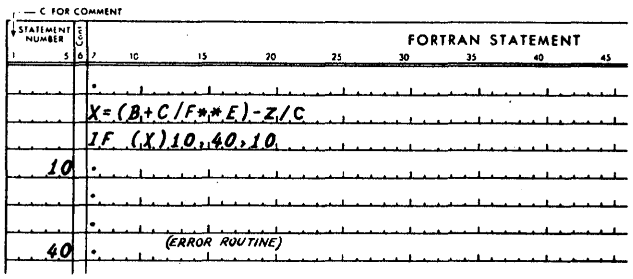
Control is transferred to statement number n_1, n_2, n_3 depending on whether the value of a is *less than, equal to, or greater than zero,* respectively.

EXAMPLE

IF (A − B) 10, 5, 7

which means "If the value of A minus B is less than zero, transfer to statement 10. If the value of A minus B is equal to zero, transfer to statement 5. If . . . A minus B is greater than zero, transfer to statement 7."

Suppose a value, X, is being computed. Whenever this value is negative or positive, it is desired to proceed with the program. Whenever the value is zero, an error routine is to be followed. This may be coded as:



```
C FOR COMMENT
STATEMENT NUMBER | FORTRAN STATEMENT
      .
      X=(B+C/F**E)-Z/C
      IF (X)10,40,10
   10 .
      .
      .
   40 .     (ERROR ROUTINE)
```

IF (SENSE SWITCH) Statement

This statement permits the program to transfer to a particular statement depending on the setting of any one of the four Console Program Switches.

GENERAL FORM

IF (SENSE SWITCH i) n_1, n_2

where i is the number of one of the Console Program Switches, and n_1, n_2 are statement numbers.

The parentheses enclosing the words SENSE SWITCH, and the commas separating the statement numbers are required punctuation.

The program transfers to the statement number n_1 when the designated Program switch is on, or to the statement numbered n_2 when it is off.

EXAMPLE

```
IF (SENSE SWITCH 3) 14, 10
```

which means, "If Sense Switch 3 is on, transfer to statement 14, otherwise transfer to statement 10."

DO Statement

GENERAL FORM

$$\text{DO n i} = m_1, m_2$$

or

$$\text{DO n i} = m_1, m_2, m_3$$

where n is a statement number, i is a non-subscripted fixed-point variable, and m_1, m_2, and m_3 are either an unsigned fixed-point constant or a non-subscripted fixed-point variable. *If m_3 is not stated, it is understood to be 1.*

EXAMPLES

```
DO 30 J = 1, 10
DO 30 J = 1, K, 3
```

The DO statement is a command to repeatedly execcuate the statements that follow, up to and including the statement with statement number n ., i.e., it forms a program loop.

The statements are executed with i = m_1 the first time. For each succeeding execution, i is increased by m_3. After the statements have been executed with i equal to m_2 (or as near as possible *without exceeding* m_2), control passes to the statement following the last statement in the range of the DO.

DO Range

The range of a DO is that set of statements which are executed repeatedly; i.e., it is the sequence of consecutive statements immediately following the DO, up to and including the statement numbered *n*.

DO Index

The index of a DO statement is the fixed-point variable i, which is controlled by the DO in such a way that its value begins at m_1, and is increased each time by m_3, up to, but not including the value which exceeds m_2. Throughout the range, the i-value is available for computation, either as an ordinary fixed-point variable or as the variable of a subscript. After the last execution of the range, the DO is said to be "satisfied."

Suppose for example, that control has reached statement 10 of the program:

```
.
.
10 DO 11 I = 1, 10
11 A (I) = I * N (I)
12...
.
.
```

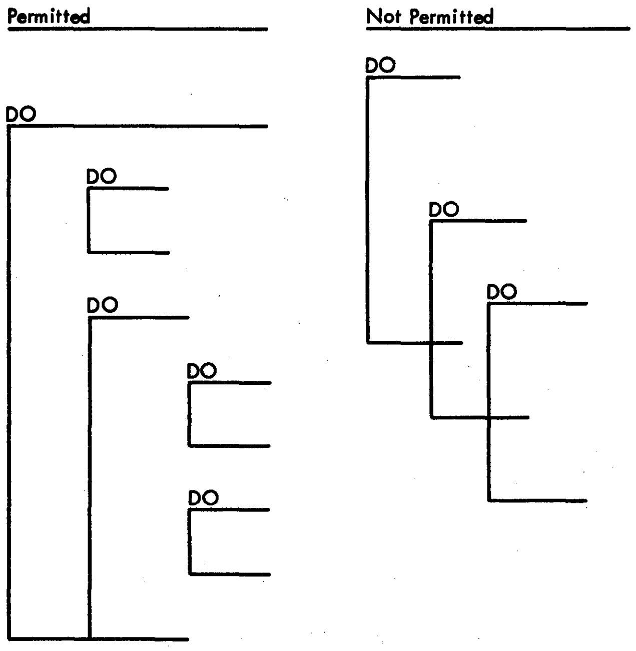
The range of the DO is statement 11, and the index is I. The DO sets I to 1 and control passes into the range. The value of 1 * N(1) is computed, converted to floating-point and stored in location A(1). Since statement 11 is the last statement in the range of the DO and the DO is unsatisfied, I is increased to 2 and control returns to the beginning of the range, statement 11. The value of 2 * N(2) is then computed and stored in location A(2). The process continues until statement 11 has been executed with I = 10. Since the DO is satisfied ($m_1 = m_2$), control then passes to statement 12.

DO's Within DO's

There may be other DO statements among the statements in the range of a DO. When this is so, the following rule must be observed.

If the range of a DO includes one or more other DO's, then all of the statements in the range of the latter must also be in the range of the former.

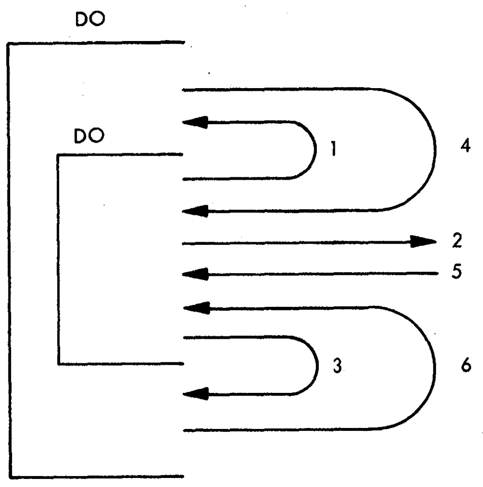
A set of DO's satisfying this rule is called a "nest of DO's." This rule is illustrated in the drawing below. (Brackets are used to illustrate the range of a DO).



Transfer of Control and DO's

Transfers of control from and into the range of a DO are subject to the following rule:

No transfer is permitted into the range of any DO from outside its range. Thus, 1, 2, and 3 are allowable transfers in the drawing below, but 4, 5, and 6 are not.



Preservation of Index Values. When control leaves the range of a DO in the ordinary way (i.e., when the DO becomes satisfied and control passes on to the next statement after the range) the exit is said to be a normal exit. After a normal exit from a DO occurs, the value of the index controlled by that DO is not defined, and the index cannot be used again until it is redefined. However, if the exit occurs by virtue of a transfer out of the range, the current value of the index remains available for any subsequent use. If the exit occurs because of a transfer which is in the ranges of several DO's, the current values of all the indexes controlled by those DO's are preserved for any subsequent use.

Exits. When a CALL statement (see CALL STATEMENT) is executed in the range of a DO, care must be taken that the called subprogram does not alter the DO index or indexing parameters. This applies as well when a FORTRAN function is called for in the range of a DO.

Restrictions on Statements in the Range of a DO. A statement which redefines the value of the index or of any of the indexing parameters (m's) is the only type of statement not permitted in the range of a DO. In other words, the indexing of a DO loop must be completely set before the range is entered. The first statement in the range of a DO must not be a non-executable statement, such as END, CONTINUE, and FORMAT statements. Also a DO loop cannot end with a transfer statement.

CONTINUE Statement

CONTINUE is a dummy statement which results in no instructions in the object program. It is most frequently used as the last statement in the range of a DO to provide a transfer address for IF and GO TO statements that are intended to begin another repetition of the DO loop.

EXAMPLE

CONTINUE

As an example of a program which requires a CONTINUE, consider the table search:

```
.
.
.
10 DO 12 I = 1, 100
   IF (ARG-VALUE (I) ) 12, 20, 12
12 CONTINUE
13 ...
.
.
```

This program causes a scan of the 100-entry VALUE table until it finds an entry that equals the value of the variable ARG, whereupon it exits to statement 20 with the value of I available for fixed point use; if no entry in the table equals the value of ARG, a normal exit occurs to the statement (13) following the CONTINUE.

PAUSE Statement

GENERAL FORM

PAUSE or PAUSE n

where n is an unsigned fixed-point constant.

EXAMPLES

PAUSE
PAUSE 33333

This statement halts the machine. Depressing the Start key causes the program to resume execution of the object program with the next statement. In a PAUSE *n* statement, where *n* is a 5-digit number within the range of valid 1620 addresses, the *n* can be displayed on the 1620 console in OR-2.

CALL EXIT Statement

This statement is used at the end of a FORTRAN program to return control to the Monitor Control Record Analyzer routine.

EXAMPLE

```
        CALL EXIT
```

STOP Statement

GENERAL FORM

```
      STOP or STOP n
```

where n is an unsigned fixed-point constant.

EXAMPLES

```
      STOP
      STOP 33333
```

When the object program is executed, the machine types STOP on the console typewriter, halts, and *n* can be displayed as it is for the PAUSE *n* statement. Depressing the Start key causes control to be returned to the Monitor Control Record Analyzer routine.

END Statement

GENERAL FORM

END or END (I_1, I_2, I_3, I_4, I_5)

where I is 0, 1, or 2.

EXAMPLES

```
      END
      END (1, 2, 0, 1, 1)
```

This statement differs from the previous control statements in that it does not affect the flow of control in the object program being compiled. Its application is to the FORTRAN II-D processor during compilation. An END statement will generate a halt and branch (to the Monitor Control Record Analyzer routine) in the object program. The statement END (I_1, I_2, I_3, I_4, I_5) is acceptable; however, the I's specified are meaningless in 1620 FORTRAN II-D.

The END statement must be the last statement (physically) of the source program.

Input/Output Statements

Input statements are used to read data into core storage and output statements are used to print or punch or store data. The READ, ACCEPT, ACCEPT TAPE, PUNCH, PUNCH TAPE, PRINT, and TYPE statements require the use of the FORMAT statement which is described under the section entitled SPECIFICATION STATEMENTS.

In addition to the statements listed above, the RECORD and FETCH statements also must include an ordered *list* of the quantities to be transmitted (see SPECIFYING LISTS OF QUANTITIES).

All FORTRAN II-D Input/Output statements cause the object program to make use of the Supervisor I/O routine (see section entitled I/O ROUTINES under SUPERVISOR PROGRAM).

Specifying Lists of Quantities

The input/output statements that call for transmission of data must include an ordered list of the quantities to be transmitted. The listed order must be the same as the order in which the words of information exist (for input), or the desired order for the output.

The formation and meaning of a list is best described by an example. Assume that the value of K has been previously defined.

```
     A, B(3), (C(I), D(I, K), I = 1, 10)
     ((E(I,J), I = 1, 10, 2), F(J, 3), J = 1, K)
```

If this list is used with an output statement, the information will be written on the output medium in this order:

```
A, B(3), C(1), D(1, K), C(2), D(2, K), . .,
C(10), D(10, K), E(1, 1), E(3,1), . . . , E (9,1),
F(1, 3),
E(1, 2), E(3, 2), . . . , E(9, 2), F(2, 3),
. . . ,
E(1, K), E(3, K), . . . , E(9, K), F(K, 3)
```

Similarly, if this list is used with an input statement, the successive values, as they are read from the external medium, are placed into core storage in the indicated order. The list reads from left to right with repetition for variables enclosed within parentheses. Only variables, not constants, may be listed.

If such a list is used, the execution is exactly that of a DO loop. It is as though each opening parenthesis (except subscripting parentheses) were a DO, with indexing given immediately before the matching closing parenthesis, and with the DO range extending up to that indexing information. The order of the above list can thus be considered the equivalent of the following "program":

```
     1. OUTPUT A
     2. OUTPUT B(3)
     3. DO 5 I = 1, 10
     4. OUTPUT C(I)
```

5. OUTPUT D(I, K)
6. DO 9 J = 1, K
7. DO 8 I = 1, 10, 2
8. OUTPUT E (I, J)
9. OUTPUT F (J, 3)

Note that indexing information, as in DO's, consists of three constants or fixed-point variables, and that the last of these may be omitted, in which case, it is assumed to be 1.

For a list of the form K, A(K) or of the form K, (A(I), I = 1, K), where an index or indexing parameter itself appears earlier in the list of an input statement, the indexing will be carried out with the newly read-in value.

Input/Output in Matrix Form

As outlined in a previous section, FORTRAN II-D treats variables according to conventional matrix practice. Thus, the input/output statement

READ 1, ((A (I, J), I = 1, 2), J = 1, 3)

causes the reading of I x J (in this case, 2 x 3) items of information. The data items are read into storage in the same order as they are found on the input medium.

INPUT/OUTPUT OF ENTIRE MATRICES

When input/output of an entire matrix is desired, an abbreviated notation may be used for the list of the input/output statement; only the name of the array need be given and the indexing information may be omitted.

Thus, if A has previously been listed in a DIMENSION statement, the statement,

READ 1, A

is sufficient to read in all the elements of the array. The elements of the array are stored in successively higher storage locations. (If A has not previously appeared in a DIMENSION statement, only the first element would be read in.)

Lists for the RECORD and FETCH statements must be formed in the following manner:

1. *Matrix Lists*
   If any item in the list is a matrix, all items in the list must be matrices. All matrices will start at the beginning of a record. Matrices written with a matrix list must be read with a matrix list.
2. *Element Lists*
   An element list may consist of any one or more of the following types (assume K previously defined).
   a. A
   b. B(1)
   c. C(1, K)
   d. (B(I), D (I, K), I = 1, 10)
   e. ((E (I, J), I = 1, 10, 2), F (J, 3), J = I, K)
3. The mode and order of lists must be the same for the reading and writing of the same data.

Arrangement of Arrays in Storage

Arrays are stored "column-wise," with the first of their subscripts varying most rapidly, and the last varying least rapidly. Arrays which are 1-dimensional are simply stored sequentially. A 2-dimensional array named A would be stored sequentially in the order $A_{1,1}$, $A_{2,1}, \ldots, A_{M,1}, A_{1,2}, A_{2,2}, \ldots, A_{M,N}$. A 3-dimensional array named T would be stored in the order $T_{1,1,1}, T_{2,1,1}, T_{3,1,1}, \ldots, T_{M,1,1}, T_{1,2,1}, \ldots, T_{M,N,1}, T_{1,1,2}, T_{2,1,2}, \ldots,$

The storage of arrays is in ascending order, i.e., the elements are stored sequentially in locations with ascending addresses.

READ Statement

The READ statement is used to read data into core storage from the 1622 Card Read-Punch.

GENERAL FORM

READ n, List

where *n* is the statement number of a FORMAT statement and List is a list of the quantities to be read.

EXAMPLES

READ 8, A, B, C
READ 211, VOLT (I), OHM (J)

The READ statement causes data to be read from a card and causes the quantities from the card to become the values of the variables named in the list. Successive cards are read until the complete list has been "satisfied," i.e., all data items have been read, converted, and stored in the locations specified by the list of the READ statement. The FORMAT statement to which the READ refers, describes the arrangement of information on the cards and the type of conversion to be made.

ACCEPT TAPE Statement

The ACCEPT TAPE statement is used to cause data to be read into core storage from the 1621 Paper Tape Reader.

GENERAL FORM

ACCEPT TAPE n, List

where *n* is the statement number of a FORMAT statement, and List is as described under INPUT/OUTPUT STATEMENTS.

EXAMPLE

```
ACCEPT TAPE 30, K, A (J)
```

The ACCEPT TAPE statement causes the object program to read information from the paper tape reader. Record after record is brought in, in accordance with the FORMAT statement, until the complete list has been satisfied.

ACCEPT Statement

The ACCEPT statement is used to allow data to be read in from the console typewriter.

GENERAL FORM

ACCEPT n, List

where n is the statement number of a FORMAT statement, and List is as described under INPUT/OUTPUT STATEMENTS.

EXAMPLE

```
ACCEPT 20, A, B, C, D (3)
```

The ACCEPT statement causes the object program to return the carriage of the console typewriter to await the entrance of data. The information is entered in accordance with the FORMAT statement until the complete list has been satisfied.

PUNCH Statement

The PUNCH statement is used to cause data to be punched out in cards by the 1622 Card Read-Punch.

GENERAL FORM

PUNCH n, List

where n is the statement number of a FORMAT statement, and List is as described under INPUT/OUTPUT STATEMENTS.

EXAMPLE

```
PUNCH 40, ( A (J), J = 1, 10)
```

The PUNCH statement causes the object program to punch cards in accordance with the FORMAT statement until the complete list has been satisfied.

PRINT and TYPE Statements

The PRINT statement and the TYPE statement are used to type out data on the console typewriter.

GENERAL FORM

PRINT n, List
TYPE n, List

where n is the statement number of a FORMAT statement and List is as described under INPUT/OUTPUT STATEMENTS.

EXAMPLE

```
PRINT 2, ( A (J), J = 1, 10)
```

The PRINT and TYPE statements cause output data to be typed on the console typewriter. A carriage return occurs and successive lines are typed in accordance with the FORMAT statement, until the complete list has been satisfied.

PUNCH TAPE Statement

The PUNCH TAPE statement is used to cause data to be punched by the 1624 PAPER TAPE PUNCH.

GENERAL FORM

PUNCH TAPE n, List

where n is the statement number of a FORMAT statement, and List is as described under INPUT/OUTPUT STATEMENTS.

EXAMPLE

```
PUNCH TAPE 25, ( A (J), J = 1, 10)
```

The PUNCH TAPE statement causes information to be punched by the paper tape punch.

Successive records are punched in accordance with the FORMAT statement until the complete list has been satisfied.

FIND Statement

This statement is used to position the disk access arm over a cylinder.

GENERAL FORM

FIND (I)

where I specifies the record number where reading or writing will start. The parameter I must be either:

1. A nonsubscripted fixed-point variable.

EXAMPLE

FIND (IMAX)

or

2. A subscripted fixed-point variable.

EXAMPLE

FIND (IMAX(3))

The FIND statement causes the disk access arm to be positioned over a cylinder which will subsequently be read from or written on. The FIND statement may precede a FETCH or RECORD statement that contains the same I parameter, and, in this manner, takes advantage of additional processing time while the access arm is moving.

The record numbers (I) start at 1, and correspond to every sector if one-sector records are specified in the DEFINE DISK statement; if two-sector records are specified, the record numbers correspond to every second sector.

Only areas of disk storage within the area defined by a DEFINE DISK statement can be specified by a FIND.

FETCH Statement

This statement is used to read data from the 1311 Disk Storage Drive.

GENERAL FORM

FETCH (I) List

where I specifies the record number and List is as described under INPUT/OUTPUT STATEMENTS.

EXAMPLE

FETCH (IMAX (3)) (A (J), J = 1, 10)

The FETCH statement may be preceded by a FIND statement containing the same I parameter. When the FETCH statement is executed, a check is performed to determine if the access arm is positioned over the proper cylinder. If the access arm is properly positioned, reading begins; if it is not, a seek is initiated (seek time is not available for computation).

The data designated by the list is read from the record specified by (I). If the list specifies more items than can be obtained from one record, then the value of (I) is incremented by one and reading proceeds from the next sequential record. This procedure continues until either the list has been "satisfied," i.e., until the data for all the variables in the list has been read in, or until the end of the area specified by N_2 (see DEFINE DISK) has been reached. At the conclusion of a read operation, the value of I is one greater than the number of the last record read. The parameter (I) is the same as described for the FIND (I) statement.

The compiled instructions for the FETCH statement cause control to be transferred to the Monitor Input/Output routine (see Supervisor section).

RECORD Statement

This statement is used to write data on the 1311 Disk Storage Drive.

GENERAL FORM

RECORD (I) List

where I specifies the record number and List is as described under INPUT/OUTPUT STATEMENTS.

EXAMPLE

RECORD (IMAX (3)) (A (J), J = 1, 10)

The RECORD statement may be preceded by a FIND statement containing the same I parameter. When the RECORD statement is executed, a check is performed to determine if the access arm is positioned over the proper cylinder. If the access arm is properly positioned, writing begins; if it is *not*, a seek is initiated (seek time is not available for computation).

The data designated by the list is written on the record specified by (I). If the list specifies more items than can be contained in one record, then the value of (I) is incremented by one and writing proceeds to the next sequential record. This procedure continues until either all items in the list have been written or until the end of the area specified by N_2 (see DEFINE DISK) has been reached. At the conclusion of a write operation, the value of I is one greater than the number of the last record written. The parameter (I) is the same as described for the FIND (I) statement.

The compiled instructions for the RECORD statement cause control to be transferred to the Monitor Input/Output routine (see Supervisor section).

Specification Statements

The SPECIFICATION statements supply necessary information to the FORTRAN processor, or information to increase program efficiency. No executable instructions are created in the object program for a SPECIFICATION statement.

DIMENSION Statement

The DIMENSION statement provides the information necessary to allocate storage for arrays in the object program.

GENERAL FORM

```
DIMENSION v, v, v, . . .
```

where each v is the name of a variable, subscripted with 1, 2, or 3 unsigned fixed-point constants. Any number of v's may be given.

EXAMPLE

```
DIMENSION A(10), B(5, 15), CVAL (3, 4, 5)
```

Each variable which appears in subscripted form in a program or subprogram must appear in a DIMENSION statement of that program or subprogram; the DIMENSION statement must precede the first appearance of that variable. The DIMENSION statement lists the maximum dimensions of arrays; in the object program, references to these arrays must never exceed the specified dimensions.

The above example indicates that B is a two-dimensional array for which the subscripts never exceed 5 and 15. The DIMENSION statement, therefore, causes 75 (i.e., 5 x 15) fields to be set aside for the array B.

A single DIMENSION statement may specify the dimensions of a number of arrays. The maximum number is limited by the number of continuation cards permitted. A program must not contain a DIMENSION statement which includes the name of the program itself, or any program which it calls. If any of the subscripts in a DIMENSION statement exceeds 9999, an error will be indicated.

EQUIVALENCE Statement

The EQUIVALENCE statement provides one method of controlling the allocation of data storage in the object program.

GENERAL FORM

```
EQUIVALENCE (a, b, c, . . . ), (d, e, f, . . . ) , . . .
```

where a, b, c, d, e, f, . . . are variables that may be subscripted with constants only.

EXAMPLE

```
EQUIVALENCE ( A, B(1), C(5) ), ( D (17), E(3) )
```

When the logic of the program permits, the number of storage locations used can be reduced by causing locations to be shared by two or more variables. The EQUIVALENCE statement should not be used to obtain mathematical equality between two or more elements. If fixed-point and floating-point variables are equivalenced, their word lengths must be the same, i.e., $f. + 2$ must equal k.

An EQUIVALENCE statement may be placed anywhere in the source program, except as the first statement in the range of a DO. Each pair of parentheses of the statement list encloses the names of two or more quantities which are to be stored in the same locations during execution of the object program; any number of equivalences may be given.

In an EQUIVALENCE statement, a term such as $C(p)$ can be defined for $p>0$ to mean the pth location of the C array. For example, $C(5)$ would be the fifth location in the C array. Note that in an EQUIVALENCE statement a two- or three-dimensional array must be referenced by a linear subscript (a single subscript notation which denotes the element of an array regardless of how the array is dimensioned). If p is not specified, it is understood to be 1.

Thus, the example indicates that the A, B, and C arrays are to be assigned storage locations such that the elements A(1), B(1), *and* C(5) are to occupy the same location. In addition, it specifies that D(17) and E(3) are to share the same location.

Quantities or arrays which are not mentioned in an EQUIVALENCE statement are assigned unique locations.

COMMON Statement

Variables, including arrays, appearing in COMMON statements are assigned to specific storage locations. Storage is assigned separately for each program compiled.

GENERAL FORM

```
COMMON A, B . . .
```

where A, B . . . are the names of variables and *non*-subscripted array names.

EXAMPLE

```
COMMON X, ANGLE, MATA, MATB
```

The COMMON storage area may be shared by a program and its subprograms. In this way, the COMMON statement enables a data storage area to be shared between programs in a way analogous to that by which the EQUIVALENCE statement permits data storage-sharing within a single program. Where the logic of the programs permits, this can result in a large saving of storage space.

Array names appearing in the COMMON statement must previously have appeared in a DIMENSION statement in the same program.

The COMMON storage area is located at the high end of core storage, starting with address 19999, 39999 or 59999. Variables in a COMMON statement are assigned storage locations in descending sequence. For example:

```
COMMON A, B, C
```

With $f = 10$, A, B, and C would be stored in locations 19999, 19987, and 19975 and similarly for 40,000 or 60,000 positions. If C is dimensioned as C(10), then 19975 is the address of C(10), which is the last element in the array, and 19867 is the address of C(1).

The COMMON statement takes precedence over the EQUIVALENCE statement. Due to the complex interaction of these two statements, the programmer must adhere to the following two rules:

1. Variables which are to be placed in COMMON storage must be assigned prior to any EQUIVALENCE statement containing these variables. For example,

   ```
   COMMON A
   EQUIVALENCE (A, B, C)
   ```

   The order in which the variables appear in the EQUIVALENCE statement is irrevelent and rule 1 applies if the COMMON variable is B or C.

2. Within an EQUIVALENCE list there may be no more than one variable which previously has been:
   a. equivalenced, or
   b. placed in COMMON.

The following sequence of statements is invalid:

```
EQUIVALENCE (A, B, C)
EQUIVALENCE (X, Y, Z)
EQUIVALENCE (A, Z)       Violates (a) above
COMMON      D
EQUIVALENCE (D, X, P)    Violates combina-
                         tion of (a) and
                         (b).
```

The sharing of storage locations desired in the above statements can be achieved by writing the statements as follows:

```
COMMON      D
EQUIVALENCE (D, X, P)
EQUIVALENCE (A, B, C, X)
EQUIVALENCE (X, Y, Z)
```

or

```
COMMON      D
EQUIVALENCE (D, A, P, B, C, X, Y, Z)
```

A diagnostic error message results if either Rule 1 or 2 is violated.

Arguments in Common Storage

COMMON statements may be used as a medium for transmitting arguments from the calling program to the called FORTRAN function or SUBROUTINE subprogram. In this way, they are *implicitly*, rather than *explicitly* transmitted as when listed in the parentheses following the subprogram name.

To obtain implicit arguments, it is necessary to have only the corresponding variables in the two programs occupy the same location. This can be accomplished by having them occupy corresponding positions in COMMON statements of the two programs. For example, (A, B, C) and (E, F, G) become implicit arguments when the calling program contains the statement COMMON A, B, C, and the called subroutine contains the statement COMMON E, F, G.

NOTES:

1. To force correspondence in storage locations between two variables in different programs which otherwise would occupy different relative positions in COMMON storage, it is valid to place dummy variable names in a COMMON statement. These dummy names, which may be dimensioned, will cause reservation of the space necessary to cause correspondence.
2. While implicit arguments can take the place of all arguments in CALL-type subroutines, there must be at least one explicit argument in a FORTRAN function. Here, too, a dummy variable may be used for convenience.

   When one variable is EQUIVALENCED to a second variable which appears in a COMMON statement, the first variable is also located in COMMON storage.

DEFINE DISK Statement

The DEFINE DISK statement specfies to the FORTRAN processor the size and quantity of data records that will be used with a particular program and its associated subprograms. This statement must appear in the main program (or link program) and may appear only *once* in that program, when Disk I/O statements appear in any part of the program or subprograms. Thus, all subprograms used by that main program or link program must use the same size record defined in the statement.

GENERAL FORM

DEFINE DISK (N_1, N_2)

where the parameters N_1 and N_2 are defined as follows:

N_1 – a fixed-point constant which specifies the number of words contained in a record of data. The value chosen for N_1 depends upon two things: (1) the word length *(w)* specified when the program was compiled, and (2) whether the user wants the length of a record of data to be one or two physical sectors.

The value (N_1) is determined by following two rules:
If w times $(N_1) \leqq 100$, then the record length will be one disk sector.
If w times (N_1) is more than 100 and $\leqq 200$, then the record length will be two disk sectors.

For example, assume that the word lengths specified at compile time were 8 for floating-point numbers *(f)* and 4 for fixed-point numbers *(k)*. Since a record might contain all floating-point or all fixed-point numbers (words), the larger of the two specified word lengths must be used to determine w. In this example, the floating-point length is the larger of the two word lengths; its total length is 10 (word length $= f + 2$). Therefore, if a data record is to be contained in one physical disk sector (100 disk locations), then N_1 must be in the range of 1 to 10. An N_1 of 10 would be making the most efficient use of the available disk storage. In this example, if the length of a data record is to be two physical disk sectors, then N_1 would be in the range of 11 to 20. A data record may not be greater than 2 sectors (200 digits).

If arrays are read or written, the variables are not moved to a buffer area before going to or coming from the disk provided that both f and k are even in length. In this case, a group mark is placed at the end of the array before writing to disk. If 10-digit variables are used, the most efficient use of the disk would be with arrays containing 9, 19, 29, 39, etc. variables, so that the group mark is placed in the same sector as the variables to be recorded.

N_2 – a fixed-point constant which specifies the number of data records that will be used by this main program and its associated subprograms. N_2 is used by the compiler to reserve a portion of the specified work cylinder area (see DEFINE PARAMETERS ROUTINE in the DISK UTILITY PROGRAM section of this manual) for the purpose of transferring data to and from disk storage. The number of sectors that the compiler will reserve depends upon the record length specified by N_1. If one-sector records are specified, then N_2 sectors will be reserved; if two-sector records are specified, then 2 times N_2 sectors will be reserved.

FORMAT Statement

The FORMAT statement is used to describe the format of data being transmitted to and from the typewriter, card, or paper tape units.

GENERAL FORM

FORMAT $(s_1, \ldots, s_n)$

where s_1 is a format specification. The FORMAT specifications must be separated by commas, slashes, or left parentheses.

EXAMPLE

FORMAT (I2/ (E12.4, F10.4))

The Input/Output statements, in addition to the list of quantities to be transmitted, contain the statement number of a FORMAT statement describing the information format to be used. The FORMAT statement also specifies the type of conversion to be performed between the internal machine language and the external notation. FORMAT statements are not executable: their function is merely to supply information to the object program. Therefore they may be placed anywhere in the source program (except as the first statement in the range of a DO).

For the sake of clarity, examples given in this section are for typing on the console typewriter. However, the description is valid for any input/output unit simply by generalizing the concept of "typewritten line" to that of the unit record in the selected input/output unit. Thus, a unit record may be:

1. A typewritten line with a maximum of 87 characters.
2. A punched card with a maximum of 80 characters.
3. A paper tape record with a maximum of 87 characters. (The input record length may be variable up to 87; the output record length is fixed at 87.)

Numerical Fields

Three forms of conversion for numerical data are available:

FROM/TO INTERNAL	TYPE	TO/FROM EXTERNAL
Floating-point variable	E	Floating-point number with exponent
Floating-point variable	F	Floating-point number without exponent
Fixed-point variable	I	Integer

These types of conversion are specified in the forms:

E*w.d*, F*w.d*, and I*w*.

where *w* and *d* are unsigned fixed-point constants.

Format specifications are used to describe the input and output format. The format is specified by giving, from left to right, beginning with the first character of the record;

1. The control character (E, F, or I) for the field.
2. The width (*w*) of the field. The value of *w* must be large enough to include the field *d*, plus spaces for a sign and the decimal point. In addition, four spaces for the exponent are needed in E-type conversion. The width specified may be greater than required to provide for spacing between numbers.
3. For E- and F-type conversions, the number of decimal positions (*d*) (of the field) which appear to the *right* of the decimal point.

Specifications for successive fields are separated by commas. No format specification should be given that provides for more characters than the input/output unit record. Thus, a FORMAT statement for typewriter output should not provide for more than 87 characters per line, including blanks. For example: the statement FORMAT (I2, E12.4, F10.4) might cause the following line to be typed:

$$\overbrace{\text{b7}}^{\text{I2}}\overbrace{\text{-92.3100E+00}}^{\text{E12.4}}\overbrace{\text{bbbb-.0076}}^{\text{F10.4}}$$

(In these examples, b is included to indicate blank spaces.)

Alphameric Fields

FORTRAN II-D provides a method by which alphameric information may be read or written.

The specification for this purpose, *wH*, is followed in the FORMAT statement by *w* alphameric characters. For example:

24H THIS IS ALPHAMERIC DATA

Note that blanks are considered alphameric characters and must be included as part of the count *w*.

Information handled with the *H* specification is not given a name and may not be referred to or manipulated in storage in any way.

The effect of *wH* depends on whether it is used with input or output.

1. *Input, w* characters are extracted from the input record and replace the *w* characters included with the specification.
2. *Output.* The *w* characters following the specification, or the characters which replaced them, are written as part of the output record. Blanks are *not* ignored in an *H* specification as they are elsewhere.

For example: The statement FORMAT (3HXY = F8.3) could produce any of the following lines:

XY = b–93.210
XY = b999.999
XY = bb28.768

Another alphameric specification, *Aw*, causes *w* alphameric characters to be read into or written from a variable or array name. Since each alphameric character is represented in core storage by two decimal digits, *w* must be less than, or equal to, the largest whole number resulting from *k*/2 or *f*/2, depending on whether the variable or array name is fixed or floating. If *k* or *f* is odd, a zero will be supplied as the least significant digit for the field in core storage. To facilitate manipulation of alphameric fields which are stored as floating-point numbers, the numbers will have zero as an exponent. This will have no effect on input/output. However, if the first character in a field is a blank, decimal point, or close parenthesis, the field will be treated as zero in the floating-point arithmetic subroutines.

Blank Fields

Blank characters may be provided in an output record, and characters of an input record may be skipped, by means of the specifications *w*X where $0 \leqq w \leqq 87$ (*w* is the number of blanks provided or characters skipped). When the specification is used with an input record, *w* characters are considered to be blank, regardless of what they actually are, and are skipped over.

Repetition of Field Format

It may be desired to print *n* successive fields within one record, in the same fashion. This may be specified by giving *n* (where *n* is an unsigned fixed-point constant which must be ≤ 99) before E, F, I, or A. Thus, the statement FORMAT (I2, 3E12.4) might result in:

27 – 92.3100E + 00b75.8000E – 02b55.3600E – 02

Repetition of Groups

A limited parenthetical expression is permitted in order to enable repetition of data fields according to certain format specifications within a longer FORMAT statement specification. Thus, FORMAT (2(F10.6, E10.2), I4) is equivalent to FORMAT (F10.6, E10.2, F10.6, E10.2, I4). The number of repetitions is limited to a maximum of 99.

Scale Factors

The *E*-type specification implies a scale factor. Therefore, *E*16.8 for an output field will result in the printing or punching of a maximum of ten significant digits in the form (–)XX.XXXXXXXXE(–)XX. A maximum of *f* digits can be placed to the right of the decimal point if the *d* specification is greater than *f*. In this case, *d–f* low-order zeros will be inserted to satisfy the *d* specification. The following guide may be used when working with *E*-type specifications.

1. If *f* (floating-point precision) $\leqq w–6$, then *f* significant digits will be printed or punched.

2. If $f > w–6$, then $w–6$ significant digits will be printed or punched. For example, if $f = 10$ and the floating-point number is stored as $\bar{1}23456789\bar{1}\bar{3}\bar{5}$, it will be printed as –12.34567891E–37, according to specification *E*16.8.

The *F*-type specification also implies a scale factor. Therefore, *F*16.8 for an output field will result in the printing or punching of a maximum of fourteen significant digits in the form (–)XXXXXX.XXXXXXXX. However, a maximum of *f* digits will be placed to the right of the decimal point and the result will be right justified in the output field. If *f* is larger than $w–2$, only $w–2$ digits will appear in the output.

The X specification should be used to space fields in the *E*-type format. In the statement

E16.8, 1X, E16.8, 1X, E16.8

a space will be provided between adjacent fields.

A field read according to the *E*-type format need not have the exponent E(–)XX; i.e., it may actually take the same form as the *F*-type format.

The *P*-scale factor may be used in a specification but it will be ignored by the FORTRAN II-D processor.

Multiple Record Formats

To deal with a block of more than one typewritten line, a FORMAT specification may have several different one-line formats, separated by a slash (/) to indicate the beginning of a new line. Thus, FORMAT (3F9.2, 2F10.4/8E14.5) specifies a multiline typewritten block in which line 1 has format 3F9.2 and 2F10.4, and line 2 has format 8E14.5.

If a multiple-line format is desired, such that the first two lines are typed according to a special format and all remaining lines are typed according to another format, the last line specification should be enclosed in a second pair of parentheses; e.g., FORMAT (I2, 3E12.4/2F10.3, 3F9.4/(10F12.4)). If data items remain to be transmitted after the last line format specification has been completely satisfied, the format repeats from the last left parenthesis.

As these examples show, both the slash and the closing parenthesis of the FORMAT statement indicate the termination of a record.

Blank lines may be introduced into a multiline FORMAT statement by listing consecutive slashes.

Format and Input/Output Statement Lists

The FORMAT statement indicates, among other things, the maximum size of each record to be transmitted. In this connection it must be remembered that the FORMAT statement is used in conjunction with the list of some particular input/output statement, except when a FORMAT statement consists entirely of alphameric fields. When the FORMAT statement is used with the list, control in the object program switches back and forth between the list (which specifies whether data remains to be transmitted) and the FORMAT statement (which gives the specifications for transmission of that data).

Automatic Fix/Float

During execution of input/output statements, it is permissible to read a fixed-point argument into a floating-point field or a floating-point argument into a fixed-point field, and to write from a floating-point field in a fixed-point format or from a fixed-point field in floating-point format. During reading, the format specification dictates the data conversion, and the list designation controls the mode of storing the argument. During writing, the format specification dictates the mode of the field printed or punched.

Ending a Format Statement

During input/output of data, the object program scans the FORMAT statement to which the relevant input/output statement refers. When a specification for a numerical field is found and list items remain to be transmitted, input/output takes place according to the specification, and scanning of the FORMAT statement resumes. If no items remain, transmission ceases and execution of that particular input/output statement is terminated. Thus, a numerical input/output operation will be brought to an end when a specification for a numerical field or the end of the FORMAT statement is encountered, and there are no items remaining in the list.

Data Input to the Object Program

Input data to be read when the object program is executed must be in essentially the same format as given in the previous examples. Thus, a card to be read according to FORMAT (I2, E12.4, F10.4) might be punched:

```
27b-0.9321Eb02bbb-0.0076
```

Within each field, all information must appear at the extreme right. Plus signs may be omitted or indicated by a b (blank) or +. Blanks in numerical fields are regarded as zeros, but zeros may not be substituted for blanks. For example, a sign cannot be preceded by zeros. Numbers for E-type and F-type conversion may contain any number of digits, but only the high-order *f* digits are retained. Numbers for I-type conversion may not contain more than *k* significant digits. The concept of *f* and *k* is treated in this manual under CONSTANTS, VARIABLES, SUBSCRIPTS, AND EXPRESSIONS.

To permit economy in punching, certain relaxations in input data format are permitted.

1. Numbers for E-type conversion need not have four columns devoted to the exponent field. The start of the exponent field must be marked by an E, or if the E is omitted, by a + or − (not a blank). Thus E2, E02, +2, 02, Eb02, and E+02 are all permissible exponent fields. Blanks are not permitted between characters in the exponent field except for the optional blanks which may replace a plus sign. Numbers for E-type conversion must be right-justified in the data record field.
2. Numbers for E-type or F-type conversion need not have their decimal points punched. If not punched, the FORMAT specification will supply them; for example, the number −09321+2 with the specification E12.4 will be treated as though the decimal point has been punched between the 0 and the 9. If the decimal point is punched in the card, its position overrides the indicated position in the FORMAT specification.

Library Functions

There are seven library functions (which are a part of 16 FORTRAN relocatable subroutines) included in the 1620 Monitor I system. These subroutines are selected for loading only when called for in the object program. The functions are:

TYPE OF FUNCTION	FORTRAN NAME
Logarithm (natural)	LOGF
Exponential	EXPF
Cosine of an angle given in radians	COSF
Sine of an angle given in radians	SINF
Arctangent of an angle given in radians	ATANF
Square Root	SQRTF
Absolute Value	ABSF

The name of the library function is followed by the argument enclosed in parentheses. The argument can be a variable (subscripted or not subscripted), or an expression.

EXAMPLES

```
A = COSF (B)
A = SQRTF (BETA)
Y = A - SINF (B* SQRTF (C) )
```

For the last example, the assembled instructions of the object program will:

1. Branch to the square root subroutine to compute the value of C.
2. Multiply the square root value of C (obtained in step 1) by B.

3. Branch to the SINF subroutine to compute the sine of the value obtained from step 2.
4. Subtract the value computed so far from the variable A.
5. Replace the present value of the variable Y with the value of the complete expression.

Approximation Method and Estimated Errors

Results of the library subroutines are truncated, and, in general, errors are no greater than one in the last digit of the mantissa. Approximation methods and errors for functional subroutines are described in greater detail in the following paragraphs.

1. *Logarithm.* The natural logarithm of the fractional part of the positive argument is evaluated by using a power series expansion. The exponent of the argument is multiplied by ln 10. The product is added to the logarithm of the fraction, and the sum is the logarithm of the argument. For an argument with its value A in the range $.99 < A \leqq 1.01$, the leading digits of its logarithm will be zeros, and the result will contain less than *f* significant digits because of normalization. The maximum truncation error in the result is $\pm 10^{-f}$
2. *Exponential.* The value of e^A, where A is the value of the argument, is calculated by using a series approximation for 10^A. For $|A| = 227.955924206...$ an exponent overflow will result for $A>0$ or exponent underflow for $A<0$. The value of A is multiplied by log e and the product separated into an integer and a fractional part. The integer becomes the exponent of the result and the fractional part is used to produce its mantissa by series approximation. If A is greater than zero, the maximum error in the result is $\pm 5 \times 10^{-f}$.
3. *Cosine-Sine.* The cosine and sine functions of an argument with value A in radians are computed by using a series approximation for cosine A with sine A = cosine $(\frac{\pi}{2} - A)$. The value A is reduced to within the range $-\frac{\pi}{2} \leqq A \leqq \frac{\pi}{2}$. For arguments with exponents less than 03, the magnitude of the maximum truncation error in the mantissa of the result does not exceed 10^{-f}. Accuracy in the mantissa of the result decreases as the size of the argument (exponent 03 or greater) increases.
4. *Arctangent.* The arctangent function of an argument with value A is evaluated by using a series approximation. The result is given in radians. The maximum truncation error in the mantissa of the result is $\pm 10^{f}$, except for results with an exponent less than or equal to -2. The maximum error for these results is ± 1 in the $(f+1)$ decimal place.
5. *Square Root.* The square root is derived by the odd integer method. The result is accurate to 1 in the last digit of the mantissa.
6. *A ** B.* A^B is evaluated as EXPF (B*LOGF (A)). Three subroutines, logarithm, multiply, and exponential, are involved. An error in one of these subroutines may propagate other errors or increase the error in a succeeding subroutine. Normally, the magnitude of the error does not exceed 10^{1-f}.

Additional Library Functions

Up to fourteen additional functions can be added to the library of subroutines. These functions are defined (written) in machine language or SPS (see ADDING SUBROUTINES TO FORTRAN LIBRARY).

GENERAL FORM

NAME (A)

where NAME is 1 to 6 alphabetic or numerical characters (no special characters) of which the first is alphabetic, and A is the argument enclosed in parentheses.

EXAMPLE

TIME (A)

The mode of the additional library function is determined by its argument.

EXAMPLE

TIME (ABLE) Floating point
TIME (LABEL) Fixed point

Library functions can be called by means of an arithmetic expression that includes the name of the function. The appearance of the name in the arithmetic expression serves to call the function; the value (a single numerical quantity) of the function is then computed, using the argument which is supplied in the parentheses following the function name. Only one value is produced by a given Library function. The mode of a Library subroutine is determined by its argument.

EXAMPLES

```
        COS (A)   Floating point
        COSH (I)  Fixed point
```

The relocatable Library subroutines supplied with the 1620 FORTRAN II-D System, with the exception of Absolute Value Function (ABSF), will not accept fixed-point arguments.

Arithmetic Statement Functions

These functions are defined by a single FORTRAN II-D arithmetic statement and apply only to the particular program in which they appear. They are named in the same manner as the Library functions:

> The name of the function consists of 1 to 6 alphabetic or numerical characters (not special characters) of which the first must be alphabetic. The name of the function is followed by parentheses enclosing the arguments, which are separated by commas.

The function statement is defined as follows:

GENERAL FORM

```
NAME(ARG)= E
```

where NAME is a function name followed by parentheses enclosing its arguments (which must be non-subscripted variables) separated by commas, and E is an expression which does not involve subscripted variables. Any functions appearing in E must be available to the program or already defined by preceding arithmetic statements. The function names in the main program must agree with those defined in the arithmetic statements and FUNCTION statements.

EXAMPLES

```
FRSTF (X) = A*X+B
SCNDF (X,B) = A*X+B
THRDF (D) = FRSTF(E)/D
FRTHF (F,G) = SCNDF (F, THRDF (G) )
FFTHF (I,A) = 3.0*A**I
SXTHF (J) = J + K
```

As is the case with the Library functions, the appearance of the name in the arithmetic statements serves to call the function. The value of the function (a single numerical quantity) is then computed, using the arguments which are supplied in the parentheses following the function name. Only one value is produced by a given arithmetic statement function.

In 1620 FORTRAN II-D, the *mode of the value is determined by the function name*, e.g., if the function name begins with I through N, the mode will be fixed point.

The right-hand side of a function statement may be any expression not involving subscripted variables that meets the requirements specified for expressions. In particular, functions may be used freely, provided that any such functions, if it is not a Library function, has been defined in a *preceding* function statement. No function can be used as an argument of itself.

As many as desired of the variables appearing in the expression on the right-hand side may be stated on the left-hand side as arguments of the function. Since the arguments are really only dummy variables, their names are unimportant (except insofar as they indicate fixed-point or floating-point mode) and they may even be the same as names appearing elsewhere in the program.

Those variables on the right-hand side which are not stated as arguments are treated as parameters Thus, if FRSTF is defined in a function statement as FRSTF(X) = A*X+B then a later reference to FRTSF(Y) will cause $ay+b$, based on the current values of a, b, and y, to be computed. The naming of parameters, therefore, must follow the normal rules of uniqueness.

A function defined by a function statement may be used in the same way as any other function. Its arguments may be expressions and may involve subscripted variables; thus, a reference to FRSTF (Z+Y(I)), as a result of the previous definition of FRSTF, will cause $a(z+y_i) + b$ to be computed on the basis of the current values of a, b, y, and z.

Functions defined by arithmetic statements are always compiled as closed subroutines.

All the arithmetic statements defining functions to be used in a program must precede the first executable statement of the program.

FORTRAN Functions and Subprograms	Method of Naming	Method of Defining	Method of Calling
Library (closed) function	Same for four	Individual	Same for three
Arithmetic Statement function		Individual	
FUNCTION subprogram		Same for two	
SUBROUTINE subprogram			Individual

Dummy Variables within an Arithmetic Statement Function

A variable appearing as a dummy argument within an arithmetic statement function must not previously have been defined except as a dummy argument in a previous arithmetic statement function. After the variable is used as a dummy argument, it may appear elsewhere in the program.

Subprogram Statements

Subroutines which are referred to by other FORTRAN II-D programs can be written as subprograms in the FORTRAN II-D language. A subroutine is considered to be any sequence of instructions that performs a desired operation. A subprogram is defined as a program written in FORTRAN language that is referred to or used by another FORTRAN source program.

Two types of FORTRAN II-D coded subprograms are available: the FUNCTION subprogram and the SUBROUTINE subprogram. Four statements, SUBROUTINE, FUNCTION, CALL, and RETURN, are necessary for their definition and use.

Although FUNCTION subprograms and SUBROUTINE subprograms are treated together and may be viewed as similar, it must be remembered that they differ in two fundamental respects:

1. The FUNCTION subprogram, which results in a FORTAN function, as defined under FUNCTIONS, is always single-valued, whereas the SUBROUTINE subprogram may be multivalued.
2. The FUNCTION subprogram is called or referred to by the arithmetic expression containing its name; the SUBROUTINE subprogram can only be referred to by a CALL statement (see CALL STATEMENT).

Subprograms of each of these two types are coded in FORTRAN II-D language. In all respects, they conform to the rules for FORTRAN programming.

FUNCTION Statement

The FUNCTION statement, always first in a FUNCTION subprogram, defines it as a FORTRAN FUNCTION subprogram.

GENERAL FORM

FUNCTION Name ($a_1, a_2, \ldots, a_n$)

where Name is the symbolic name of a single-valued function, and each argument $a_1, a_2, \ldots, a_n$, of which there must be at least one, is a nonsubscripted variable name.

The function name consists of 1 to 6 alphabetic or numerical characters, the first of which must be alphabetic.

EXAMPLES

```
FUNCTION ARCSN (RADS)
FUNCTION ROOT (B, A, C)
FUNCTION  INTRT (RATE, YEARS)
```

In a FUNCTION subprogram, the name of the function must appear either in an input statement list, or at least once as the variable on the left-hand side of an arithmetic statement. An example of the latter is:

```
FUNCTION NAME (A, B)
.
.
.
NAME=Z+B
.
.
.
RETURN
```

The value of the function is returned to the calling program. The mode of a function subprogram is determined by its name.

EXAMPLES

```
FUNCTION AMAST (A, K) Floating point
FUNCTION IAMAST (A, K) Fixed point
```

The arguments following the name in the FUNCTION statement may be considered as "dummy" variable names, that is, during object program execution other actual arguments are subsituted for them. Therefore, the arguments which follow the function reference in the calling program must agree in number, order, and mode with those in the FUNCTION statement in the subprogram. Furthermore, when a dummy argument is an array name, the corresponding actual argument must also be an array name. Each of these array names must appear in similar DIMENSION statements within its respective program. *None of the dummy variables may appear in EQUIVALENCE statements in the FUNCTION subprogram.*

SUBROUTINE Statement

GENERAL FORM

SUBROUTINE Name ($a_1, a_2, \ldots, a_n$)

where Name is the symbolic name of a subprogram, and each argument, $a_1, a_2, \ldots, a_n$, if any, is a nonsubscripted variable name. The name of the subprogram consists of 1 to 6 alphabetic or numerical characters, the first of which must be alphabetic.

EXAMPLES

```
SUBROUTINE MATMP (A, N, M, B, L, C)
SUBROUTINE QDRT (B, A, C, ROOT 1, ROOT 2)
```

The SUBROUTINE statement, always first in a SUBROUTINE subprogram, defines it as a SUBROUTINE subprogram. A subprogram introduced by the SUBROUTINE statement must be a FORTRAN program and may contain any FORTRAN II-D statements except FUNCTION, DEFINE DISK, or another SUBROUTINE statement.

A SUBROUTINE subprogram must be referred to by a CALL statement in the calling program. The CALL statement specifies the name of the subprogram and its arguments.

Unlike the FUNCTION subprogram which results in the calculation of only a single numerical value, the SUBROUTINE subprogram uses one or more of its arguments to return results. Therefore, the arguments so used must appear on the left side of an arithmetic statement in the subprogram (or alternately, in an input statement list within the subprogram).

The arguments of the SUBROUTINE statements are dummy names that are replaced, at the time of execution, by the actual arguments supplied in the CALL statement. There must, therefore, be correspondence in number, order, and mode, between the two sets of arguments. Furthermore, when a dummy argument is an array name, the corresponding actual argument must also be an array name. Each of these array names must appear in similar DIMENSION statements within its respective program.

For example, the subprogram headed by

```
SUBROUTINE MATMP (A, N, M, B, L, C)
```

could be called by the main program through the CALL statement

```
CALL MATMP (X, 5, 10, Y, 7, Z)
```

where the dummy variables, A, B, C, are the names of matrices. A, B, C must appear in a DIMENSION statement in subprogram MATMP, and X, Y, Z must appear in a DIMENSION statement in the calling program. The dimensions assigned must be the same in both statements.

None of the dummy variables may appear in EQUIVALENCE statements in the SUBROUTINE subprograms. These subprograms may be independently compiled or used in a multiple compilation with others.

CALL Statement

The CALL statement refers only to the SUBROUTINE subprogram, whereas the RETURN statement is used by both the FUNCTION and SUBROUTINE subprograms.

GENERAL FORM

CALL Name ($a_1, a_2, \ldots, a_n$)

where Name is the name of a SUBROUTINE subprogram, and $a_1, a_2, \ldots, a_n$ are arguments.

EXAMPLES

```
CALL MATMP (X, 5, 10, Y, 7, Z)
CALL QDRT (P*9.732, Q/4.536, R-S**2.0, X1, X2)
```

This statement is used to call SUBROUTINE subprograms; the CALL transfers control to the subprogram and presents it with the parenthesized arguments. Each argument may be one of the following types:

1. Fixed-point constant.
2. Floating-point constant.
3. Fixed-point variable, with or without subscripts.
4. Floating-point variable, with or without subscripts.
5. Arithmetic expression.

The arguments presented by the CALL statement must agree in number, order, mode, and array size with the corresponding arguments in the SUBROUTINE statement of the called subprogram, and none of the arguments may have the same name as the SUBROUTINE subprogram being called.

RETURN Statement

EXAMPLE

```
RETURN
```

This statement terminates any subprogram of the type headed by either a SUBROUTINE or a FUNCTION statement, and returns control to the calling program. A RETURN statement must, therefore, be the last executed statement of the subprogram. It need not be the last statement of the subprogram physically, but can be any point reached by a path of control. Any number of RETURN statements may be used.

CALL LINK Statement

This statement is used to call a new program from disk storage and transfer to the first executable statement in that program.

GENERAL FORM

CALL LINK (NAME)

where NAME is the name of a FORTRAN program as contained in the Equivalence table. The program name must be formed with one to six alphabetic or numerical characters (no special characters) of which the first is alphabetic.

EXAMPLES

CALL LINK (JOE)
CALL LINK (PROG18)

The CALL LINK statement is used to call another program into core storage. The program that is called will cause all subprograms and library subroutines that it references to be read into core storage (the arithmetic and I/O subroutines are also reloaded). Any program called by using the CALL LINK statement must be in disk storage or it is assumed that the "link" program is the *first* mainline program encountered by the system input unit. If the logic of the program allows one of several links to possibly be called, it is necessary that the link programs be on disk storage. If a subprogram is not available (in disk storage) that the "link" program references, the FORTRAN loader will request that the missing subprogram be loaded into core from cards or paper tape.

Only 50 links that call LOCAL subprograms can appear in any one FORTRAN job.

The COMMON area is not destroyed during the loading of the link programs. If the size of COMMON differs (between the calling program and the link program being called), the COMMON area size will be the size defined for the new program.

FORTRAN II-D Processor

The 1620 FORTRAN II-D Processor program is used to change a user-written FORTRAN *source* program into an *object* program of 1620 machine language instructions. All programs are compiled in relocatable format, i.e., the program instruction addresses are compiled relative to a starting address of 00000. The instruction addresses must be modified before execution can take place.

The processor operates under control of the Monitor I Supervisor program. It can be called into operation only by use of the FOR or FORX Monitor Control records.

The Monitor I system permits the following FORTRAN operations:

1. FORTRAN source program compilation.
2. FORTRAN source program compilation and immediate execution of the compiled program. From the programmer's point of view, this is equivalent to entering a source program into the machine as an object program.
3. Object programs may be placed in disk storage after compilation and/or they may be punched in cards or paper tape.
4. Execution of FORTRAN object programs that are in disk storage or are in cards or paper tape.
5. Execution of programs in "links," a procedure necessary where the total program is too large to fit into core storage at one time. A "link" is a section of the total program (see CALL LINK STATEMENT).

General Compilation Process

Although the process of compiling an object program is a continuous one, there are two phases through which the source statements pass before an object program is compiled. The user enters the source statements using cards, paper tape, or typewriter input and obtains output in cards, paper tape, or on the typewriter or disk storage. The input and output units are selected by the use of control records. The object program may be placed permanently on disk and it may be punched out in either paper tape or cards.

The source statements are analyzed during Phase I and broken apart into instruction generating elements that are strings of 4-digit codes. These strings are then written on disk storage for use by Phase II, which outputs the 1620 coding in relocatable format. Errors are indicated on the console typewriter as they are detected. The final output of any compilation is a single program or subprogram. Unless this program is an independent entity, capable of being executed without other programs or library functions, the user will need to load the other programs before execution can take place. To initiate the loading process, the user may (1) call the compiler with a FORX Control record, or (2) call the compiled program using an XEQS Control record (see MONITOR I CONTROL RECORDS). A program loaded by means of a FORX or XEQS Control record will also have all associated subroutines and subprograms loaded with it. (All associated subprograms are loaded except those defined as "load-on-call"; see LOCAL CONTROL RECORD).

The arrangement for stacked input to process a source program is shown in Figure 14. the FOR or FORX Monitor Control record is followed by optional FORTRAN Control records, followed by the source program. The inclusion of a PAUS Monitor Control record will allow the operator to set the Console Program switches to the desired position (options for Program switches are shown in Table 9.) After setting the switches, the operator depresses START and the Supervisor reads the FOR or FORX record, the FORTRAN Control records, if any, and begins compilation of the source program.

During Phase I (reading the source program and creating instruction generating elements), the Phase I source program errors that are listed in Table 10 may be detected. The Phase I errors are of two types: Type I, compilation continues, but outputting of intermediate output is stopped; Type II, compilation and outputting both continue.

As a program is being compiled, it is placed in a disk storage area used for temporary storage. When all source statements have been read and the instruction generating elements entered in the temporary storage area, Phase II of the processor takes control. Phase II converts the Phase I instruction codes into machine language instructions and places the instructions in the temporary disk storage area. When all intermediate output is processed, a message denoting the end of compilation is printed. Depending on the FORTRAN Control records loaded with the source program, any of the following options can occur:

1. Control is returned to the Supervisor. Control records for output were not used; compilation was apparently for editing only of source program.
2. Object program is loaded to permanent area of disk storage and/or outputted in cards or paper tape.
3. Object program is executed using data from input unit or disk storage.

The compiled object program contains a header record which specifies various parameters and information needed when it is to be loaded for execution, such the program name, length of the program, *f* and *k*, FORTRAN program constant, and indicators to specify the Library subroutines used.

Subprogram identification records, consisting of 18 digits of name and address information, are created for each subprogram called by a program. Up to 100 subprograms may be used with any one FORTRAN main program or Link program.

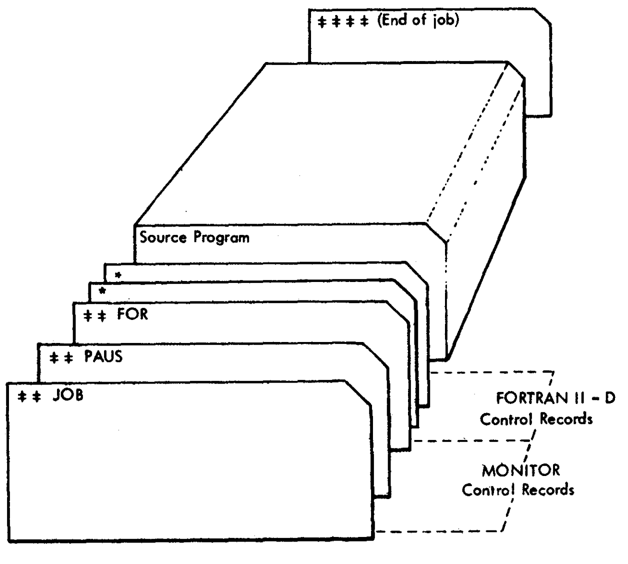


Figure 14. Typical Stacked Input for FORTRAN Compilation

FORTRAN II-D Control Records

The FORTRAN Compiler can utilize four control records that specify output options, etc. When they are used, these records may be in any order but they must be read in between the FOR or FORX Control record and the source program statements as shown in Figure 15.

The FORTRAN Control records must have an asterisk in column 1 and the Name must be punched beginning in column 2. If a control record (* in column 1) is read and is not a legally named record, the message shown below is typed and the program halts.

ERROR, INVALID CONTROL RECORD

The operator must correct the invalid record in the input unit and depress START.

The prescribed format and specific function of each control record is described below.

FANDK. The FORTRAN II-D Compiler, as delivered to the user, will process an object program with a floating-point *word* length of 10 digits (*f* of 08 + 2=10) and a fixed-point word length of 4 digits. The operator may vary these lengths, at *compilation time,* by using the FANDK control record. The format of the FANDK Control record is as follows:

Columns	1-6	* FANDK
	7-8	*ff*
	9-10	*kk*
	11-80	not used

where *ff is the floating-point mantissa length* and *kk is the fixed-point word length.*

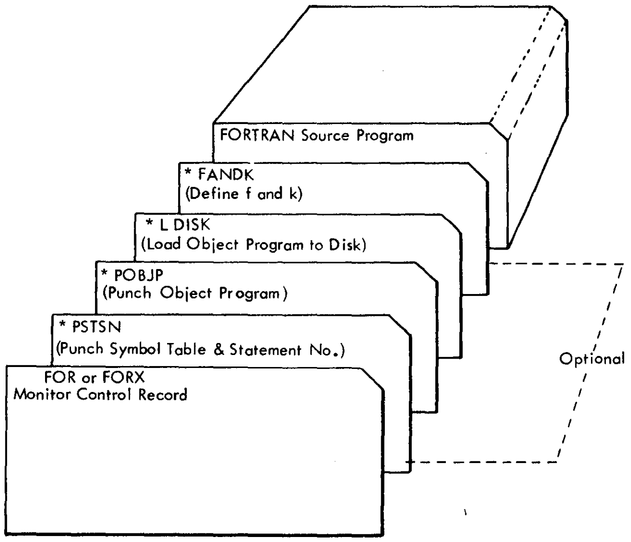


Figure 15. FORTRAN II-D Control Records

If entry is from the console typewriter, the same format must be followed.

The range of *f* is 2 through 28, of *k*, 4 through 10. If *f* or *k* is out of the prescribed range, the following message is typed:

ERROR, F OR K OUTSIDE RANGE

PSTSN. This control record causes the symbol table and addresses of numbered statements to be punched. The format is as follows:

Columns	1-6	*PSTSN
	7	n
	8-80	not used

where n is 2 if paper tape output is desired or 4 for card output. See NOTE below.

POBJP. The POBJP Control record causes the object program to be punched following compilation. The format is as follows:

Columns	1-6	*POBJP
	7	n
	8-80	not used

where n is the same as for the PSTSN Control record.

The format of the processor output (object program) is given under LOADER ROUTINE of the SUPERVISOR section of this publication.

NOTE: If n is not 2 or 4 in the PSTSN or POBJP Control records, the following message is typed out on the console typewriter and the program will halt.

ERROR, INVALID OUTPUT UNIT CODE

The operator must correct the record that contains the error and depress the 1620 Start key.

LDISK. The LDISK Control record causes the object program to be moved to a permanent area of disk storage following compilation. The format for the LDISK control statement is:

Columns	1-6	*LDISK
	7-12	name (optional)
	13-16	number (optional)
	17-80	not used

where Name is the left-justified program name, and number is a 4-digit DIM entry number not already in use. If a DIM entry is not supplied, the Disk Utility program will assign one.

After compilation, the Disk Utility Program will load the programs to disk and create a DIM entry for the program. At that time, the Name supplied (in the LDISK record) will be placed in the Monitor Equivalence table. It is not necessary to supply the name of a FUNCTION or SUBROUTINE subprogram. The name used in the FUNCTION or SUBROUTINE statement will be used.

Entering the Source Program

The source program can be entered in the form of a punched paper tape, a deck of punched cards, or a list of statements to be typed in at the console typewriter. This entry option is specified in the FOR or FORX Monitor Control record.

OPERATING PROCEDURES

All of the following operations may be performed before the processor is called, except possibly items 1 and 3. If the operation taking place just prior to the compilation of a source program required the Console Program switches to be set differently than the desired settings for compiling a Monitor PAUS Control record should have been inserted before the FOR or FORX record. This will allow time for the operator to change the switches.

The operations required to process a source program are as follows:

1. Set the Console Program switches for the desired compilation operations (see Table 9).
2. Set all check switches to PROGRAM.
3. If punching is to take place, ready the paper tape punch with feed code leader or, ready the card punch by loading blank cards and depressing the Punch Start key.
4. Place a FOR or FORX Control record in the input unit (see the Monitor I Control Records section for format).

Table 9. Program Switch Settings for FORTRAN II-D

SWITCH	ON	OFF
1	Source statements are typed on the console typewriter as they are processed. Source statement errors are typed in the form ERROR n.* At the end of Phase 1, symbol table and statement numbers are typed out.	Source statements are not listed. Source statement errors are typed in the form SSSS and CCCC ERROR n.* Symbol table and statement numbers are not typed out.
2	Trace instructions for arithmetic statements are compiled but no additional instructions are generated.	Trace instructions for arithmetic statements are not compiled.
3	A trace instruction is compiled to trace the value of the expression generated in an IF statement. An additional instruction is generated in the object program for every IF statement.	Trace instructions for IF statements are not compiled.
4	Errors made while typing source statements can be corrected by a. turning on switch 4, b. pressing the Release and Start keys,	c. turning off switch 4, d. retyping statement.

*See description under Phase 1 Errors

5. Place any desired FORTRAN Control records in the input unit (see FORTRAN II-D CONTROL RECORDS).
6. Place the source program statements in the input unit (specified in the FOR or FORX record).

To resume machine operation, if the machine was stopped to allow the operator to perform any of the above operations, depress the Start key.

Typewriter Input. If source statements are entered by way of the console typewriter, each statement must be terminated with a record mark. After a statement is typed, the operator must depress the R-S key to process that statement. As soon as a statement is processed, the carriage returns to await entry of the next statement. A statement of up to 330 characters may be typed with no intervening punctuation, spacing, etc.

Normally card format need not be followed, however, in a comment statement the C must be followed by at least two blanks (spaces) before the comment is typed.

Phase I Errors

During Phase I of compilation, a number of tests are made for source program errors. If an error is found in a source statement and Program Switch 1 is on, a message in the form

ERROR n

is typed, where n is the error code (see Table 10). If switch 1 is off, the error message is in the form

SSSS + CCCC ERROR n

where ssss is the last statement number encountered by the program prior to the error, and cccc is the number of statements following the last numbered statement. ssss + cccc is the statement that contains the error. For example the message

0509 + 0012 ERROR 1

means that the twelfth statement following the statement numbered 509 is incorrect. If an error occurs before a statement number is encountered, ssss will be 0000. Errors detected after processing the END statement reference the END statement. Comment cards, blank cards, and continuation cards are not included in the statement count.

If any Type I errors (see Table 10) are found during Phase I, no attempt is made to process the source program through Phase II. At the completion of Phase I, control is returned to the Monitor I Supervisor program (Monitor Control Record Analyzer routine).

If a Type II error is found (other than Error 60), compilation continues on through Phase II. However, any FORTRAN Control records specifying output that were included with the source program will be disregarded and control will transfer to the Supervisor program at the completion of Phase II. If Error 60 is encountered, normal processing is continued since N_1 and N_2 can be corrected when loading the object program (see SUBROUTINE ERROR CHECKS).

Phase II Errors

During processing of the intermediate output, certain checks are performed which were impossible to perform during Phase I. If an error is detected, an error message in one of the following forms is typed:

XXXX SYMBOL TABLE FULL
XXXX IMPROPER DO NESTING
XXXX DO TABLE FULL
XXXX MIXED MODE

Table 10. FORTRAN Phase I Source Program Errors

TYPE 1: Compilation continues but outputting of intermediate output is stopped. Only one error of this type is detected in any one statement.

Error No.	Condition
1	Undeterminable, misspelled, or incorrectly formed statement.
2	Syntax error in a nonarithmetic statement (exception: DO statements).
3	Dimensioned variable used improperly, i.e., without subscripting, or subscripting appears on a variable not previously dimensioned.
4	Symbol table full (processing may not be continued).
5	Incorrect subscript.
6	Same statement number assigned to more than one statement.
7	Control transferred to FORMAT statement.
8	Variable name greater than 6 alphameric characters.
9	Variable name used both as a nondimensioned variable name and as a Subroutine or Function name.
10	Invalid variable within an EQUIVALENCE statement.
11	Subroutine or Function name or dummy variable used in an EQUIVALENCE statement (subprogram only).
12	k not equal to f + 2 for equivalence of fixed point to floating point variables.
13	Within an Equivalence list, placement of two variables previously in Common, or one variable previously equivalenced and another either equivalenced or placed in Common.
14	Sense Switch number missing in an IF (Sense Switch n) statement.
15	Statement number or numbers missing, not separated by commas, or nonnumerical in a transfer statement.
16	Index of a computed GO TO missing, invalid, or not preceded by a comma.
17	Fixed point number greater than k digits.
18	Invalid floating point number.
19	Incorrect subscripting within a DIMENSION statement.
20	First character of a name not alphabetic.
21	Variable within a DIMENSION statement previously used as a nondimensioned variable, or previously dimensioned or used as a Subroutine or Function name.
22	Dimensioned variable used within an arithmetic statement function.
23	More than four continuation cards.
24	Statement number in a DO statement appeared on a previous statement.
25	Syntax error in a DO statement.
26	FORMAT number missing in an input/output statement.
27	Statement number in an input/output statement appeared previously on a statement other than a FORMAT statement, or a number on a FORMAT statement appeared in other than an input/output statement.
28	Syntax error in input/output list or an invalid list element.
29	Syntax error in CALL statement, or an invalid argument.
30	SUBROUTINE or FUNCTION statement not the first statement in a subprogram.
31	Syntax error or invalid parameter in a SUBROUTINE or FUNCTION statement.
32	Syntax error or invalid variable in a COMMON statement.
33	Variable in a Common list previously placed in Common or previously equivalenced.
34	Library function name appeared to the left of an equal sign or in a COMMON, EQUIVALENCE, DIMENSION, or input/output statement; or function name not followed by a left parenthesis.
35	Syntax error in FORMAT statement, or invalid FORMAT specifications.
36	Invalid expression to the left of an equal sign in an arithmetic expression.
37	Arithmetic statement function preceded by the first executable statement.
38	Invalid expression in an IF or CALL statement, or invalid expression to the right of an equal sign in an arithmetic statement.
39	Unbalanced parenthesis.
40	Invalid argument used in calling an Arithmetic statement function or Function subprogram.
41	Syntax error in disk I/O statement.
42	Disk I/O list omitted.
43	Disk I/O list contains both simple variables and array names.
44	COMMON exceeds core storage size. (May occur when large array is defined.)

TYPE 2: Compilation of intermediate output continues.

Error No.	Condition
51	DO loop ended with a transfer statement.
52	No statement number for next executable statement following a transfer statement.
53	Improperly ended nonarithmetic statement.
54	Unnumbered CONTINUE statement.
55	Number of Common addresses assigned in excess of storage capacity because of Equivalence. See note at end of Table.
56	Statement number or subscript greater than 9999 (only first 4 significant digits are retained).
57	RETURN statement appeared in program other than a subprogram (statement ignored).
58	RETURN statement not contained in a Subroutine or Function subprogram.
59	Statement number not defined. See note at end of Table.
60	Syntax error in DEFINE DISK statement, invalid use of, or DEFINE DISK statement missing.

NOTE: Errors 55 and 59 are not detected if Type I errors occur during compilation.

where xxxx is the relative number of the statements within the program, not counting storage allocation statements, comments, or blank cards. The number does not correlate with an actual statement number.

If an IMPROPER DO NESTING or MIXED MODE message occurs, compilation is continued, but only to check for other errors. The FORTRAN Control records, PSTSN, POBJP, and LDISK will be disregarded. the object program will not be executed and control will be returned to the Supervisor program.

Compilation stops immediately after the SYMBOL TABLE FULL or DO TABLE FULL message is typed and control is returned to the Supervisor program. The approximate allowable number of symbols differs with the core storage size of the source machine. For a 1620 with 20,000 positions, approximately 200 symbols are allowed. For a 1620 with 40,000 or 60,000 positions, the number of symbols allowed is approximately 1200 or 2200, respectively.

End of Compilation

When all of the intermediate output is processed, the following messages are printed:

```
nnnnn CORES USED
aaaaa NEXT COMMON
END OF COMPILATION
```

where nnnnn is the number of core positions the object program requires (object program and data areas except COMMON), and aaaaa is the next available core storage position of the COMMON area, (aaaaa + 1 is the last used position of COMMON).

If FORTRAN Control records specifying output are included with the source program, the outputting takes place following the END OF COMPILATION message.

Identification Data

When a program (or subprogram) is compiled, it is headed with an identification record that will be used when the program is to be loaded into core storage for execution.

Both main program and subprogram header identification records are shown and described as below:

Mainline or Link

$\bar{0}010\bar{0}$	2	$\bar{6}7$	$\bar{9}87898$	$\bar{N}1$	$\bar{N}2$	$\bar{W}$ord Length	$\bar{R}$ec Length	Length	$\bar{f}f$	$\bar{k}k$	$\bar{F}$irst Core	$\bar{N}$ext Common	Subroutine Indicators
5	1	2	6	2	5	2	3	5	2	2	5	5	30

Digits

Subprogram

$\bar{0}010\bar{0}$	2	$\bar{6}7$	$\bar{9}87898$	$\bar{S}$ubprogram Name	Length	$\bar{f}f$	$\bar{k}k$	$\bar{E}$ntry Address Less Six	$\bar{N}$ext Common	Subroutine Indicators
5	1	2	6	12	5	2	2	5	5	30

$\bar{0}010\bar{0}$	The address of the origin of the program less 100.
2	An indicator to the relocating loader that a constant to be relocated is forthcoming.
$\bar{6}7$	The number of digits in the forthcoming constant.
$\bar{9}87898$	An arbitrarily chosen constant to identify this as a header record for a FORTRAN program.
Subprogram Name	The name of the program in double digit representation (left-justified). Used only in subprograms and FORTRAN function headers.
N1	The number of words per disk record. (From the DEFINE DISK FORTRAN statement.) This field is present only for mainline programs and links.
N2	The number of logical records in the disk, as used by the FORTRAN program. (From the DEFINE DISK FORTRAN statement.) This field is only present in mainline program and link header records.
Word Length	The number of digits in the words used to determine a logical record. This value is the larger of the floating word and the fixed word length.
Rec. Length	The number of sectors to be used when reading or writing logical records. This value is limited to the numbers 1 and 2.
Length	The length of the program (This must be an even number).
ff and kk	The length of the mantissa and the fixed point words in this program.
Entry Address Less Six	The first location in the subprogram, less six, to enter the subprogram.
First Core	The first location in the program to begin execution. Present only for mainline or link programs.

Next COMMON	The next location available in COMMON. This must be an even address (e.g., 19998) so that COMMON can correctly be written on disk during operation of the FORTRAN loader. Subprograms do not use this value.
Sub. Indicators	A digit position for each library subroutine in the FORTRAN system.

The identification record occupies one whole sector when it is on the disk. The format for the balance of this sector, if the program is in relocatable format, is shown below:

$$\bar{0}0000\bar{2}\bar{1}701234567891234567$$

Subprograms Called by FORTRAN

The names of the subprograms called by a program are stored at the end of the program. The address within the calling program where the address of the subprogram will be placed is also stored along with the name of the subprogram. These 18-digit name and address records are created for the subprograms called and the last record is followed by a record mark. Up to 100 subprograms may be used with any one FORTRAN main program or link (50 can be loaded with the program; 50 can be called on an as-needed basis, i.e., LOCAL).

```
|       Name        |  Address  | 0 |
     12 Digits           5        1
```

The names and addresses of the subprograms called are moved to the FORTRAN loader work area when the subprogram is loaded. This FORTRAN loader will determine which of the subroutines called by the subprogram have not already been loaded, and will load those routines (exception: LOCAL subprograms cannot call a new subroutine; see OBJECT PROGRAM EXECUTION). The proper addresses are placed within the calling programs to link them with the subroutines that they call.

Trace Feature

Under program switch control, instructions can be compiled into the object program to enable the operator to trace the flow of the program when it is executed. During execution of the object program, the trace output is under control of Program Switch 4 as described under OBJECT PROGRAM EXECUTION.

The trace output contains the value of the left-hand side of each executed arithmetic statement and/or, the value of the expression calculated in an IF statement.

Subroutines

The subroutines for 1620 FORTRAN II-D are divided into two groups: (1) Library subroutines and (2) Arithmetic and Input/Output subroutines.

LIBRARY SUBROUTINES

Sixteen relocatable subroutines are included in 1620 FORTRAN II-D (see Table 11).

The Library subroutines are loaded only when they are used in a program, "link" program, or subprogram, i.e., they are loaded before any execution of the calling program takes place but they are loaded only if required by the calling program. During compilation of a program, a 30-digit field of zeros is created (in the header record). When a subroutine is called for by a source statement or *"required for use,"* a 1 is inserted in the proper location of the subroutine indicator field. The position in the field corresponds directly to the Subroutine number given in Table 11.

"Required for use" means that even though the user has not directly called a specific subroutine it may be required by the system. For example, the LOGF and EXPF subroutines are used to compute the values of floating-point roots and powers in arithmetic statements. They are loaded, if required, before execution of the program that requires them. Likewise, the Subscripting routines (library numbers 3, 4, 5), and the disk routines (library numbers 6-11), though never directly called, will be loaded if required. Sections of library routines 6-11 are loaded to disk storage Clyinder 1 (relative to the start of the disk work area) and the specific section required is loaded from the disk to core storage only when it is needed. All reading and writing of even arrays (f and k even) will be done without loading additional instructions from disk storage.

It is possible for the user to add 14 subroutines to the FORTRAN library. The subroutines can be written using Symbolic Programming System (SPS) language. Procedures for adding subroutines and information for writing subroutines in SPS is given under ADDING SUBROUTINES TO THE FORTRAN LIBRARY.

Two forms of the supplied library subroutines are included with the FORTRAN II-D System. One form is for users that have the floating-point feature installed on their machine; the other form operates without the floating-point feature. Only one form is loaded when the Monitor I System is initially loaded by the user.

The FORTRAN statements FIND, RECORD, and FETCH are processed by relocatable subroutines numbered 6 through 11. These routines are loaded into core storage only if the disk FORTRAN statements are utiliz-

Table 11. FORTRAN II-D Library Subroutines

					STORAGE REQUIREMENTS	
TYPE OF FUNCTION	SYMBOLIC NAME	SUB-ROUTINE NUMBER	DIM ENTRY NUMBER	TRANSFER ADDRESS	WITH FLOATING POINT FEATURE	WITHOUT FLOATING POINT FEATURE
Logarithm (natural)	LOGF	1	10	02248	802	850
Exponential	EXPF	2	11	02253	1062	1158
Subscripting (1 dimension)	ENTSC1	3	12	02258		
Subscripting (2 dimensions)	ENTSC2	4	13	02263	510	510
Subscripting (3 dimensions)	ENTSC3	5	14	02268		
FIND	ENTFID	6	15	02273		
RECORD	ENTREC	7	16	02278		
FETCH	ENTFET	8	17	02283	1300	1342
Routine to load or unload disk buffer	ENTSWD	9	18	02288		
Routine to write or read arrays	ENTARR	10	19	02293		
Routine to complete FETCH or RECORD	ENTCPT	11	20	02298		
Cosine	COSF	12	21	02303	904	976
Sine	SINF	13	22	02308		
Arctangent	ATANF	14	23	02313	1234	1258
Square Root	SQRTF	15	24	02318	526	538
Absolute Value	ABSF	16	25	02323	58	82

ed in the object program. Different routines may be used to RECORD (or FETCH) an array. The routines that will write out or read in an entire array with one disk instruction will be used if both the fixed word length and the floating word length are even (not necessarily equal). If either of these variables was defined as odd in length, the array will be split into records that are the same length as those used when reading or writing lists of variables. The maximum speed in reading and writing of data from and to the disk is attained with even values for f and k.

The FIND statement is used to position the access arm in the disk storage drive in advance of a FETCH or RECORD. It may be necessary for the FORTRAN system to change the position of the arm after a FIND operation has been initiated. If this is the case, the same location for the arm as specified in the FIND statement will be sought again after the FORTRAN system operation is complete. This automatic repositioning to the FIND cylinder will occur after every arm disturbance until the next FETCH or RECORD statement has been executed.

The FORTRAN statements that use subscripting notation will determine which of three different subscripting subroutines are entered by the program. Subroutine numbers 3, 4 and 5 are used to identify subscripting routines that handle one, two and three dimensions, respectively (this is actually one subroutine with three entry points).

ARITHMETIC AND INPUT/OUTPUT ROUTINES

The arithmetic and input/output subroutines, including constants and working areas, are basic routines needed for proper execution of the object program. They are loaded without being specifically called for by the object program. Besides performing the fundamental tasks of adding, subtracting, etc., these routines also perform some diagnostic testing on the data being manipulated.

Two sets of these arithmetic and input/output routines are available. One set is for those users without the floating-point feature, and the second set for those users with the floating-point feature. The floating-point set operation is faster and in some cases, requires less core storage for the routines used to perform the arithmetic operations.

Each set of the arithmetic and input/output routines is supplied in two forms. One form of each set consists of all the arithmetic and I/O routines in core at one time. The second form of each set is sectionalized to form groups of arithmetic and I/O routines. These groups each contain several routines and are loaded to core storage only when required. The sets that are divided into groups allow the user's programs to be loaded into core storage starting at 07500. The all-in-core sets allow the user's programs to be loaded starting at 14000.

The routines to be read from disk when required will be placed in the first cylinder of the disk work area. (This is cylinder zero unless the user defines different parameters.) The routines are placed in this cylinder just before execution of the FORTRAN program. The first cylinder of the work area has been chosen because the access time when seeking this cylinder will be very short, and, consequently, the FORTRAN programs should operate at an optimum speed. When the in-core form of subroutines is selected, the routines are not stored in the first work storage cylinder. Only one set (one in-core storage set and one out-of-core storage set) will be in disk storage. The user selects the form according to his machine features. The user further selects either the in-core form or the out-of-core form as being standard. A different form may be selected from the one indicated as standard when the Monitor System is delivered (see DFINE CONTROL RECORD). The selection can be accomplished at load time by placing the proper digit in the 28th column of the XEQS Control statement. If no digit is placed in this column, the system standard will be used. The set that contains routines to be *read into core storage as required* is the one that will be utilized if no new standard has been specified.

The digits that may be placed in this column are listed below:

1—No floating-point feature – read in when required

2—No floating-point feature – in core

3—Floating-point feature – read in when required

4—Floating-point feature – in core

Any digit higher than 4 will result in the error message below:

ERROR L8

After this message, the loading routine will set the digit to the system standard (specified in column 83 of the disk sector communications area) and continue.

If a 3 or 4 is used for a set number and only set 1 and 2 are on the disk, 1 or 2 will be used, respectively. Similarly, if 1 or 2 is used and only 3 and 4 are on the disk, 3 will be used in place of 1, and 4 will be used in place of 2. No message will be typed in this event.

The arithmetic and input/output subroutines available with 1620 FORTRAN II-D are shown in Table 12. By referring to the symbolic names for the subroutines in the listing, their equivalent absolute addresses can be found.

Symbol Table Listing

If Program Switch 1 is in the ON position during the FORTRAN compilation, the storage addresses of the symbol table will be listed in the following order and form.

1. Floating point constants   Fixed point constants

$\bar{X}XXXX$ $\bar{M}MMMMMMM\bar{C}C$ $\bar{X}XXXX$ $\bar{F}FFFF$

where $\bar{X}XXXX$ is the low-order address of the constant.

Table 12. FORTRAN Arithmetic and Input/Output Subroutines

Subroutine	Symbolic Name	Operation
Floating Point Arithmetic		
Add	FAD	FAC + A → FAC
Subtract	FSB	FAC - A → FAC
Reverse Subtract	FSBR	A - FAC → FAC
Multiply	FMP	FAC x A → FAC
Divide	FDV	FAC / A → FAC
Reverse Divide	FDVR	A / FAC → FAC
Set FAC to zero	ZERFAC	0 → FAC
Fixed Point Arithmetic		
Add	FXA	FAC + I → FAC
Subtract	FXS	FAC - I → FAC
Reverse Subtract	FXSR	I - FAC → FAC
Multiply	FXM	FAC x I → FAC
Divide	FXD	FAC / I → FAC
Reverse Divide	FXDR	I / FAC → FAC
Common Subroutines		
Load FAC	TOFAC	A → FAC or I → FAC
Store FAC	FMFAC	FAC → A or FAC → I
Reverse Sign of FAC	RSGN	- FAC → FAC
Fix a Floating Point Number	FIX	FIX (FAC) → FAC
Float a Fixed Point Number	FLOAT	FLOAT (FAC) → FAC
Exponentiation		
Fixed Point J ** I	FIXI	FAC ** I → FAC
Floating Point A ** (±I)	FAXI	FAC ** (±I) → FAC
Floating Point A ** (±B)	FAXB	FAC ** (±B) → FAC
Input/Output		
Read Card	RACD	
Read Tape	RAPT	
Read Typewriter	RATY	
Write Card	WACD	
Write Tape	WAPT	
Write Typewriter	WATY	

FAC	-	simulated accumulator
A & B	-	floating point variables
I & J	-	fixed point variables
→	-	store in

$\bar{\text{M}}$MMMMMMM$\bar{\text{C}}$C is a floating-point constant.

$\bar{\text{F}}$FFFF is a fixed-point constant.

2. Simple variable   Dimensioned variables

   $\bar{\text{X}}$XXXX NAME $\bar{\text{X}}$XXXX NAME $\bar{\text{Y}}$YYYY

   where $\bar{\text{X}}$XXXX for simple variables is the address at object time where the value for NAME will be stored.

   $\bar{\text{X}}$XXXX for dimensioned variable is the address at object time of the first element in the array, NAME.

   $\bar{\text{Y}}$YYYY is the address of the last element in the array, NAME.

   If NAME* is typed, this indicates a dummy parameter within an arithmetic statement function.

3. Called subprograms

   $\bar{\text{X}}$XXXX NAME

   where $\bar{\text{X}}$XXXX is the location at which the starting address of the subprogram will be stored.

4. Statement numbers

   $\bar{\text{S}}$SSS $\bar{\text{X}}$XXXX

   where $\bar{\text{X}}$XXXX is the address of the first instruction generated for the statement numbered $\bar{\text{S}}$SSS.

If the statement number pertains to a FORMAT statement, the location $\bar{\text{X}}$XXXX will be the actual address of the FORMAT specification.

Symbol Table Listing for Subprograms

When compiling a subprogram, the dummy arguments are listed after statement numbers, as follows:

$\bar{\text{X}}$XXXX NAME

where $\bar{\text{X}}$XXXX is the location at which the actual address of the variable in the mainline program, corresponding to the argument, NAME, will be stored upon entering the subprogram. This same form is also used for simple and dimensioned variables.

The addresses listed are not the actual addresses at object time. Since programs are relocated upon loading, the listed addresses have to be adjusted relative to the starting location of the program or subprogram.

Object Program Execution

A 1620 FORTRAN II-D program may consist of three parts: a main program, a group of subprograms, and the library subroutines utilized by the main program and subprograms. The main program exercises control over the entire operation being performed. In addition to the normal execution of instructions, it has the ability to call subprograms and subroutines.

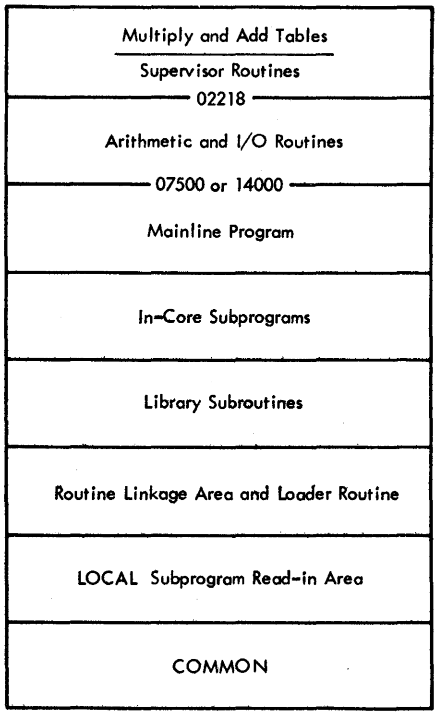
Subprograms, defined with a FUNCTION or SUBROUTINE subprogram statement at compilation time, can be placed into two groups (which have nothing to do with the way they were defined at compilation). For execution, subprograms are either:

1. Loaded into core along with the main program (or link program) that calls them, or,
2. They may be left on disk storage and brought into core storage only when called.

The user must determine which subprograms are to be loaded into core storage prior to execution and which are to be loaded when called for immediate execution. The subprograms that are to be loaded when called are defined by using a LOCAL Control record. The subprograms that are named in the LOCAL record will be loaded to the work area of disk storage prior to execution of the main program or link program that calls them.

A subprogram requires no changes in order to be a LOCAL subprogram. A subprogram written for use in core with the main program can be used instead as a LOCAL subprogram merely by naming it in a LOCAL control record. However, a LOCAL subprogram cannot call an "in core" subprogram that is not called from either (1) the main program, or, (2) another "in core" subprogram, and a LOCAL subprogram can never call another LOCAL subprogram.

The following illustration shows the layout of core storage during execution of a typical FORTRAN program.



Converting FORTRAN Object Programs to Core Image

FORTRAN object programs which are stored on disk are in System Output format. When called for execution, these programs must first be converted to core image format before they can be executed. In cases where the main program and/or the in-core subprograms are very long, the conversion time might become excessive. To eliminate this conversion each time a program is executed, the user can convert main programs and in-core subprograms to core image format and permanently store them on disk in that form.

The means of conversion is the Replace Programs routine, a disk utility function described earlier in this manual. A procedure for converting to core image format follows:

1. Clear core storage to zeros.
2. Load the Supervisor program to core storage.
3. Read in the following sequence of cards:

```
++JOB
++DUP

*DREPL  NAME  DIM1  DIM2  CORE  D M   F
 Col    7     13    17    39    49 50 60
```

Columns	1-6	Code word * DREPL.
	7-12	Alpha name of program.
	13-16	DIM entry number of new program.
	17-20	DIM entry number of program to be replaced. This will be the same as the number punched in Column 13-16.
	39-43	Fixed core storage address where program will be loaded when called for execution.
	49	Input unit (D for disk).
	50	M denotes conversion from System Output format to core image format.
	60	Any non-blank character.

NOTE: This procedure must be followed in its entirety each time the DREPL function is used with FORTRAN programs.

LOCAL Control Record

Subprograms that are to be loaded when needed are defined at *load time* by a LOCAL Control record. The format of the LOCAL Control record is as follows:

Columns	1-6	* LOCAL
	7-80	Main program name, comma, Subprogram name, comma, Subprogram name, comma, etc.

The main program may be identified in either of two ways. If it has a name in the system Equivalence table, that name may be used. If no name is in the Equivalence table to identify the main program, the name may be omitted. In this event, the comma normally following the program name must be retained. Blanks may not be included between names and commas. The commas must be placed between subprogram names. When more than one card or tape record is needed to identify all LOCAL subprograms, a comma must be placed following the last name on each record that is followed by another LOCAL record. The name of the main program is omitted from all continued LOCAL records, but they must contain the asterisk and code word LOCAL.

EXAMPLES

```
* LOCAL MLNAME, SUB1, SUB2, SUB3,
* LOCAL SUB4
* LOCAL LNKNAM, SUB2, SUB6
* LOCAL LNK2, SUB2, SUB6, SUB7
```

Up to 100 subprograms (50 maximum, in core; 50 maximum, LOCAL) may be called by a main program or link.

LOCAL records to be entered into the system must follow the FORTRAN source program when compiling and executing a program (see Figure 16). The number of LOCAL records to be read must be placed in columns 9-10 of the FORX record or columns 29-30 of the XEQS Control record. When typing the LOCAL record, no more than 79 columns may be used. When the LOCAL record is in paper tape, only 79 positions may be used and the positions used must be followed by a blank.

Library subroutines to be loaded at load time are selected by interrogation of the subroutine indicators in the header record that precedes each program, link, and subprogram. Each indicator, one for each subroutine, must be a numerical 1 if the corresponding subroutine is to be loaded to core storage at load time. All subroutines are loaded if called in the main program or if called in subprograms that are loaded to core storage. *A LOCAL subprogram must not call for a subroutine that was not called for by the main program or a subprogram loaded to core storage* (see Table 13).

DATA Control Record

The purpose of the DATA control record is to indicate to the FORTRAN loader that all segments of the program have been loaded prior to beginning execution.

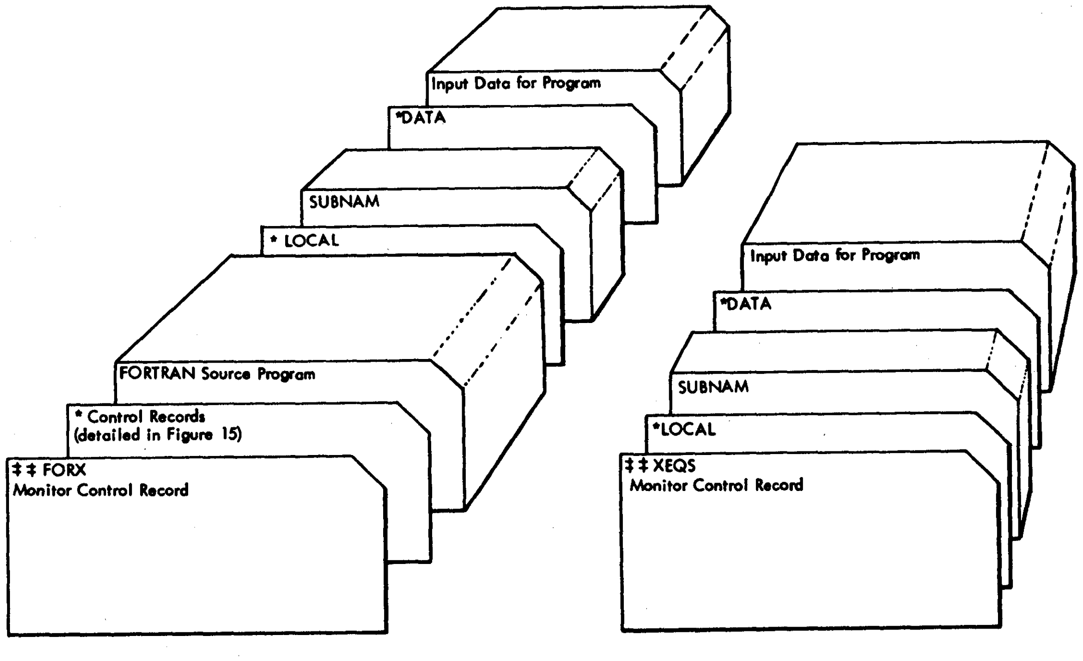


Figure 16. Positioning of LOCAL and DATA Control Records in FORTRAN Stacked Input

The rules for inclusion of the DATA control record are:

1. If the mainline or link program, or *any* of its associated subprograms are loaded from the paper tape reader or card reader, a DATA control record *must be included* in the stacked input whether or not any data is to be read by the program.
2. If the mainline or link program, and its associated subprograms are *all* loaded from disk a DATA control record must *not* be included in the stacked input.

ERROR CODE	MEANING, REASON	RESULT
L1	Invalid LOCAL control record Word LOCAL misspelled, misplaced, or no asterisk	Typeout JOB ABANDONED; branch to MONCAL*
L2	Invalid name in LOCAL record Not formed according to FORTRAN rules	Typeout JOB ABANDONED; branch to MONCAL*
L3	Multiple name in LOCAL record Same subprogram name appears more than once for some program or link, or program or link name appears more than once	Typeout JOB ABANDONED; branch to MONCAL*
L4	LOCAL subprogram table full Greater than 50 LOCAL subprograms per link not allowed Mainline table (link names) full More than 50 links calling LOCALS not allowed	Typeout JOB ABANDONED; branch to MONCAL*
L5	Invalid header record Does not conform to standard FORTRAN header record	Branch to MONCAL*
L6	Unequal F or K Subprogram F and/or K does not compare with main program F or K	Subprogram not loaded
L7	New subroutine called from LOCAL subprogram LOCAL subprogram cannot call new subroutine	Subprogram loaded; subroutine not loaded
L8	Invalid arithmetic and input/output subroutine set Not defined as 1, 2, 3, or 4	Set defined as system standard is loaded, depending on group loaded
L9	In-core subprogram table full Greater than 50 subprograms not allowed	Ignore above 50th subprogram
L10	New subprogram called from LOCAL subprogram LOCAL subprogram cannot call new subprogram	LOCAL subprogram loaded; new subprogram not loaded
L11	LOCAL subprogram disk storage area overlaps reserved disk work area	LOCAL subprogram is not loaded

*MONCAL is the symbolic name for Monitor Control Record Analyzer routine.

Table 13. FORTRAN Loader Errors

The format for the DATA Control record is as follows:

Card

Columns	1-5	*DATA
	6-80	must be blank

Paper Tape

```
     |←——75 zeros——→|
*DATA00000000000000000000Ⓔ
```

When the DATA record is recognized by the loading routine, a check is made to determine which subprograms have not yet been loaded. If there are any such subprograms, they are listed on the console typewriter, and the machine halts. The operator must then see that these subprograms are made available for the loading routine to load before depressing the Start key. If all subprograms have been loaded, any remaining data in the input unit will be skipped until the DATA record is read.

Console Program Switch Settings

Switch 1. When switch 1 is on, a list of the programs being loaded is typed on the console typewriter. The format of the list is:

XXXXXX NNNNN LLLLL LOADED

where xxxxx is the name of the program or subprogram or the number of the subroutine, NNNNN is the beginning core storage address, and LLLLL is the length of the program.

Switch 4. When switch 4 is on, and trace instructions have been compiled into the object program, the trace output is listed on the console typewriter. The trace output contains the value of the left-hand side of each executed arithmetic statement and/or, the value of the expression in an IF statement.

If the typewriter input is called for by the object program the operator must:

1. Type in the required data.
2. Turn Console Program Switch 4 to the OFF position.
3. Depress the Release key.
4. Depress the Single Instruction key 7 times.

5. Turn Console Program Switch 4 to the ON position.
6. Depress the Start key.

If the operator makes a mistake when typing the input data it is necessary only to depress the R-S key and retype the required data.

Operating Procedure

To execute a previously compiled FORTRAN program, the following items must be placed in the input unit.

1. JOB Control record.
2. XEQS Control record.
3. LOCAL Control records (if required).
4. Main program (if not previously loaded to disk storage).
5. Subprograms (if required and not previously loaded to disk storage).
6. DATA Control record. Note: This must be supplied even if data has been loaded to disk storage.
7. Input data (if not previously loaded to disk storage).
8. Job End Control record.

When called for execution, the main program is converted from relocatable format and loaded into core storage (see LOADER ROUTINE in Supervisor section of this manual for a description of operation and errors). Following the loading of the main program the "in-core" subprograms are loaded. If any subprograms are not available the message

LOAD SUBNAM

is typed, where SUBNAM is the name of the subprogram that must be loaded in the input unit.

When all "in-core" subprograms are loaded, the library subroutines needed by the main program and "in-core" subprograms are loaded into core storage. Following the loading of the subroutines, if any subprograms have been defined as LOCAL subprograms, an "object-time read-in routine" is loaded and following it a linkage area is reserved for each LOCAL subprogram.

Then, the first LOCAL subprogram is loaded *into core storage.* The address to which this subprogram is loaded will be the input address for all LOCAL subprograms. The first LOCAL subprogram is then moved to the top end of the work area of disk storage and the next LOCAL subprogram is loaded to core storage. LOCAL subprograms may be loaded to the system by way of the input unit, however they must be stacked following any 'in-core" subprograms to be loaded.

OVERLAP Errors

During the loading of the main program, subprograms, subroutines, or the read-in routine or the program linkage areas, the available core storage area may be exceeded.

If a main program or link program would exceed the available area the following message is typed and control is transferred to the Supervisor program (see MONITOR CONTROL RECORD ANALYZER ROUTINE).

NAME $\bar{X}$XXXX OVERLAP
JOB ABANDONED

NAME is the name of the program or link program, $\bar{x}$xxxx is the number of core storage positions required by that program. If the program has no assigned name, MAIN is printed for NAME.

If a subprogram would exceed the available area the NAME $\bar{x}$xxxx OVERLAP message is typed and the named program is not loaded. Subprograms following the "overlap subprogram" are loaded if possible.

If a subroutine would overlap the available core storage area the message

NN $\bar{X}$XXXX OVERLAP

is typed, where NN is the library subroutine number and $\bar{x}$xxxx is the length of the subroutine. The subroutine is not loaded.

If the LOCAL subprogram read-in routine or program linkage areas exceed the available core storage areas, the message

FLIPER $\bar{X}$XXXX OVERLAP

is typed. FLIPER is the name assigned to represent the read-in routine and $\bar{x}$xxxx is the length of the routine and linkage area required. The read-in routine and the linkage area are not loaded.

After all possible programs are loaded, and there is any error – overlap or others – the message

EXECUTION INHIBITED

is typed and a branch to MONCAL is executed. (MONCAL is the symbolic name for the entry point to the Monitor Control Record Analyzer routine.)

During loading of a FORTRAN program, the errors listed in Table 13 may appear.

Subroutine Error Checks

A number of error checks have been built into the library subroutines. The basic philosophy in the disposition of an error is to type an error message, set the result of the operation to the most reasonable value under the circumstances, and continue the program (note error D1 exception, described below). Subroutine error codes, the nature of the error, and the value of the result in FAC (symbolic name of the accumulator in which arithmetic operations are performed) are listed in Table 14.

Table 14. FORTRAN Subroutine Error Codes

ERROR CODE	ERROR	RESULT IN FAC
D1	Disk I/O used without a DEFINE DISK statement.	
D2	Logical record specified exceeds N2.	
D3	No group mark found at end of an array that was read from disk.	
E1	Overflow in FAD or FSB	$\bar{9}9......9\bar{9}9$
E2	Underflow in FAD or FSB	$\bar{0}0......0\bar{9}\bar{9}$
E3	Overflow in FMP	$\bar{9}9......9\bar{9}9$
E4	Underflow in FMP	$\bar{0}0......0\bar{9}\bar{9}$
E5	Overflow in FDV or FDVR	$\bar{9}9......9\bar{9}9$
E6	Underflow in FDV or FDVR	$\bar{0}0......0\bar{9}\bar{9}$
E7	Zero division in FDV or FDVR	$\bar{9}9......9\bar{9}9$
E8	Zero division in FXD or FXDR	$\bar{9}99.....$
E9	Overflow in FIX	$\bar{9}9......$
F1	Loss of all significance in FSIN or FCOS	$\bar{0}00.....0\bar{9}\bar{9}$
F2	Zero argument in FLN	$\bar{9}9......\bar{9}\bar{9}9$
F3	Negative argument in FLN	ln/x/
F4	Overflow in FEXP	$\bar{9}9......9\bar{9}9$
F5	Underflow in FEXP	$\bar{0}0......0\bar{9}\bar{9}$
F6	Negative argument in FAXB Negative argument in FSQR	$/A/^B$ SQR/x/
F7	Input data in incorrect form or outside the allowable range	
F8	Output data outside the allowable range	
F9	Input or output record longer than 80 or 87 characters (whichever is applicable to the I/O medium being used)	
G1	Zero to minus power in FIXI	$\bar{9}99.....$
G2	Fixed-point number to negative power in FIXI	$\bar{0}00.....$
G3	Overflow in FIXI	$\bar{9}99.....$
G4	Floating-point zero to negative power in FAXI	$\bar{9}9......9\bar{9}9$
G5	Overflow in FAXI	$\bar{9}9......9\bar{9}9$
G6	Underflow in FAXI	$\bar{0}0......0\bar{9}\bar{9}$
G7	Zero to negative power in FAXB	$\bar{9}9......9\bar{9}9$

The error printout is in the form

ER XX

where xx is the error code in the table.

If error D1 occurs, the machine halts, the typewriter carriage returns, and the operator must enter the DEFINE DISK statement parameters by means of the typewriter in the form of

NNXXXXX

where NN corresponds to N_1, and xxxxx corresponds to N_2 as described for the DEFINE DISK statement. Error D1 will be indicated until the values of N_1 and N_2 are within the correct range.

The FORTRAN loader further checks the value of N_2 (number of data records as specified in the DEFINE DISK statement) to see if the N_2 disk work area would be overlaid by operation of the FORTRAN loader. The FORTRAN loader uses the disk working area (starting from the high-order positions) for tables, COMMON save area, and LOCAL subprograms. Also. the first (low-order) 218 sectors of the disk work area are reserved to store the short form groups of the arithmetic and input/output subroutines. If N_2 *times* Record length *plus* 218 is greater than the lowest disk address used by the FORTRAN loader, N_2 will be redefined as

$$\frac{X-218}{\text{Record Length}}$$

where X is the lowest disk address used by the FORTRAN loader. The user is notified of this action by the following message:

MAX N2 ALLOWABLE $\bar{X}XXXX$

where $\bar{x}xxxx$ is the maximum allowable value for N_2 Loading and execution of the programs continues.

If Error D2 occurs, the specified record will not be written (or read), and the index value (I) may be incorrect.

If Error F7 occurs, the field which is incorrect is replaced by zeros, and processing continues.

The exponent portion of an E-type input data field must be right-justified in that field and may contain only one sign. Deviations from this rule are not checked. For exponents greater than 99 (absolute value), the value is reduced modulo 100.

If Error F8 occurs, the incorrect field is set to blanks in the output record, and an additional record is typed. This record contains the incorrect field in the form

```
E (f + 6). f     for floating-point numbers, and
I (k + 1)        for fixed-point numbers.
```

This additional record is also produced on the output unit (card punch, tape punch, or typewriter) called for by the source statement.

If Error F9 occurs, the incorrect field is ignored and processing continues. However, a remote possibility exists that part of the subroutines and the object program may have been destroyed by the abnormal record. In this case, the program may inexplicably halt at some later point in its execution.

Object Program Subroutine Linkage

The linkage generated by the FORTRAN II-D compiler is in the form

```
BTM SUBR, A
```

where SUBR is the name of the entry point for the subroutine and A is the address of the operand. For the relocatable library subroutines, an indirect address is used in the linkage. The actual address of the library subroutine entrance is stored at locations 02244-02248, 02249-02253, etc. The FORTRAN object program linkage for entrance to the first library subroutine will appear as,

```
BTM-02248, A
```

Adding Subroutines to the FORTRAN Library

The user may write library subroutines in SPS language and have them placed in the FORTRAN library. The subroutine must be assembled using the SPS II-D program, and may be loaded to disk storage at assembly time or at a later time using the Disk Utility program.

When the subroutine is loaded, a special DIM entry number which is reserved for FORTRAN library subroutines, must be used. This number is specified using the SPS control statement, ID NUMBER. Only 30 DIM entries are reserved, and, of these, the first 16 are in use when the system is delivered. The first 11 must not be replaced — they are required for correct operation of the FORTRAN system — but the user is free to replace any of the others if he desires.

If a subroutine has multiple entries, the first DIM entry will define the subroutine, but a DIM entry is required for each entry point, and no subroutine may have more than 9 entry points.

All entry points for a subroutine must be indicated in the following manner at the beginning of the source statements.

```
SUB      DSA    ENTRY1, ENTRY2, . . , ENTRY9
         DORG   SUB–4
ENTRY1
```

where ENTRY1 is the name of the first entry point, and ENTRY2 is the name of the second, etc.

The user must provide a 5-position area immediate-preceding each entry point. This space will be used to contain the address of the parameter for the subroutine when the subroutine is entered.

The symbolic name for each entry point must be specified in a DSA statement at the beginning of the SPS source program when it is assembled (even if only one entry point is desired). Also the *number* of entry points to the subroutine must be defined in a special DEND statement. The operand of the DEND statement must specify the number of entries to the subroutine.

Working Areas

In writing the subroutine, the programmer may first move the argument into one of the working areas such as FAC, BETA, or SAVE. In arithmetic subroutines, the exponent of a floating point result is usually stored in SAVE before being moved to FAC. A careful study of the arithmetic subroutines may reveal that the relocatable subroutine to be added can share the normalization, sign determination, overflow, underflow, and error typeout sections. The value calculated by the subroutine must be left in FAC. Even if no value is calculated, it is advisable to place a constant in FAC.

When programming a subroutine with variable length floating-point numbers, it may be necessary to use certain addresses and constants available in the arithmetic and input/output subroutines. A reference to the listings of these subroutines will yield the information on these addresses and constants. As the mode of operation (fixed or floating point) is determined by the argument of the subroutine, the FORTRAN II-D Processor does not distinguish fixed-point from floating-point subroutines. It is up to the user to have a thorough knowledge of the added subroutines and to use them correctly.

Loading the Library Subroutine

The Disk Utility Program can be utilized to add a subroutine to the FORTRAN library or it can be added at assembly time.

An example of the control records required for adding a subroutine directly to the FORTRAN library at assembly time follows:

```
‡‡ SPS
* LIBR
* NAME HCOS                         Monitor and
* ASSEMBLE RELOCATABLE              SPS control
* STORE RELOADABLE                    records
* ID NUMBER 0026

SUB    DSA    HCOS, HSIN
       DORG SUB–4
       .
       .                            The
       .                            User-
HCOS   OP                           written
       .                            library
       .                            function
HSIN   OP                           SPS
       .                            instructions
       .
       DEND 2
```

The Disk Utility Program can be utilized to load the subroutine to the library in which case the NAME, STORE RELOADABLE and ID NUMBER statements can be omitted, but an OUTPUT CARD or OUTPUT PAPER TAPE statement would have to be included.

The DLOAD (or DREPL) Control record must contain the information as shown in the following example:

```
*DLOAD HCOS 0026 0101200002 C I
Col.   7    17   39   44    49 50
```

Columns	1-6	Code word *DLOAD.
	7-12	Alpha name of program to be used in FORTRAN arithmetic statements.
	17-20	DIM entry number.
	39-43	The length of the subroutine. This number must be even.
	44-48	The number of entry points.
	49	Input unit (P for paper tape).
	50	Core image format.

Other options, such as read-only protection, are available if they are desired (see DLOAD CONTROL RECORD in the Disk Utility Program section of this manual).

Additional Entries and Synonyms

A DFLIB Control record must be entered if the subroutine contains more than one entry point or if one entry point is to be called by more than one name. The format for the DFLIB Control record to add the second entry for the preceding examples is as follows:

Columns	1-6	*DFLIB
	7-12	HSINbb
	13	Not used
	14-15	27
	16-80	Not used

where HSIN is the user-assigned name, left-justified, of the subroutine being added, and 27 is the next consecutive DIM entry number, after the DIM entry number used for the original entry. The DIM entry number must be between 21 and 39 and must correspond with the sequential position of the entry as it is written in the operands of the DSA statements in the source program. As delivered, the system already makes use of DIM entry numbers 10 through 25 for FORTRAN library subroutines. However, the last 5 may be removed, if desired. *If no subroutines are removed from the FORTRAN library set, the available DIM entry numbers for additional library subroutines are 26 through 39.*

The sequence of input to assemble and load a subroutine is shown in Figure 17.

FORTRAN Subprogram Written in SPS

The FORTRAN user is able to create subprograms using SPS and have these subprograms available immediately for call by FORTRAN programs. To accomplish this he must follow the writing specifications outlined here.

The Indicator Record

Each subprogram to be called by a FORTRAN program must contain a header record to identify the routine and to provide other essential information. The SPS instructions necessary to create this record are shown below:

```
S  DS      ,*+101
   DC      6, 987898, 5-S
   DAC     6,NAMEbb, 7-S
   DVLC    22-S, 5, LAST, 2, ff, 2, kk, 5, Entry
             Address-6, 5, 0, 30, 0
   DSC     17, 0, 0
   DORG    S-100
```

where LAST is the address of a ‡ or the first digit of a subprogram name in the call list (see CALLING OTHER SUBROUTINES).

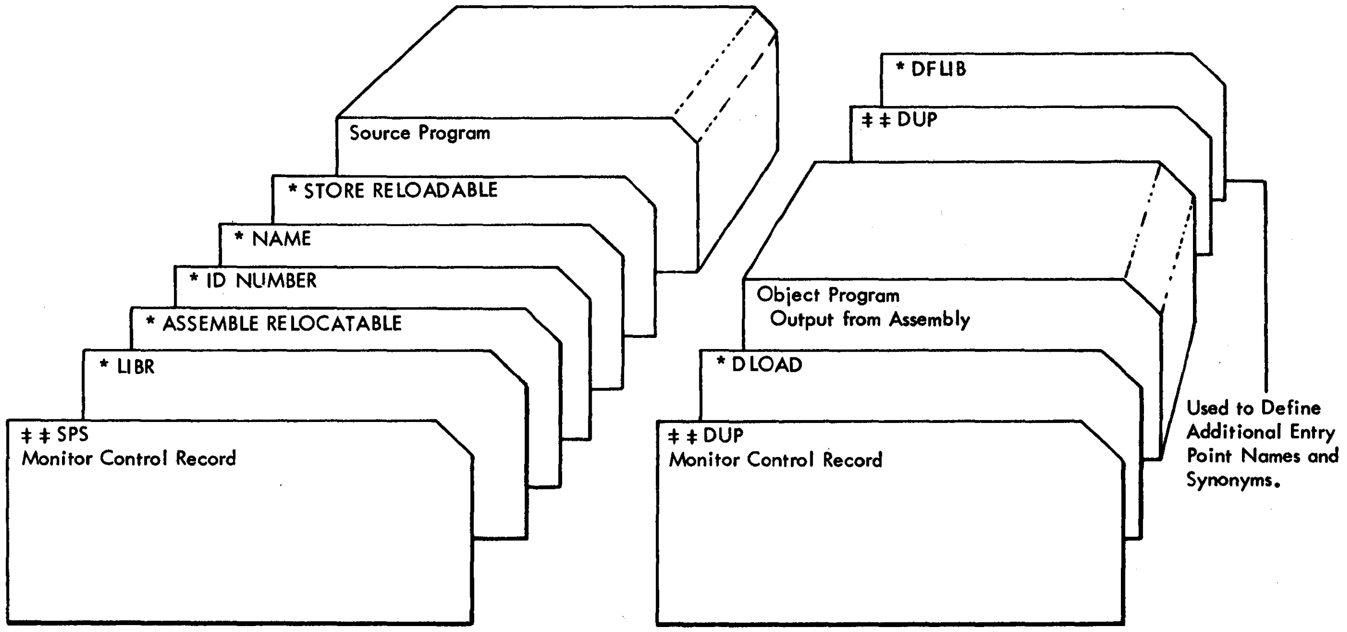


Figure 17. Adding Subroutines for FORTRAN Subroutine Library

Calling Library Subroutines

If the subprogram or function is to call FORTRAN library subroutines, the user must write out the DVLC operand that contains the 30 zeros in the example. Those positions that correspond to specific subroutines must contain a one instead of a zero. The correspondence between the positions in this field and the standard library subroutines are presented in Table 11 so that the user may select any subroutine in the library. To effect a transfer to any library subroutine from the SPS written subprogram, the user must write the following instruction; BTM-SUBR, PARAM, where PARAM is the address of the parameter required by the subroutine and -SUBR is the address for the subroutine entry shown in Table 11. The subroutine entry address must be indirect.

Calling Other Subroutines

If the user wishes to call other subprograms from an SPS written subprogram, he may do so. To do this the user must code:

```
BTM  NAMESP,  *+11
DSA  A,B, . . . , Z
```

for each transfer to another subprogram, and must include the name of the subprogram called. Each named subprogram must have a 5-digit address field reserved in the subroutine. The names of the subprograms to be called must be placed, in double digit form, at the end of the subprogram along with the addresses of the reserved address fields within the subprogram. A record mark must follow the last subprogram name and address. The length of the program, which is specified using LAST in the previous example, must not include the names and addresses that are at the end of the subprogram. These names and addresses must follow the last location of the subprogram. They can be coded as shown below:

```
DAC    6, NAM1bb,
DVLC   , 5, N1LOC
DAC    6, NAM2bb,
DVLC   , 5, N2LOC
DC     2, 0 @
```

This coding will cause the two subprograms named NAM1 and NAM2 to be called out and made available when the SPS routine that requires them is loaded.

where N1LOC is the low-order address of the P field of the BTM instruction that calls this subprogram. For example;

```
BTM  XXXXX,  *+11
N1LOC————↑
```

The SPS instructions corresponding to a CALL SUBP (A, B, C, . . . N) are:

```
        BTM   SUBP, *+11
        DSA   A, B, C, . . . N
```

The entry address, SUBP, is determined when the program is loaded, so it need not be included in the BTM instruction, i.e., BTM, *+11.

Writing FORTRAN Subprograms in SPS

In addition to the header record described above, linkage to obtain the subprogram parameters must be included in the subprogram. If no parameters are needed, or if the subprogram knows the location of the parameters, the user writes:

```
         DC 5,0
SUBNAM   AM SUBNAM-1,1,10   )
                            }   The
                            }  subprogram
          B  SUBNAM-1, , 6  )
          DC  1, @
```

If one parameter is needed, and this subprogram is never nested within any other function or subprogram, the user writes:

```
         DC 5, 0
SUBNAM   AM SUBNAM-1, 5, 10
         TF  INSUB, SUBNAM-1, 11
         AM SUBNAM-1, 2     )
                            }   The
                            }  subprogram
          B  SUBNAM-1, , 6  )
          DC  1, @
```

If several parameters are to be moved to the subprogram, a loop may be utilized to conserve core storage. The parameters must be stored in the subprogram in consecutive order. An example of the coding to accomplish this for three parameters is shown below:

```
INSUB   DSA    0, 0, 0
        DC     1, @
        DC     5, 0
SUB     TFM    TF+6, INSUB-4
        AM     TF+6, 4, 10
        AM     SUB-1, 5, 10          (1)
        TF     CF+11, SUB-1, 11      (2)
        BNF    *+36, CF+11           (3)
CF      CF     CF+11                 (4)
        TF     CF+11, CF+11, 11      (5)
TF      TF     INSUB, CF+11          (6)
        AM     TF+6, 1, 10
        BNR    SUB+12, TF+6, 11
        AM     SUB-1, 2, 10          (7)

        B      SUB-1, , 6            (8)
        DC     1, @
```

The instructions that constitute the body of the subprogram are placed between number 7 and 8 above. Instruction 7 must add "two" if the number of parameters is an odd number, or "one" if this number is even. The record mark must be in the first even location following the subprogram.

The instruction numbered 8 (B SUB-1, , 6) returns control to the calling program. When writing subprograms in SPS the user must place this instruction at every point that a return is required.

Writing FORTRAN FUNCTION Subprograms in SPS

The user must add one instruction to the set required for SUBROUTINE subprograms written in SPS in order to write FORTRAN FUNCTION subprograms (subroutines normally produced by FORTRAN statements preceded by a FUNCTION statement). This instruction is BTM TOFAC, ANS. TOFAC is the entry point of a FORTRAN system routine that will put the value at the address ANS into FAC, the FORTRAN accumulator. In this instruction ANS is the location where the result of the subroutine calculation is stored. This instruction must precede each instruction that returns control to the calling program (e.g., the BTM instruction must precede the number (8) instruction in the preceding example). Each time a return is desired, this BTM instruction must be repeated. The actual address of TOFAC is constant and is available in the FORTRAN II-D listing.

Disk Storage Location of the FORTRAN Compiler

The FORTRAN compiler and operating system is an integrated part of the Monitor I system. It is possible, however, to eliminate the programs that constitute the FORTRAN portion of the Monitor I, and still utilize the remainder of the system. It is also possible to delete specific FORTRAN library subroutines and to utilize the remainder of the FORTRAN system. The procedure to follow in order to delete any program from the disk is described in the DUP section. The DIM entry numbers that may be specified for deletion are shown below.

	DIM No.	*Disk Location*
FORTRAN Compiler	136, 137, 153, 156	Cylinder 86, 87, 88, 89, 90
FORTRAN Subprogram Loader	138, 147, 149, 150, 152, 157	Cylinder 80, 84
FORTRAN Arithmetic and I/O	144, 145, 146	Cylinder 84, 85
FORTRAN Library Subroutines	*10–39	Cylinder 81

If the FORTRAN system is deleted using the DUP routines, the portion of the disk which it occupies will become available for assignment of other user written programs.

*When the system is delivered, 10 through 25 are in use. The symbolic names of the subroutines that may be deleted are COSF, SINF, ATAN, SQRTF, and ABSF. The other subroutines may be deleted only if the complete FORTRAN system is deleted.

IBM 1620-1443 Monitor I System

A printer-oriented Monitor I System is available for 1620 Systems that are equipped with an IBM 1443 Printer. This system is an extension of the standard Monitor I System described in the preceding pages of this publication. Only the differences between the standard system and the printer-oriented system are described here. Specifications and operating procedures pertaining to the standard system are valid for the printer-oriented system if no specific mention is made of them in this section.

General Description

The 1620-1443 Monitor I System requires that an IBM printer be included in the 1620 system configuration. Of course, any other applicable input/output units, special features, etc., may also be attached to the system.

The principle advantage of a printer-oriented system is that it provides a convenient and relatively fast means of obtaining listings of assembled or compiled programs, symbol tables, and any other data that might be desired. The component programs of the Monitor I System have been modified to make the best possible use of the printer. The console typewriter is used only for messages which must be acted upon immediately by the operator.

The following paragraphs were written under the assumption that the reader has a practicable knowledge of the IBM 1443 Printer. Those without such knowledge should read the publication, IBM *1443 On-Line Printer for 1620/1710 Systems* (Form A26-5730).

Supervisor Program

The I/O routine of the Supervisor Program has been modified to handle printer output for both SPS and FORTRAN object programs. This was accomplished without changing entry points, linkages, or core storage requirements. The language used to gain access to the I/O routine is described in the respective SPS and FORTRAN portions of this supplement.

Printer Errors

If a printer error (indicator code 25) occurs, the message

PRT ERR XXXXX

is typed out, and control is returned to core address xxxxx in the calling program. In addition, the error is recorded in a printer error counter which has been added to the error counters used in the standard system. Whenever the error indicators or error counters are typed out, they will be in the following sequence:

$\bar{0}6\bar{0}7\bar{1}6\bar{1}7\bar{2}5\bar{3}6\bar{3}7\bar{3}8$

The only change is the addition of the Printer Check indicator (code 25).

NOTE: If a halt occurs at address 00467, either with or without a prior error message, the program can be resumed by pressing the Reset and Start keys.

Programming Considerations

1. The Printer Busy indicator (code 35) is not tested by the I/O routine; therefore, if the test is desired, the user must perform it before executing a call to the printer.
2. Carriage control operations must be handled in the user's program.

Disk Utility Program

Disk-to-Output Routine

The Disk-to-Output routine has been modified to use the printer for all output that formerly was typed out on the console typewriter. To specify the printer as an output unit, the user must punch the letter L in column 17 of the DDUMP control card.

The format of all printed output (in this routine) will be 100 characters (1 sector) per line, with the exception of the availability list and the equivalance table which will retain the format that was used in the standard system.

Define Parameters Routine

The define parameters routine in this system allows the FORTRAN II-D to be set for either 120 or 144 print position usage of the 1443 Printer during compilation and execution. The user may either put a "zero" in column 59 of the *DEFINE record for 120 print position operation or a "one" in that column for 144 positions. This indicator is set in disk sector positon 65 of the Communications Area and is a "zero" when the system is delivered.

SPS II-D

The SPS II-D assembly program has been modified to include twelve printer-oriented imperative mnemonics, two printer declarative mnemonics, and three new SPS control statements.

Printer Imperative Mnemonics

The imperative mnemonics included in the printer-oriented Monitor I System are listed in Table 14.1. Also shown are the actual OP codes generated and the P and Q address functions. Table 14.2 shows the Q-address modifiers that are generated in the object program.

Some examples of printer statements are shown below:

Label	Operation	Operands & Remarks
	PRN	DATA,,,PRINT NUMERICALLY
	PRNS	DATA,,,PRINT NUMERICALLY AND SUPPRESS SPACING
	SKIP	,2,,SKIP IMMEDIATE TO CARRIAGE CHANNEL 2
	SKAP	,5,,SKIP AFTER PRINTING TO CARRIAGE CHANNEL 5
	SPIM	,3,,MOVE CARRIAGE 3 SPACES,IMMEDIATE
	SPAP	,3,,MOVE CARRIAGE 3 SPACES AFTER PRINTING

Table 14.1 Imperative Printer Mnemonics

	OPERATION CODE		OPERANDS	
OPERATION	MNEMONIC	ACTUAL	P ADDRESS	Q ADDRESS
Printer Dump	PRD	35	Storage address from which leftmost (first) numerical character is written	None Required
Printer Dump and Suppress Spacing	PRDS	35	Same as Printer Dump	None Required
Print Numerically	PRN	38	Same as Printer Dump	None Required
Print Numerically and Suppress Spacing	PRNS	38	Same as Frinter Dump	None Required
Print Alphamerically	PRA	39	Storage address from which leftmost Alphameric character is written (odd numbered address)	None Required
Print Alphamerically and Suppress Spacing	PRAS	39	Same as Print Alphamerically	None Required
Skip Immediate	SKIP	34	Not used	Control Code
Skip after Printing	SKAP	34	Not used	Control Code
Space Immediate	SPIM	34	Not used	Control Code
Space after Printing	SPAP	34	Not used	Control Code
Branch on channel 9	BCH9	46	Address branched to if indicator 33 is on. This indicator is turned on by the detection of a hole in channel 9 of the carriage tape.	None Required
Branch on channel 12	BCOV	46	Address branched to if indicator 34 is on. This indicator is turned on by the detection of a hole in channel 12 of the carriage tape.	None Required

In these examples, the operand DATA represents the core storage address of the data to be printed; the numerical operands (2, 5, 3, 3) are either channel numbers for skip operations or the number of spaces for space operations. Table 14.3 shows the Q operands to be placed in the skip and space statements for all possible skip and space operations.

Printer Declarative Statements

Two printer declarative mnemonics are included in the printer-oriented Monitor System. Descriptions of the mnemonics follow together with the two-digit code that is generated for the I/O constant. Note the two forms of each declarative.

Mnemonic	Code	Name
DPRN	$\bar{1}4$	Define Printer Numerical
DPRN, , S	$\bar{2}2$	Define Printer Numerical – Suppress Spacing
DPRA	$\bar{1}8$	Define Printer Alphameric
DPRA, , S	$\bar{2}6$	Define Printer Alphameric – Suppress Spacing

These declarative operations generate an I/O constant in the object program which can be used by a PUT I/O macro-instruction to print data under control of the I/O routine. When used in the source program,

Table 14.2 OP Codes and Q Modifiers Generated for Printer Mnemonics

MNEMONIC	OPERATION	OP CODE	Q MODIFIERS			
			Q_8	Q_9	Q_{10}	Q_{11}
PRD	Printer Dump	35	0	9		0
PRDS	Printer Dump and Suppress Spacing	35	0	9		1
PRN	Print Numerically	38	0	9	Not used, will be zero	0
PRNS	Print Numerically and Suppress Spacing	38	0	9		1
PRA	Print Alphamerically	39	0	9		0
PRAS	Print Alphamerically and Suppress Spacing	39	0	9		1
SKIP	Skip Immediate	34	0	9	*	
SKAP	Skip After Printing	34	0	9	*	
SPIM	Space Immediate	34	0	9	*	
SPAP	Space After Printing	34	0	9	*	

*Modifiers which specify the particular skip or space operation (see Table 14.3). For a detailed description of these modifiers, refer to the publication entitled IBM 1443 On-Line Printer for 1620/1710 Systems (Form A26-5730).

Table 14.3 Q Operands and Q Modifiers for Skip and Space Operations

CONTROL CODES		ACTUAL Q_{10}, Q_{11} MODIFIERS	
		IMMEDIATE	AFTER PRINTING (DELAY)
Skip to Channel	1	71	41
	2	72	42
	3	73	43
	4	74	44
	5	75	45
	6	76	46
	7	77	47
	8	78	48
	9	79	49
	10	70	40
	11	33	03
	12	34	04
Number of Spaces (SPIM or SPAP)	1	51	21
	2	52	62
	3	53	63

they identify the output record area. Three operands are required for each statement. The first operand is used to specify the address where the I/O constant is to be loaded into core storage. This operand may be an absolute value or a symbolic name. If a symbolic name is used, the symbol must previously have been defined as an absolute value. If the operand is omitted, the processor will assign the address to which the constant will be loaded in core storage.

The second operand, which may be symbolic or actual, is the address of the output record area. This address will be included in the I/O constant in the object program. The third operand may be a letter S or it may be omitted. If the letter S is present, the automatic single space after printing will be suppressed whenever the associated output record is printed. Remarks are permitted following the third operand.

If a label is included with this statement, the storage address assigned to it will be that of the leftmost position of the generated I/O constant.

The statements which follow show how a DPRA declarative statement is used with a PUT I/O macro-statement to print a 110-alphameric character record under control of the Supervisor I/O routine. In this example, the first two statements define the output area where the record is stored.

Label	Operation	Operands & Remarks
RECORD	DAS	1,10,,DEFINE ALPHAMERIC OUTPUT RECORD
	DAC	1,@,,MUST FOLLOW WITH RECORD MARK
PRNTR	DPRA	,RECORD,,IDENTIFY PRINTER OUTPUT RECORD
	PUT	PRNTR,,WRITE OUTPUT REC AS SPECIFIED BY DPRA

The declarative statements are usually written preceding or following the program; however, the macro-statement is entered in the program at the point where printing is to take place.

Printer Control Records

In the printer-oriented Monitor I System, there are three SPS control records that pertain to printer operations. The first two described here are modifications of SPS control records used in the standard system; the third is a new control record.

**LIST PRINTER* – This record replaces the LIST TYPEWRITER record used in the standard system. It causes a listing to be printed by the printer during assembly.

The format of the listing is as follows: page and line number, label, op code, operands, remarks, core location, and object instruction.

**PRINT SYMBOL TABLE* – This record replaces the TYPE SYMBOL TABLE record used in the standard system. It causes the symbol table to be printed by the printer following assembly.

***XXXXXXXXXXX* – This record is used to print a heading above listings and/or symbol table printouts. The data (signified by Xs above) that follows the two asterisks is printed at the top of the respective printouts. When this record is typed out during assembly, it is shown as an Identification record with the code (ID) typed to the left of the two asterisks. This is also true of all Identification records used in the printer-oriented system; that is, the code (ID) is typed to the left of the record instead of to the right as in the standard system.

IBM-Defined System Symbols

In the printer-oriented system, the following system symbols are available to the user. Notice that the symbols are the same as those used in the standard system, but some of the equivalences are changed.

Symbol	Equivalence	Description
9RCYL0	00513	These are the low-order positions of four 2-digit fields which contain the numbers of cylinders (00-99) where the disk access arm is repositioned *after* a disk operation in which a reposition has been requested. The four fields refer to drives 0, 1, 2, 3, respectively.
9RCYL1	00515	
9RCYL2	00517	
9RCYL3	00519	
9CCYL0	02132	These are the low-order positions of four 2-digit fields, similar to the previous four. However, these positions contain the cylinder numbers of the *current* access arm positions (the position of the arm after the last disk IORT operation).
9CCYL1	02134	
9CCYL2	02136	
9CCYL3	02138	

Assembly Errors

In addition to the SPS II-D error codes and descriptions listed in a previous section of this manual, the following error conditions will cause an error typeout:

1. A space specification that is either 0 or greater than 3
2. A skip specification that is either 0 or greater than 12

Either of the above conditions will be indicated by the error code ER5.

If no Error Stop control record is included in the assembly, the processor will cause an erroneous space or skip specification to be set to 1.

Error messages will appear on the typewriter, not on the printer.

FORTRAN II-D

The FORTRAN II-D compiler has been modified to take advantage of the printer while compiling object programs. Although the basic FORTRAN language remains unchanged, the specifications of several FORTRAN output statements have been modified. Also, the printer has replaced the console typewriter as the basic printed-output medium.

Language

PRINT – In the printer-oriented system, a PRINT statement is used to transfer data to the 1443 Printer. A TYPE statement is still used to transfer data to the console typewriter.

FORMAT – A FORMAT statement, when used in conjunction with a PRINT statement, can provide for up to 120 characters for each printed line (or 144 characters if the system has been properly defined). A FORMAT statement used with a TYPE statement is still limited to 87 characters for each typed line.

In addition to the normal function of a FORMAT statement, there is another function that it must perform when used with a PRINT statement. This function consists of designating the desired space or skip operation for each printed line. A printer-oriented FORMAT statement must begin with 1H followed by a control character which specifies the desired operation. The control characters and their effects are:

blank	– single space before printing
0	– double space before printing
1-9	– immediate skip to channels 1-9

The control character itself does not become part of the printed output.

EXAMPLE

```
      PRINT 2, A, B, J
    2 FORMAT (1H0, F8.2, F8.2, I8)
```

This specification will provide a double space between the line being printed and the previous printed line.

NOTE: The control carriage specification is applicable to the first line of print only. If more than one line is called for, the user must be sure that the carriage control specification precedes the normal specifications for each line of print. For example:

```
    2 FORMAT (1H0, F8.2/1H0, E14.6)
```

Control Records

The five new control records that have been added to FORTRAN II-D for the printer-oriented Monitor System are:

```
 * LIST PRINTER
 * ARITHMETIC TRACE
 * IF TRACE
 * ALL STATEMENT MAP
** XXXXXXXXX
```

The first four in the list are compile-time options previously available through console switch settings. The last record is a means of obtaining a heading at the top of each printed page.

The format of these records is the same as described in the section entitled FORTRAN II-D CONTROL RECORDS.

- * *LIST PRINTER* – This record will cause program listings and symbol table output to be printed.
- * *ARITHMETIC TRACE* – This record will cause trace instructions to be compiled for arithmetic statements.
- * *IF TRACE* – This record will cause trace instructions to be compiled for the purpose of tracing the value of the expression generated in an IF statement.
- * *ALL STATEMENT MAP* – This record will cause the address of the first instruction generated for each statement to be printed.
- ** *XXXXXXXXX* – This record will cause a heading to be printed at the top of each printed page. The Xs represent the heading. Up to 78 characters may be specified.

Listings and Symbol Table Output

Listings and symbol table output will appear on the printer instead of the console typewriter. The control records used to obtain these outputs are described above. The formats of the outputs have been changed to take advantage of the characteristics of the printer. The loader map also appears on the printer.

Error Messages

All error messages will appear on the printer except loader error messages; in addition, error message D1 will also appear on the console typewriter. Instructions to the operator, for example, LOAD SUBNAM, will appear on the console typewriter.

Subroutine error code F9 has been modified to include any printer records that exceed 120 characters (or 144 characters if the system has been so defined).

When a phase 2 error is detected, an error message in one of the following forms is printed on the 1443 printer.

```
SSSS+CCCC SYMBOL TABLE FULL
SSSS+CCCC IMPROPER DO NESTING
SSSS+CCCC DO TABLE FULL
SSSS+CCCC MIXED MODE
```

where ssss is the last statement number, encountered by the program prior to the error, and cccc is the number of statements following the last numbered statement. ssss + cccc is the statement that contains the error. Comment cards, blank cards, and continuation cards are not counted in the statement count.

Trace Routine

If the trace routine is used, its output will appear on the printer. If the 144 print position subroutines are used, floating variables will appear in E-type format and fixed variables in I-type format.

Carriage Control Tape

The carriage control tape, when used for assembling and compiling, should be punched in channel 1 to indicate the beginning of a page, and punched in channel 12 to indicate the end of a page. When the program senses the hole in channel 12, it automatically executes a skip to the channel 1 hole, which indicates the beginning of the next page.

Arithmetic and I/O Subroutines

The arithmetic and I/O subroutine sets used in the 1620-1443 Monitor I System are longer than the sets used in the standard 1620 Monitor I System. This means that the starting addresses of FORTRAN object programs are higher in the printer-oriented system, as shown below:

	Standard System	*120 Printer System*	*144 Printer System*
Using in-core arithmetic and I/O subroutines	14000	14300	14600
Using out-of-core arithmetic and I/O subroutines	07500	07800	08100

Furthermore, the subroutine entry points for both the in-core and out-of-core subroutines used with the printer system are different from those in the standard system.

Therefore, FORTRAN object programs compiled by the 1620 Monitor I System must be recompiled if they are to be used by the 1620-1443 Monitor I System.

However, programs compiled for 1620-1443 Monitor I version 1 (120 print positions) do not have to be recompiled to operate on 1620-1443 Monitor I version 2 (144 print positions).

Eight subroutine sets are available with the Monitor I version 2 System. They are identified as follows:

OPTION A – 120 print position
- Set 1 short form – without floating point feature
- Set 2 long form – without floating point feature
- Set 3 short form – with floating point feature
- Set 4 long form – with floating point feature

OPTION B – 144 print position
- Set 1 short form – without floating point
- Set 2 long form – without floating point feature
- Set 3 short form – with floating point feature
- Set 4 long form – with floating point feature

Only the floating point sets or the non-floating point sets of a given option are loaded to the disk at any one time. For example, a user who has a 120 print position printer and has the automatic floating point feature would load only OPTION A, sets 3 and 4 on his disk pack.

Monitor I System Loader

This program is used initially to load the Monitor I System from cards or paper tape into disk storage. Cards contain 75 columns of data followed by a 5-position sequence number. Sequence numbers are not present with tape data.

The system to be loaded, in card or paper tape form, is comprised of several blocks of data, each with a unique deck number, a Heading Control record, and a 9's trailer record. With this arrangement it is possible to load each new block of data to a different area of disk storage as specified by the Heading Control records. For card input, the cards within a data block must be consecutively numbered in ascending sequential order.

The combined input data, i.e., all data blocks, must be preceded by the Loader Program itself. This program is contained in approximately forty cards. If the sequence of the first four cards is inadvertently altered, the program may not operate correctly. The loader program is deck number 00, columns 30-31. All input cards, with the exception of the first four cards of deck 00, are sequence checked by the loader.

Card Formats

Heading Control

Columns	1	Asterisk (*).
	2-7	Code word, LDCNTR.
	9-14	Name of data block (program, table, etc.) to follow.
	16-21	Address of first sector to be loaded.
	23-28	Address of last sector to be loaded.
	30-32	Deck number. This number, combined with the two positions 79-80, constitutes the sequence number. Blanks are interpreted as zeros. Therefore, the number 55 and a blank in columns 30-32 are interpreted as sequence number 55000. The first card of the data block must then begin with the sequence number 55001 in columns 76-80.

Data

Columns	1-75	Data to be loaded to disk storage.
	76-80	Sequence number.

Trailer

Columns	1-5	$\bar{9}9999$
	6	ǂ
	7-8	00

Operating Procedures

Switches

The Parity, I/O and O'FLOW check switches should be in the PROGRAM position for either card or tape loading. For card or tape input, the program will halt after each trailer card if Program Switch 1 is off. If the switch is on, all data blocks are loaded without stopping the computer. Therefore, the user can stack input, if desired.

Paper Tape Loading

1. Ready the paper tape reader with the Loader tape reel.
2. Enter 36 00000 00300 from the typewriter.
3. Depress the Release and Start keys.
4. Ready the tape reader with the Data tape reel.
5. Depress the 1620 Start key.
6. Return to step 4 to load successive Data tapes.

NOTE: When loading with Switch 1 on, the Loader will continue to read more data after each data group has been loaded. Therefore, several such data input groups may be present on one input reel.

Card Loading

1. Ready the card reader with deck number 00, Loader Program. The remaining decks may be stacked behind deck 00 as explained under SWITCHES.
2. Depress the 1622 Load key.

Messages

FOR BOTH PAPER TAPE AND CARD LOADERS

Message/Cause/Operation Action

AAAAAA LOADED FROM F̄FFFFF TO L̄LLLLL, where AAAAAA is the name of a data block, F̄FFFFF is the address of the first sector loaded, and L̄LLLLL is the address of the last sector loaded. This message will type following each successful deck loading. If Program Switch 1 is on, it is an indication to the operator to load the next deck.

DISK RD WR ERROR, START TO RETRY. This message will type if a disk write error occurs that cannot be corrected by one automatic retry. Depressing the Start key will cause the write operation to be retried twice. If the error is not corrected by the retries, the message will again be typed.

RDER. This message will type if a paper tape or card reading error occurs. To correct the error, ready the reader with the corrected record and depress the Start key. (An error card will be located next to the last card in the stacker when a halt occurs for a card reading error.)

CONTROL STATEMENT INVALID, RE-ENTER. This message will type if any of the following conditions are encountered in Heading Control record data.

1. A misspelled code word.
2. A record mark in column 6.
3. First sector to be loaded is greater than last sector to be loaded. The user must supply a corrected control record and depress the 1620 Start key.

FOR CARD LOADER ONLY

Message/Cause/Operator Action

SEQ. This message will type and the program will halt if any of the cards in the loader program, with the exception of the first four cards, is out of sequence. To resume loading, (1) restore the cards to their correct sequence and place them in the card hopper, (2) depress the Start keys on both the card reader and 1620 console.

N̄NNNN CARD SEQ ERROR, CORRECT AND START, where N̄NNNN is the sequence number of the first data card out of consecutive ascending sequence. After the message is typed, the program will halt. To restart the computer, (1) restore the sequence of data cards, starting with the card in error, (2) place the resequenced cards in the card read hopper, (3) depress the Start keys on both the card reader and 1620 console.

NO TRAILER REC. CORRECT, RE-LOAD COMPLETE DECK WITH CNTR REC, AND BR TO 7404. This message will type and the program will halt if a 9's trailer record is missing following any data block. To restart the computer, the user should (1) restack the cards in the card reader so as to restart card reading with the Header card of the data block which had the missing trailer record. (2) Depress the Reset and Insert keys. (3) Enter 49 07404 from the typewriter. (4) Depress the Release and Start keys. (5) Depress the card reader Start key.

TRAILER CARD SEQ ERROR, CORRECT AND START. This message will be typed if the sequence number on the trailer card is incorrect. The procedure for restarting as the same is for any other card sequence error.

Appendix A

SPS Tables

Table 15. Summary of SPS Declarative Operations

NOTE: Except for the constants in DC, DSC, and DAC, all operands may be actual or symbolic. All symbolic length and address operands must be previously defined. All operands may use address adjustment. Remarks may follow operands except in DSA and DVLC statements. "Alpha Record Address" in the table refers to the leftmost position plus one of an alphameric field, whereas "Field Address" refers to the rightmost position of a field. The term "Numerical Record Address" refers to the leftmost position of a field.

DECLARATIVE STATEMENT FORMAT			AMOUNT ADDED TO LOCATION ASSIGNMENT COUNTER IF ADDRESS (A) IS BLANK	VALUE STORED IN SYMBOL TABLE AS EQUIVALENT TO "SYMBOL"	DATA FIELDS WHICH ARE LOADED AS A PART OF THE OBJECT PROGRAM
LABEL	OP CODE	OPERANDS			
SYM	DS	L,A	L (length). If L is blank, 0 is added.	A address. If A is blank, the field address from the location assignment counter is stored.	None.
SYM	DSS	L,A	L (length). If L is blank, 0 is added.	A address. If A is blank, the numerical record address from the location assignment counter is stored.	None.
SYM	DAS	L,A	2 x L is added. If L is blank, 0 is added.	A address must be odd. If A is blank, the alpha record address from the location assignment counter is stored.	None.
SYM	DC	L,C,A	L is added.	A address. If A is blank, the field address from the location assignment counter is stored.	C, the (numerical) constant .
SYM	DSC	L,C,A	L is added.	A address. If A is blank the numerical record address from the location assignment counter is stored.	C, the (numerical) constant.
SYM	DVLC	A,L,C, L,C, etc.	L is added.	First C address.	C, C, etc., the (numerical) constants.
SYM	DAC	L,C,A	2 x L is added.	A address must be odd. If A is blank, the alpha record address from the location assignment counter is stored.	C, the (alphameric) constant.
SYM	DSA	D,E,F,G, H, I, J, K, L,M	5 x (number of addresses) is added.	Field address of the first address on list.	A list of the actual addresses that correspond to D,E,F, etc.
SYM	DSB	L,N,A	Length of each element times the number of elements is added.	A address. If A is blank, field address of the first element is stored.	None.
SYM	DNB	L,A	L is added.	A address. If A is blank, the field address from the location assignment counter is stored.	Number of blank characters that equal L.
SYM	DDA	A,D,F,S, M	14, length of a disk control field.	(Same as DSC) .	D,F,S,M.
SYM	DGM	A	1	A address or location counter.	ǂ (Group Mark) .

Table 16. Summary of SPS Arithmetic Instructions

NOTE: Indirect Addressing (special feature) is allowable with all P address operands listed below. An * to the left of the Q operand indicates that this feature may be used with it.

OPERATION	OPERATION CODE		OPERANDS	
	MNEMONIC	ACTUAL	P ADDRESS	Q ADDRESS
Add	A	21	Storage address of units position of augend	*Storage address of units position of addend
Add Immediate	AM	11	Same as code 21	Q_{11} of instruction is units position of addend
Subtract	S	22	Storage address of units position of minuend	*Storage address of units position of subtrahend
Subtract Immediate	SM	12	Same as code 22	Q_{11} of instruction is units position of subtrahend
Multiply	M	23	Storage address of units position of multiplicand	*Storage address of units position of multiplier
Multiply	MM	13	Same as code 23	Q_{11} of instruction is units position of multiplier
Load Dividend (special feature)	LD	28	Storage address in product area to which units position of field (dividend) is to be transmitted	*Storage address of units position of dividend
Load Dividend Immediate (special feature)	LDM	18	Same as code 28	Q_{11} of instruction is units position of dividend
Divide (special feature)	D	29	Storage address at which first subtraction of the divisor occurs	*Storage address of units position of divisor
Divide Immediate (special feature)	DM	19	Same as code 29	Q_{11} of instruction is units position of divisor
Floating Add (special feature)	FADD	01	Storage address of units position of exponent of augend	*Storage address of units position of exponent of addend
Floating Subtract (special feature)	FSUB	02	Storage address of units position of exponent of minuend	*Storage address of units position of exponent of subtrahend
Floating Multiply (special feature)	FMUL	03	Storage address of units position of exponent of multiplicand	*Storage address of units position of exponent of multiplier
Floating Divide (special feature)	FDIV	09	Storage address of units position of exponent of dividend	*Storage address of units position of exponent of divisor

Table 17. Summary of SPS Internal Data Transmission Instructions

NOTE: Indirect Addressing (special feature) is allowable with all P address operands listed below. An * to the left of the Q address operand indicates that this feature may be used with it.

OPERATION	OPERATION CODES		OPERANDS	
	MNEMONIC	ACTUAL	P ADDRESS	Q ADDRESS
Transmit Digit	TD	25	Storage address to which single digit is transmitted	*Storage address of single digit to be transmitted
Transmit Digit Immediate	TDM	15	Same as code 25	Q_{11} of instruction is the single digit to be transmitted
Transmit Field	TF	26	Storage address to which units position of field is transmitted	*Storage address of units position of field to be transmitted
Transmit Field Immediate	TFM	16	Same as code 26	Q_{11} of instruction is the units position of the field to be transmitted
Transmit Record	TR	31	Storage address to which high-order position of the record is transmitted	*Storage address of high-order position of the record to be transmitted
Transfer Numerical Strip (special feature)	TNS	72	Storage address of right most position of alphameric field to be transmitted	*Storage address of the units position of the numerical field
Transfer Numerical Fill (special feature)	TNF	73	Storage address of rightmost position of alphameric field	*Storage address of the units position of the numerical field to be transmitted
Floating Shift Right (special feature)	FSR	08	Storage address to which units (rightmost) digit of mantissa is transmitted	*Storage address (rightmost) digit of mantissa to be transmitted
Floating Shift Left (special feature)	FSL	05	Storage address to which high-order digit of the mantissa is transmitted	*Storage address of low-order digit of mantissa to be transmitted
Transmit Floating (special feature)	TFL	06	Storage address to which units position of exponent is transmitted	*Storage address of units position of exponent of field to be transmitted

Table 18. Summary of SPS Logic (Branch and Compare) Instructions

NOTE: Both the BI (Branch Indicator) and BNI (Branch No Indicator) instructions require one of the switch or indicator codes listed in Table 21 as a Q address. The code indicates the switch or indicator to be interrogated for status. To relieve the programmer of having to code a Q address, unique mnemonics are included in SPS language for both BI- and BNI-type instructions. For a unique mnemonic, the processor generates the actual machine language code 46 (Branch Indicator) or 47 (Branch No Indicator) and the Q address that represents the switch or indicator.

Indirect Addressing (special feature) is allowable with all P address operands listed below except Branch Back. An * to the left of the Q address operand indicates that this feature may be used with it.

OPERATION	OPERATION CODES		OPERANDS	
	MNEMONIC	ACTUAL	P ADDRESS	Q ADDRESS
Compare	C	24	Storage address of units position of the field to which another field is compared	*Storage address of units position of the field to be compared with the field at the P address
Compare Immediate	CM	14	Same as code 24	Q_{11} of instruction is units position of the field to be compared with the field at the P address
Branch	B	49	Storage address of the leftmost digit of the next instruction to be executed	Not used
Branch and adjust assignment counter	B7	49	Storage address of the leftmost digit of the next instruction to be executed	Not used. However, these five locations <u>are</u> used by the next instruction in sequence
Branch No Flag	BNF	44	Storage address of the leftmost digit of next instruction to be executed if branch occurs	*Storage address to be interrogated for presence of a flag bit
Branch No Record Mark	BNR	45	Same as code 44	*Storage address to be interrogated for presence of a record mark character
Branch No Group Mark	BNG	55	Same as code 44	*Storage address to be interrogated for presence of a group mark character
Branch on Digit	BD	43	Same as code 44	*Storage address to be interrogated for a digit other than zero
Branch Indicator	BI	46	Storage address of leftmost position of next instruction to be executed if indicator tested is on	Q_8 and Q_9 of instruction specify the program switch or indicator to be interrogated (see Table 21)
Unique Branch Indicator Mnemonics:				
Branch High	BH	46	Same as BI	None required

Table 18. Summary of SPS Logic (Branch and Compare) Instructions (cont'd.)

OPERATION	OPERATION CODE		OPERANDS	
	MNEMONIC	ACTUAL	P ADDRESS	Q ADDRESS
Branch Positive	BP	46	Same as BI	None required
Branch Equal	BE	46	Same as BI	None required
Branch Zero	BZ	46	Same as BI	None required
Branch Over-flow	BV	46	Same as BI	None required
Branch Any Data Check	BA	46	Same as BI	None required
Branch Not Low	BNL	46	Same as BI	None required
Branch Not Negative	BNN	46	Same as BI	None required
Branch Console Switch 1 On	BC1	46	Same as BI	None required
Branch Console Switch 2 On	BC2	46	Same as BI	None required
Branch Console Switch 3 On	BC3	46	Same as BI	None required
Branch Console Switch 4 On	BC4	46	Same as BI	None required
Branch Last Card	BLC	46	Same as BI	None required
Branch Expon-ent Check (special feature)	BXV	46	Same as BI	None required
Branch No Indicator	BNI	47	Storage address of left-most position of next instruction to be exe-cuted if indicator tested is off	Q_8 and Q_9 of instruction specify program switch or indicator to be inter-rogated (see Table 21)
Unique Branch No Indicator Mnemonics:				
Branch Not High	BNH	47	Same as BNI	None required
Branch Not Positive	BNP	47	Same as BNI	None required
Branch Not Equal	BNE	47	Same as BNI	None required
Branch Not Zero	BNZ	47	Same as BNI	None required

Table 18. Summary of SPS Logic (Branch and Compare) Instructions (cont'd.)

OPERATION	OPERATION CODE		OPERANDS	
	MNEMONIC	ACTUAL	P ADDRESS	Q ADDRESS
Branch No Overflow	BNV	47	Same as BNI	None required
Branch Not Any Data Check	BNA	47	Same as BNI	None required
Branch Low	BL	47	Same as BNI	None required
Branch Negative	BN	47	Same as BNI	None required
Branch Console Switch 1 Off	BNC1	47	Same as BNI	None required
Branch Console Switch 2 Off	BNC2	47	Same as BNI	None required
Branch Console Switch 3 Off	BNC3	47	Same as BNI	None required
Branch Console Switch 4 Off	BNC4	47	Same as BNI	None required
Branch Not Last Card	BNLC	47	Same as BNI	None required
Branch Not Exponent Check (special feature)	BNXV	47	Same as BNI	None required
Branch and Transmit	BT	27	P address minus one is the storage address to which the units position of the Q field is transmitted. P address is leftmost digit of the next instruction to be executed	*Storage address of units position of the field to be transmitted
Branch and Transmit Immediate	BTM	17	Same as code 27	Q_{11} of instruction is units position of field to be transmitted
Branch Back	BB	42	Not used	Not used
Branch Back and Adjust Assignment Counter	BB2	42	Not used. However, these five locations <u>are</u> used by the next instruction in sequence	Not used. However, these five locations <u>are</u> used by the next instruction in sequence
Branch and Transmit Floating (special feature)	BTFL	07	P address minus one is the storage address to which the units position of the exponent portion of the Q field is transmitted. P is the storage address of the leftmost digit of the next instruction to be executed	*Storage address of units position of exponent of field to be transmitted

Table 19. Summary of SPS Input and Output Instructions

NOTE: Indirect Addressing (special feature) is allowable with all P address operands, where a P operand is required. None of the Q operands shown may be used with Indirect Addressing.

OPERATION	OPERATION CODE		OPERANDS	
	MNEMONIC	ACTUAL	P ADDRESS	Q ADDRESS
Read Numerically	RN	36	Storage address at which leftmost (first) numerical character is stored	Q8 and Q9 of instruction specify input device
Unique Read Numerically Mnemonics:				
Read Numerically Typewriter	RNTY	36	Same as RN	None required
Read Numerically Paper Tape	RNPT	36	Same as RN	None required
Read Numerically Card	RNCD	36	Same as RN	None required
Write Numerically	WN	38	Storage address from which leftmost (first) numerical character is written	Q_8 and Q_9 of instruction specify output device
Unique Write Numerically Mnemonics:				
Write Numerically Typewriter	WNTY	38	Same as WN	None required
Write Numerically Paper Tape	WNPT	38	Same as WN	None required
Write Numerically Card	WNCD	38	Same as WN	None required
Dump Numerically	DN	35	Same as WN	Same as WN
Unique Dump Numerically Mnemonics:				
Dump Numerically Typewriter	DNTY	35	Same as WN	None required
Dump Numerically Paper Tape	DNPT	35	Same as WN	None required
Dump Numerically Card	DNCD	35	Same as WN	None required
Read Alphamerically	RA	37	Storage address at which numerical digit of leftmost (first) character is stored. (Zone digit of first character is at P minus one)	Q_8 and Q_9 of instruction specify input device
Unique Read Alphamerically Mnemonics:				
Read Alphamerically Typewriter	RATY	37	Same as RA	None required
Read Alphamerically Paper Tape	RAPT	37	Same as RA	None required

Table 19. Summary of SPS Input and Output Instructions (cont'd.)

OPERATION	OPERATION CODE		OPERANDS	
	MNEMONIC	ACTUAL	P ADDRESS	Q ADDRESS
Read Alphamerically Card	RACD	37	Same as RA	None required
Write Alphamerically	WA	39	Storage address of numerical digit of leftmost (first) character to be written. (Zone digit of first character is at P minus one)	Q_8 and Q_9 of instruction specify output device
Unique Write Alphamerically Mnemonics:				
Write Alphamerically Typewriter	WATY	39	Same as WA	None required
Write Alphamerically Paper Tape	WAPT	39	Same as WA	None required
Write Alphamerically Card	WACD	39	Same as WA	None required
Control	K	34	Not used	Q_8 and Q_9 specify input/output device. Q_{11} specifies control functions
Unique Control Mnemonics:				
Tabulate Typewriter	TBTY	34	Not used	None required
Return Carriage Typewriter	RCTY	34	Not used	None required
Space Typewriter	SPTY	34	Not used	None required
Seek	SK	34	Storage address of disk control field	X07X1
Read Disk/WLRC	RDGN	36	Same as SK	X07X0
Write Disk/WLRC	WDGN	38	Same as SK	X07X0
Check Disk/WLRC	CDGN	36	Same as SK	X07X1
Read Disk Track/WLRC	RTGN	36	Same as SK	X07X4
Write Disk Track/WLRC	WTGN	38	Same as SK	X07X4
Check Disk Track/WLRC	CTGN	36	Same as SK	X07X5
Read Disk	RDN	36	Same as SK	X07X2
Write Disk	WDN	38	Same as SK	X07X2
Check Disk	CDN	36	Same as SK	X07X3
Read Disk Track	RTN	36	Same as SK	X07X6

Table 19. Summary of SPS Input and Output Instructions (cont'd.)

OPERATION	OPERATION CODE		OPERANDS	
	MNEMONIC	ACTUAL	P ADDRESS	Q ADDRESS
Write Disk Track	WTN	38	Same as SK	X07X6
Check Disk Track	CTN	36	Same as SK	X07X7

Table 20. Summary of SPS Miscellaneous Instructions

NOTE: Indirect Addressing (special feature) is allowable with all P or Q address operands that are marked with an *.

OPERATION	OPERATION CODE		OPERANDS	
	MNEMONIC	ACTUAL	P ADDRESS	Q ADDRESS
Set Flag	SF	32	*Storage address at which flag bit is placed	Not used
Clear Flag	CF	33	*Storage address from which flag bit is cleared	Not used
Move Flag (special feature)	MF	71	*Storage address to which flag bit is moved	*Storage address of flag bit to be moved
Halt	H	48	Not used	Not used
No Operation	NOP	41	Not used	Not used

Table 21. 1620/1710 Indicator Codes for SPS BI-BNI Instructions

NOTE: This table lists only those indicators that do not have unique mnemonics.

INDICATOR	Q ADDRESS				
	Q_7	Q_8	Q_9	Q_{10}	Q_{11}
Read Check	0	6			
Write Check	0	7			
MAR Check	0	8			
MBR-E Check	1	6			
MBR-O Check	1	7			
Operator Entry	1	8			
Terminal Address Selector (TAS) Check	2	1			
Function Register Check	2	2			
Analog Output (AO) Check	2	3			
Mask	2	6			
Customer Engineer (CE) Interrupt	2	7			
Analog Output Setup	2	8			
Multiplexer Busy	2	9			
Multiplex Complete	4	0			
Analog Output Setup Interrupt	4	1			
One Minute Interrupt	4	3			
One Hour Interrupt	4	4			
Any SIOC Interrupt	4	5			
Process Interrupt 1	4	8			
Process Interrupt 2	4	9			
Process Interrupt 3	5	0			
Process Interrupt 4	5	1			
Process Interrupt 5	5	2			
Process Interrupt 6	5	3			
Process Interrupt 7	5	4			
Process Interrupt 8	5	5			
Process Interrupt 9	5	6			
Process Interrupt 10	5	7			
Process Interrupt 11	5	8			
Process Interrupt 12	5	9			
Process Branch Indicator 1	7	0			
Process Branch Indicator 2	7	1			
Process Branch Indicator 3	7	2			
Process Branch Indicator 4	7	3			
Process Branch Indicator 5	7	4			
Process Branch Indicator 6	7	5			
Process Branch Indicator 7	7	6			

INDICATOR	Q ADDRESS				
	Q_7	Q_8	Q_9	Q_{10}	Q_{11}
Process Branch Indicator 8	7	7			
Process Branch Indicator 9	7	8			
Process Branch Indicator 10	7	9			
Process Branch Indicator 11	8	0			
Process Branch Indicator 12	8	1			
Process Branch Indicator 13	8	2			
Process Branch Indicator 14	8	3			
Process Branch Indicator 15	8	4			
Process Branch Indicator 16	8	5			
Process Branch Indicator 17	8	6			
Process Branch Indicator 18	8	7			
Process Branch Indicator 19	8	8			
Process Branch Indicator 20	8	9			
SIOC Output Error	6	0	4	3	
Alert	6	0	4	5	
SIOC Unit 1 Response	6	0	7	0	
SIOC Unit 2 Response	6	0	7	1	
SIOC Unit 3 Response	6	0	7	2	
SIOC Unit 4 Response	6	0	7	3	
SIOC Unit 5 Response	6	0	7	4	
SIOC Unit 6 Response	6	0	7	5	
SIOC Unit 7 Response	6	0	7	6	
SIOC Unit 8 Response	6	0	7	7	
SIOC Unit 9 Response	6	0	7	8	
SIOC Unit 10 Response	6	0	7	9	
SIOC Unit 11 Response	6	0	8	0	
SIOC Unit 12 Response	6	0	8	1	
SIOC Unit 13 Response	6	0	8	2	
SIOC Unit 14 Response	6	0	8	3	
SIOC Unit 15 Response	6	0	8	4	
SIOC Unit 16 Response	6	0	8	5	
SIOC Unit 17 Response	6	0	8	6	
SIOC Unit 18 Response	6	0	8	7	
SIOC Unit 19 Response	6	0	8	8	
SIOC Unit 20 Response	6	0	8	9	
Disk Address Check	3	6			
WLR/RBC	3	7			
Cylinder Overflow	3	8			
Any Disk Check	3	9			
Seek Complete	4	2			

Table 22. 1710 SPS Operation Codes

OPERATION	OPERATION CODE		OPERANDS	
	MNEMONIC	ACTUAL	P ADDRESS	Q ADDRESS
Select Address and Operate	SAO	84	Not used	Q_7 specifies operation; Q_9 - Q_{11} specify a terminal address
Unique Select Address and Operate Mnemonics:				
Select Address	SA	84	Same as SAO	$Q_7 = 1$; Q_9 - Q_{11} specify terminal address of analog input point
Select Address and Contact Operate	SACO	84	Same as SAO	$Q_7 = 2$; Q_9 - Q_{11} specify terminal address of contact point
Select Analog Output Signal	SAOS	84	Same as SAO	$Q_7 = 3$; Q_9 - Q_{11} specify terminal address of analog output channel
Select Read Numerically	SLRN	86	Depends upon particular operation	Depends upon particular operation
Unique Select Read Numerically Mnemonics:				
Select TAS	SLTA	86	Core location where high-order position of TAS is transferred	$Q_7 = 1$; Q_8 - Q_{11} are not used
Select ADC Register	SLAR	86	Core location where high-order position of ADC register is transferred	$Q_7 = 2$; Q_9 - Q_{11} specify analog input address
Select Contact Block	SLCB	86	Core location where status of the first contact scanned is stored	$Q_7 = 7$; Q_9 - Q_{11} specifies the contact block address where reading begins
Select Real-Time Clock	SLTC	86	Core location where high-order digit of RTC is transferred	$Q_7 = 4$; Q_8 - Q_{11} are not used
Select ADC and Increment (1711 Model 1)	SLAD	86	Core location where high-order position of ADC is transferred	$Q_7 = 6$; Q_8 - Q_{11} are not used
Select Manual Entry Switches	SLME	86	Core location where high-order digit of Manual Entry switches is transferred	$Q_7 = 8$; Q_8 - Q_{11} are not used
Branch Out Of Noninterruptible Mode	BO	47	Address to be placed in IR-3	Q_8 - $Q_9 = 00$; $Q_{11} = 0$
Branch Out Of Noninterruptible Mode and Load	BOLD	47	Address to be placed in IR-1	Q_8 - $Q_9 = 00$; $Q_{11} = 1$
Mask	MK	46	Not used	Q_8 - $Q_9 = 00$; $Q_{11} = 1$
Unmask	UMK	46	Not used	Q_8 - $Q_9 = 00$; $Q_{11} = 0$
Select Input Channel	SLIC	86	Not used	Q_{10} - Q_{11} specify the address of an SIOC input unit

Table 22. 1710 SPS Operation Codes (cont'd.)

OPERATION	OPERATION CODE		OPERANDS	
	MNEMONIC	ACTUAL	P ADDRESS	Q ADDRESS
Read Numerical Input Channel	RNIC	$8\bar{6}$	Core storage location where data is to be read	$Q_7 = 5$; $Q_8 - Q_{11}$ not used
Read Alpha-meric Input Channel	RAIC	$8\bar{7}$	Same as RNIC	Same as RNIC
Write Numerical Output Channel	WNOC	88	Core storage location from which data is to be written	$Q_{10} - Q_{11}$ specify an SIOC output unit
Write Alpha-meric Output Channel	WAOC	89	Same as WNOC	Same as WNOC
Unique SIOC Branch Indicator Mnemonics:				
Branch Output Record Mark	BOR	46	Core storage address of leftmost position of next instruction to be executed if indicator tested is on	None required
Branch End of Message	BRE	46	Same as BOR	Same as BOR
Branch Mode Shift	BMC	46	Same as BOR	Same as BOR
Branch Data Ready	BIR	46	Same as BOR	Same as BOR
Branch SIOC Not Busy	BCNB	46	Same as BOR	Same as BOR
Unique SIOC Branch No Indicator Mnemonics:				
Branch No Output Record Mark	BNOR	47	Core storage address of leftmost position of next instruction to be executed if indicator tested is off	None required
Branch No End of Message	BNRE	47	Same as BNOR	Same as BNOR
Branch No Mode Shift	BNMC	47	Same as BNOR	Same as BNOR
Branch No Data Ready	BNIR	47	Same as BNOR	Same as BNOR
Branch No SIOC Not Busy	BCB	47	Same as BNOR	Same as BNOR

Table 23. SPS Subroutine Macro-instruction Execution Times

NOTE: These execution times depict the total time from the encountering of a Macro-statement to the "return to mainline."

SUBROUTINE	AVERAGE EXECUTION TIME
Floating Add	Fixed length Average time = 9 ms Variable length Average time (in μs) = $5L^2 + 482L + 6854$ where L=length of mantissa
Floating Subtract	Fixed length Average time = 10.5 ms Variable length Average time (in μs) = $5L^2 + 482L + 7474$
Floating Multiply	Fixed length Average time = 18 ms Variable length Average time (in μs) = $168L^2 + 240L + 7400$
Floating Divide	Fixed length Average time = 55 ms Variable length Average time (in μs) = $520L^2 + 1500L + 7890$
Fixed Point Divide	Fixed length and variable length Average time (in ms) = 9.80 + .040 LDVD + (.520 LDVR + .740) (100-B1) where LDVD is length of dividend field, LDVR is length of divisor field, and B1 is value specified in macro-instruction
Floating Shift Right	Fixed length and variable length Average time (in μs) = 4960 + 960L - 880 (A-B)
Floating Shift Left	Fixed length and variable length Average time (in μs) = 6460 + 1520 (B-A) - 360L
Transmit Floating	Fixed length and variable length Average time (in μs) = 400 + 40L
Branch and Transmit Floating	Fixed length and variable length Average time (in μs) = 2280 + 40L
Floating Square Root	Fixed length Average time = 120 ms Variable length Average time (in μs) = $620L^2 + 9776L + 5328$
Floating Sine	Fixed length Average time = 150 ms Variable length Average time (in μs) = $168L^3 + 3792L^2 + 13340L + 4708$ NOTE: These executions times are for arguments less than 2π. Arguments greater than 2π are reduced by subtractions of 2π until within range. Therefore, the time required to perform these subtractions should be added to the average time required for an argument less than 2π.
Floating Cosine	Fixed length Average time = 155 ms Variable length Average time (in μs) = $168L^3 + 3792L^2 + 13420L + 5228$
Floating Arctangent	Fixed length Average time = 260 ms Variable length Average time (in μs) = $168L^3 + 2996L^2 + 7792L + 7260$
Floating Exponential (Natural)	Fixed length Average time = 160 ms NOTE: Add 70 ms to the average time if B is negative. Variable length Average time (in μs) = $168L^3 + 3582L^2 + 15890L + 26418$
Floating Exponential (Base 10)	Fixed length Average time = 145 ms NOTE: Add 70 ms to the average time if B is negative. Variable length Average time (in μs) = $168L^3 + 3656L^2 + 15414L + 24538$
Floating Logarithm (Natural)	Fixed length Average time = 290 ms Variable length Average time (in μs) = $168L^3 + 3440L^2 + 10530L + 12180$
Floating Logarithm (Base 10)	Fixed length Average time = 305 ms Variable length Average time (in μs) = $168L^3 + 3608L^2 + 11610L + 15108$

Table 24. 1620 Character Coding

	Character	Input			Core Storage		Output		
		Typewriter	Tape	Card	Alpha	Num	Typewriter	Tape	Card
ALPHAMERIC MODE	(Blank)	(Space)	C	(Blank)	C	C	(Space)	C	(Blank)
	. (Period)	.	X0821	12,3,8	C	3	.	X0821	12,3,8
	)	)	X0C84	12,4,8	C	4	)	X0C84	12,4,8
	+	+	X0C	12	1	C	+	X0C	12
	$	$	XC821	11,3,8	1	3	$	XC821	11,3,8
	*	*	X84	11,8,4	1	4	*	X84	11,4,8
	- (Hyphen)	-	X	11	2	C	-	X	11
	/	/	0C1	0,1	2	1	/	0C1	0,1
	, (Comma)	,	0C821	0,3,8	2	3	,	0C821	0,3,8
	(	(	084	0,4,8	2	4	(	084	0,4,8
	=	=	821	3,8	3	3	=	821	3,8
	@	@	C84	4,8	3	4	@	C84	4,8
	A-I	A-I	X0, 1-9	12, 1-9	4	1-9	A-I	X0, 1-9	12, 1-9
	0 (—)	(None)	(None)	11,0	5	C	- (Hyphen)	X	11,0
	J-R	J-R	X, 1-9	11, 1-9	5	1-9	J-R	X, 1-9	11, 1-9
	1-9 (—)	J-R	X, 1-9	11, 1-9	5	1-9	J-R	X, 1-9	11, 1-9
	S-Z	S-Z	0, 2-9	0, 2-9	6	2-9	S-Z	0, 2-9	0, 2-9
	0 (+)	0	0	0 or 12,0	7	C	0	0	0
	1-9 (+)	1-9	1-9	1-9	7	1-9	1-9	1-9	1-9
	‡	‡	082	0,2,8	C	C82	(Stop)	EOL	0,2,8
NUMERICAL MODE	(Blank)	(Space)	C	(Blank)		C	0	0	0
	0 (+)	0	0	0		C	0	0	0
	0 (-)	0̄	X, X0C	11,0		F	0	X	*11 or 11,0
	1-9 (+)	1-9	1-9	1-9		1-9	1-9	1-9	1-9
	1-9 (-)	1̄-9̄	X, 1-9	11, 1-9		F, 1-9	1-9	X, 1-9	11, 1-9
	‡	‡	082	0,2,8		C82	(Stop, WN) ‡ (DN)	EOL(WN) 082 (DN)	0,2,8
	‡̄	‡̄	X82	11,8,2		F82	‡̄	X82	11,8,2
	⧧	⧧	08421	0,7,8		**C8421	⧧	08421	0,7,8
	⧧̄	⧧̄	X8421	12,7,8		F8421	⧧̄	X8421	12,7,8
	Num Blank †	@	C84	4,8		C84	@	C84	(Blank)

† For Card Format Use Only
* Dump Numerically Operation 11 only
Write Numerically Operation 11,0
** Recorded as 0,8,4,2,1 in disk storage

Table 25. Core Storage Data Resulting From Reading Alphameric Card Data with RN Instruction

Alpha Character	Bits Entered into Core Storage by Read Numerically Instruction					
	C	F	8	4	2	1
A						X
B					X	
C	X				X	X
D				X		
E	X			X		X
F	X			X	X	
G				X	X	X
H			X			
I	X		X			X
J	X	X				X
K	X	X			X	
L		X			X	X
M	X	X		X		
N		X		X		X
O		X		X	X	
P	X	X		X	X	X
Q	X	X	X			
R		X	X			X
S					X	
T	X				X	X
U				X		
V	X			X		X
W	X			X	X	
X				X	X	X
Y			X			
Z	X		X			X

Alpha Character	Bits Entered into Core Storage by Read Numerically Instruction					
	C	F	8	4	2	1
0	X					
1						X
2					X	
3	X				X	X
4				X		
5	X			X		X
6	X			X	X	
7				X	X	X
8			X			
9	X		X			X
/						X
[1]. (period)			X		X	X
[1], (comma)			X		X	X
@	X		X	X		
(	X		X	X		
)	X		X	X		
[1]=			X		X	X
*		X	X	X		
–		X				
+	X					
Card I/O Only 11, 0		X				
Card I/O Only 12, 0	X					
‡	X		X		X	
[1]$	X	X	X		X	X
Blank	X					

[1] Interpreted as Record Mark on WN and TR instructions.

Appendix B

Alphabetic Listing of all Messages

Message	Program	Pages
$\bar{X}$XXX DO TABLE FULL	FORTRAN	122
$\bar{X}$XXX IMPROPER DO NESTING	FORTRAN	122
$\bar{X}$XXX MIXED MODE	FORTRAN	122
$\bar{X}$XXX SYMBOL TABLE FULL	FORTRAN	122
XXXXX ALABEL + XXXX ERN (see Table 6)	SPS	88
XXXXX CARD SEQ ERROR, CORRECT AND START	Loader	139
$\bar{X}$XXXX CORES USED XXXXX NEXT COMMON END OF COMPILATION	FORTRAN	124
XXXXX LD1	Supervisor	28
$\bar{X}$XXXX NAME	FORTRAN	128
XXXXX SECTORS OF DATA COPIED FROM XXXXXX TO XXXXXX	DUP	39
XXXXXX LOADED FROM XXXXXX TO XXXXXX	Loader	139
XX XXXXX OVERLAP	FORTRAN	131
XXXXXXXXXX TYPE CHANGE	DUP	33
$\bar{X}$XXXXX XXXXX XXXXX LOADED	FORTRAN	130
XXXX + XXXX ERROR N	FORTRAN	122
XXX . . . X (ID)	SPS	87
ALREADY DEFINED	SPS	93
AND FILE PROTECTED	DUP	39
BAD DISK WRITE. RESET START	Supervisor	26
CANNOT RESTORE COMMON – RESET AND START TO RETRY	Supervisor	19
CDP ERR XXXXX $\bar{0}$6 $\bar{0}$7 $\bar{1}$6 $\bar{1}$7 $\bar{3}$6 $\bar{3}$7 $\bar{3}$8	Supervisor	25
CDR ERR XXXXX $\bar{0}$6 $\bar{0}$7 $\bar{1}$6 $\bar{1}$7 $\bar{3}$6 $\bar{3}$7 $\bar{3}$8	Supervisor	25
CONDITION IGNORED	Supervisor	19
CONTROL STATEMENT INVALID, RE-ENTER	Loader	140
CORE CAPACITY EXCEEDED BY XXXXX LOCATIONS PROGRAM IS TERMINATED	SPS	92
CORRECTIONS HAVE NOT BEEN ENTERED	DUP	34
DISK RD WR ERROR, START TO RETRY	Loader	139
DISK SECTOR XXXXX CORRECTED	DUP	34
DK LOADED XX XX	DUP	32, 45
DSK ERR XXXXX $\bar{0}$6 $\bar{0}$7 $\bar{1}$6 $\bar{1}$7 $\bar{3}$6 $\bar{3}$7 $\bar{3}$8	Supervisor	26
DSK OFL	Supervisor	26
DUP * ERROR 01 (01 through 21)	DUP	40, 42, 43, 44
DUP * ERROR 51 (51 through 61)	DUP	44
DUP * TURN OFF WRITE ADDRESS KEY, START	DUP	32
DUP * TURN ON WRITE ADDRESS KEY, START	DUP	32
END OF ASSEMBLY XXXXX CORE POSITIONS REQUIRED XXXXX STATEMENTS PROCESSED	SPS	90
END OF JOB	Supervisor	14, 19
ENT ERROR $\bar{0}$6 $\bar{0}$7 $\bar{1}$6 $\bar{1}$7 $\bar{3}$6 $\bar{3}$7 $\bar{3}$8	Supervisor	25
ENTER DUP CNTRL REC	DUP	32
ENTER MONITOR CONTROL RECORD	Supervisor	19, 21
ER D1 (D1 through G7)	FORTRAN	132
ERROR L1 (L1 through L10)	FORTRAN	130
ERROR X	FORTRAN	122
ERROR IN FIELD AT COL. XX. SET SW 4 TO IGNORE, OFF TO RE-ENTER CARD	Supervisor	19
ERROR IN FIELD AT COLUMN XX. PHASE TERMINATED	Supervisor	20
ERROR, INVALID CONTROL RECORD	FORTRAN	120
ERROR, INVALID OUTPUT UNIT CODE	FORTRAN	121
EXCEEDED SPECIFIED CAPACITY BY XXXXX	SPS	91
EXECUTION	Supervisor	19
EXECUTION INHIBITED	FORTRAN	131
EXECUTION IS INHIBITED	Supervisor	21
ERROR, F OR K OUTSIDE RANGE	FORTRAN	121
FORTRAN LIBR NAME ENTERED XXXXXXXXXX	DUP	42
FLIPER XXXXX OVERLAP	FORTRAN	131
IMP ERR	Supervisor	26
IMPROPER IND CODE IN SUBR XXXX	SPS	92
JOB ABANDONED	FORTRAN	131
JOB CARD GROUP ONLY	Supervisor	19
LD 2	Supervisor	28
LD 3	Supervisor	28
LD 4	Supervisor	28
LOAD SUBNAM	FORTRAN	131
MAP ERR XXXXX XXXX	Supervisor	26
MAX N2 ALLOWABLE $\bar{X}$XXXX	FORTRAN	132
MOD ERR XXXXX	Supervisor	26
MORE THAN 5 CYLINDERS OF RELOADABLE OUTPUT SSW 4 ON TO DUMP OUTPUT, OFF TO CONTINUE, NO OUTPUT	SPS	91
MUST RELOAD	Supervisor	27
NAME XXXXX OVERLAP	FORTRAN	131

Message	Program	Pages
NO DIM ENTRY FOR SUBROUTINE	SPS	91
NO ROOM IN TABLE	SPS	93
NO TRAILER REC. CORRECT, RELOAD COMPLETE DECK WITH CNTR REC, AND BR TO 7404	Loader	139
NOT IN TABLE	SPS	93
OBJECT DIM ERROR PHASE TERMINATED	Supervisor	20
OBJECT NAME ERROR PHASE TERMINATED	Supervisor	21
PACK NUMBER. ERROR ON MODULE X. SET SSW4 TO IGNORE OFF TO RECOMPARE	Supervisor	19
PTR ERR XXXXX $\bar{0}6$ $\bar{0}7$ $\bar{1}6$ $\bar{1}7$ $\bar{3}6$ $\bar{3}7$ $\bar{3}8$	Supervisor	25
PTP ERR XXXXX $\bar{0}6$ $\bar{0}7$ $\bar{1}6$ $\bar{1}7$ $\bar{3}6$ $\bar{3}7$ $\bar{3}8$	Supervisor	25
RDER	Loader	139
RE-ENTER STATEMENT	SPS	90
RE-ENTER OPERANDS	SPS	90
SECTION	DUP	33, 34
SECTOR	DUP	33
SECTION NUMBER ILLEGAL, START TO RE-ENTER * DALTR	DUP	33
SECTOR ADDRESS ILLEGAL START TO RE-ENTER * DALTR	DUP	33
SEQ	Loader	139
SUBR NOT LOCATED IN SUBROUTINE MAP	SPS	92
SUBROUTINES OTHER THAN PGM DIV USED	SPS	91
SYSTEM DIM ERROR PHASE TERMINATED	Supervisor	21
TRAILER CARD SEQ ERROR, CORRECT AND START	Loader	140
TRP ERR	Supervisor	26
TRP ERROR MUST RELOAD	Supervisor	27
TYP ERR XXXXX $\bar{0}6$ $\bar{0}7$ $\bar{1}6$ $\bar{1}7$ $\bar{3}6$ $\bar{3}7$ $\bar{3}8$	Supervisor	25
TYPE-IN EXCEEDS SECTOR LENGTH, START	DUP	34
UNDEFINED SYMBOL XXXXX	SPS	93
WRITE AND SAVE (SEEK START STOP)	DUP	33
WRITE AND ZERO (SEEK START STOP)	DUP	33

NOTE: X's represent variable characters.

Index

GC26-5739-4

IBM 1620 Printed in U.S.A. GC26-5739-4

IBM®

International Business Machines Corporation
Data Processing Division
112 East Post Road, White Plains, N. Y. 10601