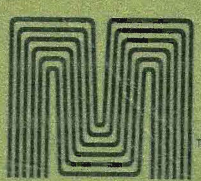
REALITY
Computer System Reference Manual



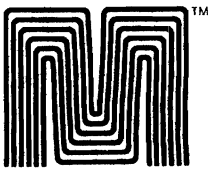
Microdata

2333

REALITY
Computer System
Reference Manual

Revised
August 1974
Revision 2

PROPRIETARY INFORMATION



™ Microdata

Microdata Corporation
17481 Red Hill Avenue
Irvine, California 92705
Telephone (714) 540-6730
TWX 910-595-1764

Microdata

Microdata Corporation
17481 Red Hill Avenue
Irvine, California 92714
(714) 540-6730 TWX: 910-595-1764

The Microdata REALITY Computer System Reference Manual will be revised periodically. If you desire to receive revisions to this manual, you must complete the following data request card with the name of the person you want to receive the revision data. Return the card to Microdata.

Please Return This Card To Microdata Corporation

Name _____ Title _____

Company _____

Address _____

City _____ State _____ Zip _____

Telephone (____) _____ Ext. _____

Microdata REALITY Computer System Reference Manual Copy Number # #2333

REV. 2

First Class
Permit No. 1972
Santa Ana
California 92711

BUSINESS REPLY MAIL

NO POSTAGE NECESSARY IF MAILED IN UNITED STATES

Postage Will be Paid by:

Microdata Corporation

17481 Red Hill Avenue
Irvine, California 92714

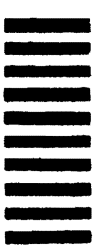


TABLE OF CONTENTS

TABLE OF CONTENTS (Continued)

TABLE OF CONTENTS (Continued)

TABLE OF CONTENTS (Continued)

TABLE OF CONTENTS (Continued)

TABLE OF CONTENTS (Continued)

TABLE OF CONTENTS (Continued)

SECTION I

INTRODUCTION

Reality is a completely new system of computer hardware and software, specifically oriented to provide a vehicle for the implementation of cost-effective information management. Information management systems implemented in Reality afford two major benefits; they are: (1) providing accurate and timely information to form the basis for significantly improving the decision making process, and (2) substantially reducing the clerical and administrative effort associated with the collection, the storage, and dissemination of the information pertaining to an organization.

Reality is a completely new computer system combining both proprietary hardware and proprietary software to create an effective tool for on-line information management. Through the use of Microprogramming, Microdata has implemented a truly revolutionary on-line transaction processing system. Three major components of the system have been implemented directly in firmware. They are, (1) virtual memory operating system; (2) the software level architecture; (3) the terminal input-output routines. The virtual memory operating system which has long been used in large computer systems has been impractical for minicomputers due to the large amount of overhead needed for the operating system itself. In Reality, the operating system has been directly implemented in highspeed, read-only memory (we called it firmware) which executes many times faster than would a comparable system normally implemented in software. Since the firmware is really an extension of the hardware of the computer hardware itself, this implementation is more precisely referred to as a virtual machine operating system. With the operating system directly implemented in read-only memory, only a small amount of main memory (core memory) is needed to run Reality.

Slightly over 4,000 bytes of core memory need be allocated for the operating system monitor. Everything else, system software, user software, and data, is transferred automatically into main memory from the disc drive by the virtual machine operating system in a demand-paged environment. Everything in the Reality computer system is organized into 512 byte pages, or <u>frames</u>, which are stored on the disc. The virtual concept allows the user to have access to a programming area not constrained by a main (core) memory, but as large as the entire available disc storage on the system.

The second feature implemented directly in firmware by Reality is the software level architecture of the machine itself. Through Microprogramming, Microdata has implemented a machine architecture expressly designed and optimized for information management. The assembly language architecture of Reality has very powerful instructions expressly designed for character moves, searches, compares, and all supporting operations germane to managing variable length fields and records. In addition, this software architecture has in existence a very large field proven software base written for information management. The information management software available on the Reality computer system equals or exceeds the software available for medium scale data processing systems costing several times the price of Microdata Reality.

The third major item implemented in Microcode is the input-output routines designed to handle communication with the terminals. In all minicomputer on-line applications, one of the main problems is that of managing the input and output from on-line interactive terminals. As these terminals increase in number, the load on the CPU becomes overwhelming and consequently the response to the terminals degrades dramatically. Microdata, in Reality, has implemented the transactions with the on-line terminals in high-speed microcode. The Microprogram implemented in read-only memory directly controls the communications from and to all of the on-line terminals connected to the Reality computer system. This means that the process execution need not be interrupted to handle a character coming in or going out to each and every terminal. The firmware handles or buffers all these transactions and only interrupts the software at the completion of a block. As a result, a very large number of terminals may be connected to the Microdata Reality System before any significant degradation in response time is detected. The response time is, of course, dependent upon the specific application and the activity level of all terminals. However, implementations of 10, 20, 30, or more terminals is not impractical. In fact, with the virtual machine operating system, which automatically manages the available resources of the computer, and the architecture itself, custom designed for data base management and all the terminal input and output handled directly by high-speed microcode, the Microdata Reality System excels as the number of terminals increases.

What does this mean to the user?

1. Due to the structure of Reality a large number of terminals can be accommodated with excellent terminal response times.

2. Due to the virtual machine implementation the user need not be concerned directly with the amount of main memory (core storage).

3. A large number of terminals (in excess of 32) may efficiently be on-line to the system.

4. All users (terminals) can share the Input/Output resources of the system. An Input/Output spooling subsystem permits any terminal to use the optional Magnetic Tape Drive and Line Printer.

5. All files can be interrogated and manipulated using the ENGLISH retrieval language, even those files built and maintained with RPG-II programs.

REALITY - CENTRAL PROCESSING UNIT

The Reality CPU, although physically small in size and priced in the minicomputer category, has the architecture of a medium scale computer. Its main memory is core and is expandable from 8,192 bytes to 65,536 bytes in increments of 8,192 bytes. Its full cycle operation is 1 microsecond per byte. The virtual memory is disc which is oriented into 512 byte frames expandable from 4,871 frames (2.5 million bytes) to 12,192,320 frames (6.4 billion bytes). That is the virtual memory addressing range of the

CPU itself. However, in standard configurations, the Microdata Reality system is currently configured from 5 million bytes to 80 million bytes of disc storage. The CPU is capable of handling a large number of asynchronous processes, each associated with an input-output device. The Reality CPU will support in excess of 32 terminals (or asynchronous processes). The CPU has 16 addressing registers and one extended accumulator for each terminal. A variable return stack accommodating up to 31 recursive subroutine calls for each terminal is also provided; however, current software convention allows only 11 entries in the stack. By indirect addressing through any one of the 16 registers, any byte in the virtual memory can be accessed. Relative addressing is also possible using an off-set displacement plus one of the 16 registers to any bit, byte, word (16 bits), double word (32 bit) or triple word (48 bits) in the entire virtual memory.

OPERATING SYSTEM

The operating system of the Microdata Reality is unique in that it is implemented directly in firmware and as such is an extension of the hardware. The features of the Microdata Reality operating system are summarized below:

Operating System (Hardware) features include:

Selection of process for execution and determination of length of execution.

Management of the allocation of core memory buffers containing disc frames.

Processing of implicit and explicit frame faults (requests for core/disc transfers).

Processing of logically linked frames and presenting them as physically sequential.

Processing of inter-module linkage and maintenance of return stacks.

Recognize process defined breakpoints and generate software traps.

Minimum core resident overhead per defined process (32 bytes).

Full duplex byte (character) I/O, to buffer transfers between a process and its associated device, echo input bytes, process parity bits, and test input bytes for process activation.

Generation of software traps on abnormal conditions, illegal op-codes, return stack overflow/underflow, disc memory protect violation, arithmetic overflow/underflow and device interrupts.

Disc I/O with overlapped seeks using block multiplexed channels providing an average access time of 35 ms and a maximum through-put of one thousand 512 byte disc frame transfers per second.

One to four IBM compatible 9-track 800 bpi magnetic tapes for file back-up, historical files and communication with other computer systems.

Virtual memory read/write protection, to selectively lock critical areas of memory from access.

Power fail-safe for automatic, safe shutdown in event of power failure.

Real time clock and console settable user execution quantities.

Bootstrap program (hardware) to re-boot the system from disc, tape or any byte I/O device.

INSTRUCTION SET

The Reality System has an extensive instruction set, including:

Bit, Byte, word, double-word, and triple word operations.

Memory to memory operation using relative addressing on bytes, words, double-words, and triple-words; for the movement, addition or subtraction of the first operand to the second operand.

Bit operations permitting the setting, resetting, and branching on condition of a specific bit.

Branch instructions which permit the comparison of two relative memory operands and branching as a result of the compare.

Addressing register operations for incrementing, decrementing, saving and restoring addressing registers.

Byte string operations for the moving of arbitrarily long byte strings from one place to another; movement may be stopped on a count runout, addressing register reaching a specified value, or encountering up to any one of seven specified delimiters.

Operations for the conversion of binary numbers to printable ASCII characters and vice versa.

Arithmetic instructions for loading, storing, adding, subtracting, multiplying, and dividing the extended accumulator and a memory operand.

Control instructions for branching, subroutine calls, and program linkage.

ENGLISH

ENGLISH is a generalized information management data retrieval language. ENGLISH is a freeform order-independent language used to retrieve information from the data files of the Reality computer system. The language consists of verbs, nouns, connectives, and values. All information in the system is stored in self-describing data bases and retrieved through the use of dictionaries or tables.

The verbs of ENGLISH are action oriented words such as list, sort, select, sum, etc. which evoke one of the ENGLISH processors.

Nouns are either the names of files or the names of attributes. They are assigned by the user and can have as many synonyms as the user finds necessary.

Connectives are provided to modify and qualify ENGLISH statements. Modifiers are nouned or phrased limiters whose impact is to limit the depth of action initiated by the verb.

Qualifiers are value limiters which logically qualify values such as equal to, not equal to, greater than, less than.

Not only does ENGLISH provide an ability to selectively or conditionally retrieve information, it also provides an automatic report generate capability. The report which normally appears on the terminal but optionally can be transmitted to the line printer for hard copy output is automatically formatted for the user by the Reality computer system. Listing output will be processed through a formatter which will create a columnar list, if possible; otherwise vertical output will be created. The output may be sorted into any sequence defined by the user and attributes may be totaled based on user specified control breaks.

The update capability permits the adding, changing or deleting items or attribute values for a specific item or items. As with the retrieval capability, updates may also be performed selectively on only those items meeting defined conditionals.

Correlative codes, stored in the dictionaries, permit the user to define certain processing relationships for specific attributes. Using correlations, the user can define arbitrarily complex file inter-relationships and maintain these inter-relationships automatically. Correlations fall into three basic groups.

> Horizontal associations permit the chaining from an item in one data list to an item in another data list. These lists may be used to maintain inverted and cross indexed files and redundantly store data in multiple locations. Similarly they may trigger the automatic retrieval of data from secondary files, eliminating the need for redundantly stored data.

Vertical associations permit the construction and maintenance of hierarchial data structures. Previous and next links are maintained automatically, permitting insertions and deletion of items into indentured lists automatically.

Internal associations permit definition of relationships within a single item. These relationships include repeating, groups and non-stored attributes defined as a function of other stored attributes.

Data Audits provide definition of permissible characteristics for attribute values. Audits include size, type, pattern, table-look-up, and range checking.

Data conversion provides for automatic conversion of values on input and output. Conversion includes data conversion, table look-up conversion and data encoding.

Storage method of values for attributes may be specified as: single value; multiple value/non-redundant store; multiple-value/redundant store; positive post; negative post.

Every file and their individual attributes may be secured for either update or retrieval by the assignment of security codes. At log on time, each user gets a list of pre-defined security codes which are then matched to codes on requested files and attributes. Only those with matches are retrieved or updated.

SOFTWARE

The software available on the Reality computer system is the most extensive data base management software available on any minicomputer. A summary of some of the processors available to all terminal users is presented below:

A high level two pass symbolic assembly language and macroprocessor translates REAL source statements, and can be used to implement cross-assemblers for other computers.

On-line editor - an interactive editor designed for creating, displaying, searching, and altering source programs and other bodies of text.

COPY, a file management processor, provides for data movement between disc files, tapes, line printers and terminals.

A file save and restore processor providing tape back-up for disc files.

ON/OFF processor to validate users wishing to gain access to the system and also to update accounting information.

MESSAGE, a message processor permitting the storing and forwarding of messages to other users whether currently on the system or not.

PROC, a facility allowing a user to define procedures "PROC's". A PROC can be used to define complex procedures involving multiple processor entry and conditional branching.

System subroutines are provided for use by user written programs, including:

An n-way polyphase disc sort/merge subroutine.

Routines for reading the standard input/output device (terminal).

Routines for retrieving and updating items in ENGLISH defined files.

A message formatter accepting value strings and formatting them into a message or report using a predefined format string.

Input/Output Spooling System which permits any terminal to use system peripherals such as the line printers or magnetic tape unit.

Numerous utility processors providing capability to:

Examine and alter physical frames.

Load assembled source programs.

List assembled source programs.

Define terminal characteristics.

On-Line Debug facilitates program debugging by:

Examining, inserting and modifying the program elements such as instructions and data.

Controlling execution by setting breakpoints at specific locations, and breaking on branches or external calls.

Single stepping execution.

Tracing execution by displaying information at designated points in a program.

Conventions regarding data typed in at the terminal.

The following conventions apply uniformly through this manual:

Where the format of a command to the system is described, upper-case characters or words are literal, that is, they represent the actual occurrence of that character or word; lower-case characters or words represent variables, that is, in actual use they are replaced by a specific value. For example, if the format is:

```
EDIT DICT file-name item-name (X,Q)
```

"EDIT", "DICT" and "(X,Q)" are literals and are to be entered exactly as shown; "file-name" and "item-name" are to be replaced in actual use by a specific value representing, respectively, the file-name and the item-name to be used. Thus, if "SYS-FILE" is a valid file-name, and "ABCD" is a valid item-name, the data:

```
EDIT DICT SYSGEN-FILE ABCD (X,Q)
```

is actually typed in.

In examples shown in this manual, data typed in at the computer terminal is underlined; computer-generated output is not.

The symbol (r) represents the entry of a carriage return or a line feed at the terminal.

Control characters are represented by the upper-case letter corresponding to the key used, with a superscript of a "c"; thus P^{c} represents "control-P"; further, P^{cs} represents "control-shift-P".

SECTION II

DATA STRUCTURES

INTRODUCTION

Reality is a virtual machine with all of the virtual memory (typically disc) being directly addressable as if it were in real memory (typically core). The virtual memory consists of a set of 512-byte frames, addressable by a positive integer called a Frame ID (FID). The entire set of data associated with a Reality system, including executable programs, process work spaces and all system and user files reside in virtual memory.

Executable Frames

Starting with frame one (FID=1), and continuing upward sequentially, are the executable frames. The extent of these frames (i.e., how high they go) is a system generation parameter. However, a minimum of 511 frames must be reserved: furthermore, not more than 4095 frames may be reserved. This initial area of the virtual memory contains every executable program or subroutine available with the Reality system. These frames are shared among all users. For example, the TCL processor is solely contained on frames 2, 4, and 5; a user executing in TCL uses these three frames simultaneously with all other users executing in TCL. Frames 1 to 399 are reserved for current and future Reality software. Frames 400 to 511 are available for user-developed software. The appendix to this section describes the frame locations for the Reality operating system software (not including ENGLISH software). Following the space reserved for the executable frames, beginning at frame number 512, is the process work space.

Process Work Space

A user interacts with the Reality system via an interactive terminal attached to a communications port on the Reality CPU. The on-going dialog with any port is called a process. Additionally certain processes not actually connected with a communications port may be defined at system generation time. These background processes can be used for such things as spooling data to a line-printer. Uniquely associated with each process is a primary control block (PCB) which is a one-frame block that defines the state of the process at any instant. The PCB contains the addressing registers for its process as well as the accumulator, condition flags, return stack and scan delimiters, all required by the hardware during execution by the process. Each PCB is followed by a 31-frame work space that is associated with it; thus 32 frames are reserved for each defined process and the first frame of each block is the PCB. Following the process work space is the file space, from which each process can get and release work space as required.

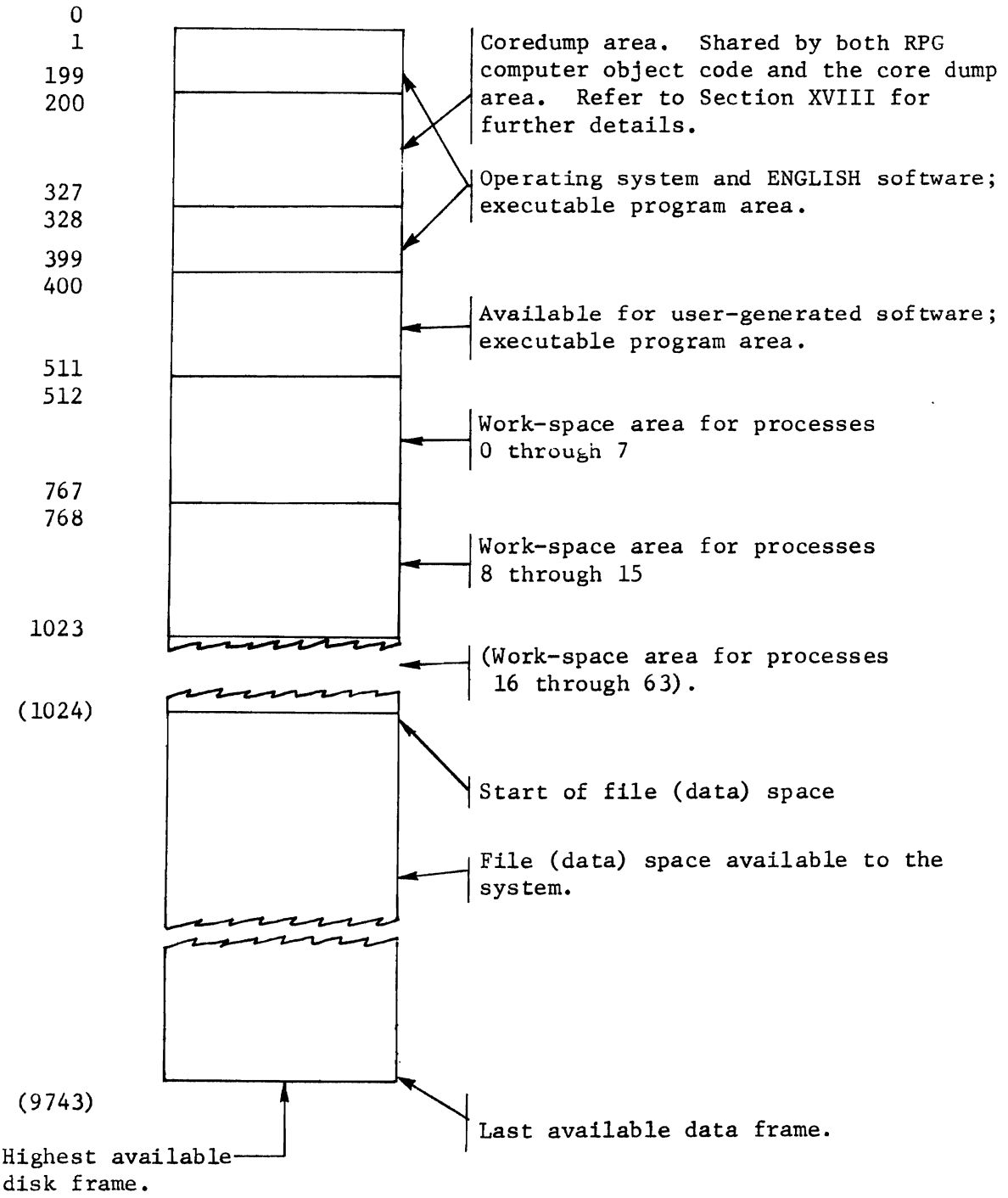
The 32 frames of process work space associated with each process are as follows:

Frame	Description and Symbolic Name
PCB + 0	Primary Control Block (PCB)
PCB + 1	Second any Control Block (SCB)
PCB + 2	DEBUG Control Block (DCB)
PCB + 3	Unassigned and unavailable
PCB + 4	BMS/50, AF/50, CS/100, IB/140, OB/140
PCB + 5	TS - one frame unlinked scratch area (TSBEG)
PCB + 6 to PCB + 9	Four frame PROC work area & stacks
PCB + 10 to PCB + 15	HS - six frame linked HS work area (HSBEG)
PCB + 16 to PCB + 21	IS - six frame linked IS work area (ISBEG)
PCB + 22 to PCB + 27	OS - six frame linked OS work area (OSBEG)
PCB + 28	UPD - one frame unlinked work area (UPDBEG) set up only when GETUPD is called from a user written program. Also used by RPG.
PCB + 29 to PCB + 31	Unassigned and unused, available for user programs.

It should be noted that the above work space assignments for HS, IS, and OS may be increased by the establishment of an appropriate entry in the LOGON item; however, the additional space is obtained from the common pool of overflow space. For a discussion of frame formats, refer to Section XV.

Disk Space Assignment

The map below describes the assignment of the disk space; the "highest available disk frame" number is dependent on the disk configuration for a particular system; several other FID's are also configuration-dependent; examples shown below (FID's in parentheses), are for a system with 32K bytes of core-memory and with one five megabyte (5 MB) disk, and sixteen processes.



In general, the configuration-dependent FID's may be computed using the rules below:

Start of file space (f_1) = 512 + u * 32 u = number of processes.

Highest available disk frame

FID_{max} =	9,743	One 5 MB disk
	19,487	Two 5 MB disks, or one 10 MB disk.
	38,975	Four 5 MB disks, or two 10 MB disks.
	77,951	Four 10 MB disks.

End of file space (f_2) = FID_{max}

File Space

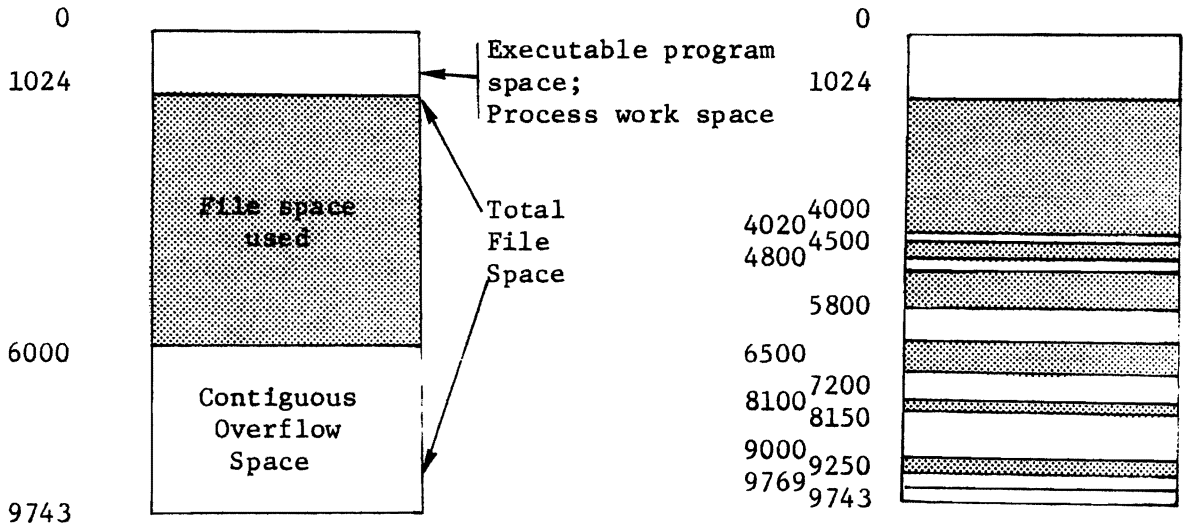
Beginning after the process work space, the remainder of the virtual memory is available for the storage of data in files. The beginning of this area is a system generation parameter. A direct access file technique is used and employs a hashing technique. All data is stored as items within files. Dictionaries, which are also files, are used to decode the formats of the data stored in an item.

Overflow Space Management

The areas of the disk that are not allocated to the files are maintained as a pool of overflow space that is available to the Reality system file management routines as additional data space, as well as to other processors as scratch work space. The Reality system maintains a table of pointers that define the available overflow space, which may be either in a "linked" form, or in a "contiguous" form. Contiguous overflow space, as the name implies, consists of blocks of contiguous frmmes defined by a set of 2 pointers that are all available, and can be taken out of the pool either singly or as a block. Linked overflow space can only be taken a frame-at-a-time. Conversely, space may be released by processors to the linked overflow pool a frame-at-a-time, or to the contiguous pool as a block.

At the conclusion of a file-restoration process on the Reality system, an initial condition may be said to exist; there is no linked overflow space, and only one block of contiguous overflow space, extending from the end of the current data space through the last available data frame. As the system obtains and releases overflow space, and as files are created and deleted, the overflow space gets fragmented, and at any particular time there may be several blocks of contiguous overflow space, and a

chain of linked overflow space. Representative examples of these two states are shown below; shaded areas indicate use of file space:



Start of Linked Overflow space	:	0
Contiguous space pointer sets.	:	[6000, 9743]

(end of table)

8000 (400 frames in linked set)

[4000, 4020]	first set
[6000, 6500]	second set
[8150, 9000]	third set
[9250, 9743]	fourth set

(end of table)

File Definition

A file is a mechanism for maintaining a set of like items logically together so that one can access these items for both retrieval and update. For the Reality system, this mechanism functions by operating on a specified item-id which uniquely identifies the item. A computational hashing technique is used which operates on the item-id, using several variables unique to the file, to produce a virtual memory address where the item is stored.

Terms used in defining and accessing files:

Item	A string of data associated with and including an item-id. Items are stored in files.
Item-id	A unique datum (key) within a file with which all of the data in an associated item is identified or referenced.
File	A set of items.
Group	An area (a set of linked frames) where items may be sequentially stored. It consists of one or more linked frames and can vary in size from file to file. (Usually 10 to 25 items per group.)
Base	The first FID of the first group in a given file.
Modulo	The number of groups allocated for a given file.
Separation	The number of frames initially allocated for each group in a file.

Hashing Algorithm

The "hashing" technique is used to distribute items within the physical structure of the file.

FID = BASE + [Remainder (Item-id/MODULO)]* SEPARATION

The item-id is treated as a variable length string of binary digits; dividing this value by the positive integer MODULO yields an unsigned integer remainder in the range:

$$0 \leq \text{remainder} < \text{MODULO}.$$

This is then the group number (i.e., 0, 1, 2, ... up to MODULO-1) where the item is to be stored. Multiplying by the SEPARATION and adding the BASE yields the actual FID of the first frame in the group.

Item Storage

After computing an FID to locate the specific group in which the item resides, each item's item-id in the group must be compared for a match. The frames comprising a group are linked both forward and backwards. This Reality system facility makes the group appear to be a physically sequential string where items are stored one immediately after another. In fact, any portion of an item may spill across a physically non-contiguous frame boundary. An example is included on the next page.

When a file is created it is allocated a primary area of (MODULO * SEPARATION) frames. Thus this amount of contiguous disk-space is permanently allocated to the file. As the file grows, individual groups may fill up. When this happens, an additional frame is added to the group from a pool of available space. This frame is linked into the group to increase the length of the logically sequential group. Additionally, if a delete or update causes the group to shrink, any unused frames outside the primary area are returned to the pool of available space.

Item Format-Physical

Character Count - The first four characters of an item are a hexadecimal character character count which specifies the total number of characters in the item, including the count field; the maximum size of an item is 32267 bytes (X'7E0B'). This character count is used to locate the beginning of the next item within a group.

Attribute Separation - After the character count is the beginning of the data in the item. The first datum is the Item-id identifying that item. Following, and marking the end of the Item-id is an attribute mark (X'FE'), which prints as "↑" or "^". Following the attribute mark are the attribute values, which may be of variable length, separated by additional attribute marks. An item is always terminated with an attribute mark.

Absense of Values - The absence of a value for an attribute is specified by an attribute mark (to maintain the proper attribute sequence) immediately following the attribute mark indicating the end of the previous value. The "space" between two adjacent attribute marks can be thought of as representing the absent value. If the last attributes within an item have no stored values, the item terminates with the Attribute Mark following the last value present. However, all items must terminate with an attribute mark. The minimum Item consists of only an Item-id followed by a single attribute mark.

Multiple Values - Between any two attribute marks (i.e. any one attribute value) multiple values may exist. These are separated by a value mark (X'FD') which prints as "]", in exactly the same manner an attribute mark separates attributes.

End of Group - An attribute mark immediately following an item signifies the end of a group. If a group is empty the first character of the group will be an attribute mark.

Dump of Sample File

The following "print out" was generated using the DUMP processor. It shows a sample file with BASE=1248, MODULO=3 and SEPAR=2. The DUMP processor assumes frames in a linked format as follows:

Byte 1	-	unused
Byte 2	-	number of next contiguous frames
Byte 3-6	-	next linked frame
Byte 7-10	-	previous linked frame
Byte 11	-	number of previous contiguous frames
Byte 12	-	unused
Byte 13-512		data portion

For each frame the first line shows the frame number (FID) and links fields in the above sequence. Subsequent lines display all non-blank data. The sample file contains one large item and all the linked frames including those outside the primary item are dumped. Attribute marks print as the character ^.

Example of File with 3 Groups and 2 Frames/Group

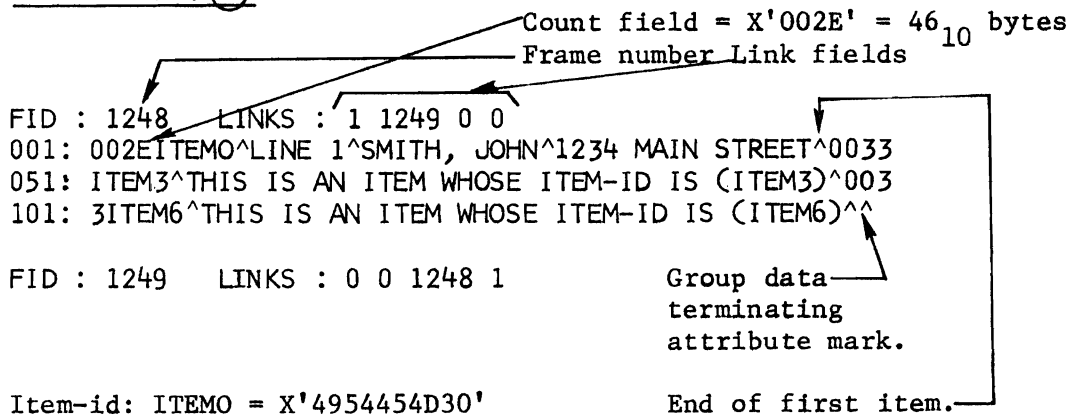
BASE = 1248, MODULO = 3, SEPAR = 2.

1st group (FID = 1248) has 3 items

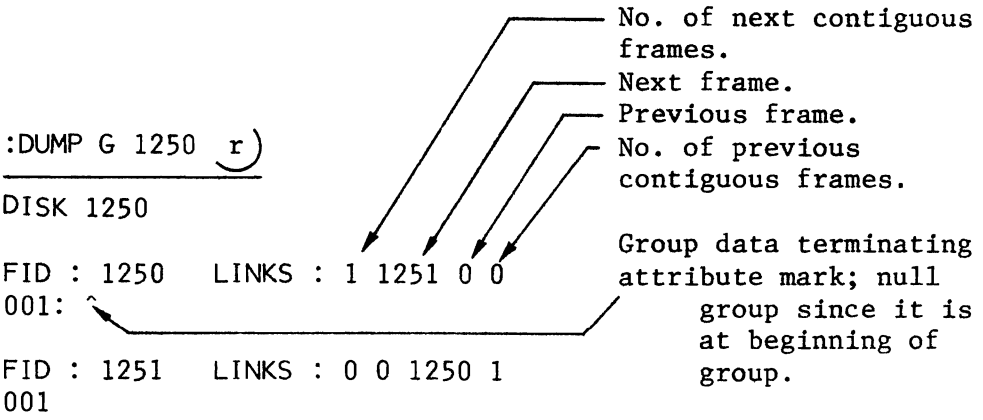
2nd group (FID = 1250) has no items

3rd group (FID = 1252) has 2 items

```
:DUMP G 1248 (r)
```




```
Item-id: ITEMO = X'4954454D30'
Hashing algorithm:
          FID  = Remainder(X'4954454D30' / 3) * 2 + 1248
               = 0 * 2 + 1248
               = 1248
```




```
:DUMP G 1250 (r)
DISK 1250

FID : 1250    LINKS : 1 1251 0 0
001: ^

FID : 1251    LINKS : 0 0 1250 1
001
```

The third group has a large item (size = X'74F' = 1871) causing the group to link out of the primary area into the overflow space.

```
:DUMP G 1252 (r)
DISK 1252

FID : 1252    LINKS : 1 1253 0 0
001: 074FITEM2^THIS IS AN ITEM WHOSE ITEM-ID IS (ITEM2)
051: ^^THE PREVIOUS ATTRIBUTE IS NULL^THIS IS THE FIRST
101:  VALUE OF A MULTI VALUED ATTRIBUTE]THIS IS THE SEC
151: OND VALUE OF A MULTI VALUED ATTRIBUTE^1234567890AB
201: CDEFGHIJKLMNOPQRSTUVWXYZ!"#$%&'()*:=-_+;/[^]<>?/..
251: ^ATTRIBUTE VALUES MAY CONTAIN ANY COMBINATION OF L
301: EGAL CHARACTERS, AND ONLY THE NUMBER OF CHARACTERS
351:  ACTUALLY IN THE VALUE WILL BE STORED.  ADDITIONAL
401: Y THE VALUE MAY BE UP TO 32,760 CHARACTERS LONG.^V
451: ALUE FOR ATTRIBUTE 6^ VALUE FOR ATTRIBUTE 7^VALUE F

FID : 1253    LINKS : 0 9327 1252 1
501: OR ATTRIBUTE 8^VALUE FOR ATTRIBUTE 9^VALUE FOR ATT
551: RIBUTE 10^VALUE FOR ATTRIBUTE 11^VALUE FOR ATTRIBU
601: TE 12^VALUE FOR ATTRIBUTE 13^VALUE FOR ATTRIBUTE 1
651: 4 VALUE FOR ATTRIBUTE 15^VALUE FOR ATTRIBUTE 16^VA
701: LUE FOR ATTRIBUTE 17^VALUE FOR ATTRIBUTE 18^VALUE
751: FOR ATTRIBUTE 19^VALUE FOR ATTRIBUTE 20^VALUE FOR
801: ATTRIBUTE 21^VALUE FOR ATTRIBUTE 22^VALUE FOR ATTR
851: IBUTE 23^VALUE FOR ATTRIBUTE 24^VALUE FOR ATTRIBUT
901: E 25^VALUE FOR ATTRIBUTE 26^VALUE FOR ATTRIBUTE 27
951: ^VALUE FOR ATTRIBUTE 28^VALUE FOR ATTRIBUTE 29^VAL
```

Primary Space

```
FID : 9327   LINKS : 0 9331 1253 0
1001: UE FOR ATTRIBUTE 30^VALUE FOR ATTRIBUTE 31^VALUE F
1051: OR ATTRIBUTE 32^VALUE FOR ATTRIBUTE 33^VALUE FOR A
1101: TTRIBUTE 34^VALUE FOR ATTRIBUTE 35^VALUE FOR ATTRI
1151: BUTE 36^VALUE FOR ATTRIBUTE 37^VALUE FOR ATTRIBUTE
1201:  38^VALUE FOR ATTRIBUTE 39^VALUE FOR ATTRIBUTE 40^
1251: VALUE FOR ATTRIBUTE 41^VALUE FOR ATTRIBUTE 42^VALU
1301: E FOR ATTRIBUTE 43^VALUE FOR ATTRIBUTE 44^VALUE FO
1351: R ATTRIBUTE 45^VALUE FOR ATTRIBUTE 46^VALUE FOR AT
1401: TRIBUTE 47^VALUE FOR ATTRIBUTE 48^VALUE FOR ATTRIB
1451: UTE 49^VALUE FOR ATTRIBUTE 50^VALUE FOR ATTRIBUTE

                                  <-End of first item.

FID : 9331   LINKS : 0 0 9327 0
1501: 51^VALUE FOR ATTRIBUTE 52^VALUE FOR ATTRIBUTE 53^V
1551: ALUE FOR ATTRIBUTE 54^VALUE FOR ATTRIBUTE 55^VALUE
1601:  FOR ATTRIBUTE 56^VALUE FOR ATTRIBUTE 57^VALUE FOR
1651:  ATTRIBUTE 58^VALUE FOR ATTRIBUTE 59^VALUE FOR ATT
1701: RIBUTE 60^VALUE FOR ATTRIBUTE 61^VALUE FOR ATTRIBU
1751: TE 62^VALUE FOR ATTRIBUTE 63^VALUE FOR ATTRIBUTE 6
1801: 4^VALUE FOR ATTRIBUTE 65^VALUE FOR ATTRIBUTE 66^VA
1851: LUE FOR ATTRIBUTE 67^0033ITEM5^THIS IS AN ITEM WHO
1901: SE ITEM-ID IS (ITEM5)^^
```

Overflow Space

Item Format - Logical

While it is important to understand the item format as described in the previous section, in normal system usage items are always accessed at a more abstract or higher level. Files are identified by a File-name. Within a File, items are referenced by an Item-id. For example, the following statement shows an item in the file 'SAMPLE-FILE' whose item-id is 'ITEMO'. Furthermore, this item has three attributes or lines each with sample data.

```
:COPY SAMPLE-FILE ITEMO (T) (r)

    ITEMO <------------------------------ Item-id
001 LINE 1
002 SMITH, JOHN
003 1234 MAIN STREET
```

Utility processors like COPY and EDIT deal at the file - item - line level. They make no logical distinction in definition between various "lines" in an item other than their implied line numbers. ENGLISH processors, however, add an additional dimension through the use of the dictionary. This dictionary informs them as to the nature of the information stored for each of the attributes. The logical item format is identical for ENGLISH and non-ENGLISH processors as in the case of COPY above. It is the

responsibility of the user to ascertain the further qualifications, if any, of the various attributes. For example, the following is a listing of the item shown above using the ENGLISH List Processor.

```
:LIST SAMPLE-FILE 'ITEMO' ATTRIBUTE-1 NAME ADDRESS (r)

PAGE 1                                          16:40  23 OCT 1973

SAMPLE-FILE ATTRIBUTE-1 NAME..... ADDRESS.....

ITEMO        LINE 1      SMITH, JOHN     1234 MAIN STREET
```

In this example the dictionary defines the second attribute (or line) as 'NAME'. This permits the user to reference his data symbolically, when in fact, the actual data stored on file is the same regardless of the Processor accessing it.

Selecting Modulo and Separation

These are general guidelines in selecting values for the modulo and separation parameters when using the CREATE-FILE processor. The guidelines are derived from the density versus overflow table explained in the next section.

Modulo: is the number of groups in the file. It should be selected with regard to the total number of items that the file is to store. For optimal hashing (the pseudo-random technique of distributing items among the groups), the modulo should be a prime number. As a trade-off between saving storage space and minimizing search-time in a group, the modulo should be such that there will be 10-20 items per group (fewer for large items).

Therefore, $m \approx$ [(Average expected number of items)/15]
m prime

Separation: is the size (in frames) per group. It should be selected with regard to the average size of items that the file is to store. A value should be selected such that 80% of the data in a group is in the "prime" space.

Therefore, $s \approx$ [(Average number of bytes per group)/(.8*500)]
where the average number of bytes per group can be computed from the average item size, and the number of items per group. Separation should be selected after the modulo.

EXAMPLE--

The new NEWAC file (prototype M/DICT) has about 160 items, average item size 30 bytes. Therefore,

$$m = (160/15) = 10.67$$

Selecting m = 11 as a prime number, average number of bytes per group = $\frac{160}{11} * 30 = 436$, and

$s = 436/(.8*500) \simeq 1$ (500 bytes per frame)

Therefore the selected modulo = 11, separation = 1.

Other considerations include the frequency of usage of the file--relatively "static" files can have more items per group; "dynamic" files should have fewer items per group.

Density Versus Overflow

The table overleaf shows the relationship between density and overflow access, where density is the percentage of primary space used, and overflow access occurs when an item is partially or wholly in overflow space.

When an item is updated, it moves to the end of the group. Thus items that are most frequenly updated occur towards the end of the group data. This accounts for the difference in the probability figures for update and retrieval.

# of updates to overflow area per 1000 updates / # of retrievals from overflow area per 1000 retrievals										
# of Items per Group	UTILIZATION									
	10%	20%	30%	40%	50%	60%	70%	80%	90%	100%
1	95 2	181 9	259 9	330 32	393 48	451 65	503 84	551 104	593 125	632 147
2	18 -	62 3	122 8	191 17	264 30	337 47	408 66	475 88	537 112	592 138
3	4 -	23 1	63 4	121 10	191 20	269 34	350 52	430 74	506 100	577 128
4	1 -	9 -	34 2	79 6	143 13	221 25	308 42	397 64	485 90	567 119
5	- -	4 -	19 1	53 3	109 9	185 19	275 35	371 56	468 82	560 112
6	- -	1 -	10 -	36 2	84 6	156 14	247 29	349 49	454 75	554 106
7	- -	1 -	6 -	24 1	65 5	133 12	223 25	330 44	442 69	550 101
8	- -	- -	3 -	17 1	51 3	113 10	203 21	313 39	431 65	547 97
9	- -	- -	2 -	12 1	40 2	97 8	185 18	297 36	421 61	544 93
10	- -	- -	1 -	8 -	32 2	84 6	169 16	283 33	413 57	542 90
15	- -	- -	- -	2 -	13 -	48 2	121 8	238 22	383 44	536 77
20	- -	- -	- -	- -	4 -	24 1	82 5	195 15	354 36	530 69
25	- -	- -	- -	- -	1 -	13 -	57 3	162 11	330 162	527 64
35	- -	- -	- -	- -	- -	4 -	28 1	116 7	293 23	523 56
50	- -	- -	- -	- -	- -	1 -	10 -	72 3	249 16	519 48

Probability of Overflow Using Modulo Addressing

FRAME	MODE	FRAME	MODE
1	DB1	50	LOGOFF
2	TCL-II	51	SYSTEM-SUBS-III
3	DISKFIO-II	52	MSG
4	TCL-INIT	53	
5	TCL-I	54	
6	TERMIO	55	
7	DISKFIO-I	56	
8	SYSTEM-SUBS-I	57	
9	SYSTEM-SUBS-II	58	
10	WRAPUP-I	59	
11	WRAPUP-II	60	
12	WRAPUP-III	61	
13	EDIT-I	62	
14	EDIT-II	63	
15	EDIT-III	64	
16	EDIT-IV	65	
17	DB2	66	
18	DB3	67	
19	DB4	68	
20	DB5	69	
21	DB6	70	
22	GAF	71	
23	PASS1	72	
24	ASTAT	73	
25	MACRO	74	
26	GEN	75	
27	ALIGN	76	
28	PASS2	77	
29	GETOP	78	
30	ADDLAB	79	
31	LOADER	80	
32	MLIST	81	
33	OF2	82	
34	OF1	83	
35	TAPEIO-I	84	
36	TAPEIO-II	85	
37	T-LOAD	86	
38	EBCDIC	87	
39	SORT	88	
40	PROC-III	89	
41	DUMP-II	90	
42	DUMP-I	91	
43	LOGON	92	
44	PROC-I	93	
45	PROC-II	94	
46	DLOAD	95	
47	ABSL	96	
48	DDUMP	97	
49	ABSD	98	

FRAME	MODE	FRAME	MODE
99		148	CARDIO
100		149	
101		150	PROC-IV
102		151	PROC-V
103		152	INPUT
104		153	TAPEIO-III
105		154	
106		155	
107		156	
108		157	
109		158	
110		159	
111		160	ARITH
112		161	
113		162	
114		163	OPNPF
115		164	PQUEUE
116		165	PFILE
117		166	PQEXT-I
118		167	PQEXT-II
119		168	PQEXT-III
120		169	XLOADER
121		170	COREDUMP
122		171	PQEXT-IV
123		172	WSPACES
124		173	
125		174	
126		175	PQEXT-V
127		176	
128		177	
129		178	
130		179	
131	DISC-DIAG	180	
132	DISC-MSG	181	
133		182	
134		183	
135		184	
136		185	
137		186	
138		187	MSETUP
139		188	MSETUP0
140	COPY-I	189	MSETUP1
141	COPY-II	190	MBOOT
142	COPY-III	191	MBUFFERS
143	COPY-IV	192	MMONITOR
144	DISK-CHARGES	193	MMONITORX
145	XREF	194	PIB0
146	CROSS	195	PIB1
147	SEL-RESTORE	196	MONITORY

FRAME	MODE
197	PCB0
198	MMONITORZ
199	MMONITORY/N2
200	
.	
.	
.	
290	BLOCK-LETTERS
.	
.	
.	
399	

CORE-MAP

This table describes the core-map of the system as it is initialized by the cold-start process: A minimum of 16K of core is required; any additional core in the particular hardware configuration is not initialized. The "STATUS" column has one of the following entries: X'80' -- LOCKED IN CORE; X'EF' WRITE TO DISK; X'FF' AVAILABLE.

BUFFER # HEX	BUFFER # DEC	ADDRESS HEX	STAT	FID(U)..(L) HEX	FID(U)..(L) HEX	FID DEC	DESCRIPTION AND PROGRAM-NAME
00	00	0000	-	-	-	-	MONITOR PCB MBOOT
01	01	0200	80	-	-	-	BUFFER TABLES MBUFFERS
02	02	0400	80	-	-	-	MONITOR OBJECT MMONITOR
03	03	0600	80	-	-	-	MONITOR OBJECT MMONITORX
04	04	0800	80	FFFF	FC		PIBS, DEV. 18/19 PIB0
05	05	0A00	80	FFFF	FB		PIBS, DEV. 1A/1B PIB1
06	06	0C00	80	FFFF	FA		MONITOR OBJECT MMONITORY/N1
07	07	0E00	-	FFFF	F9	-	MONITOR OBJECT MMONITORZ
-	-	1000	-	-	-	-	CONFIGURATOR MSETUP0
-	-	1200	-	-	-	-	LITERALS MSETUP1
-	-	1400	-	FFFF	FA		MONITOR OBJECT MMONITORY/N2
-	-	1600	-	-	-	-	DISC DIAGNOSTIC DISC-DIAG
-	-	1800	-	-	-	-	LITERALS DISC-MSG
08	08	1000	EF	0002	00	512	PCB, CHANNEL 0 PCB0
09	09	1200	EF	0000	2F	47	FILE RESTORE ABSL
0A	10	1400	EF	0000	23	35	TAPE I/O TAPEIO-I
0B	11	1600	EF	0000	24	36	TAPE I/O TAPEIO-II
0C	12	1800	80	0000	04	4	INITIALIZATION TCL-INIT
0D	13	1A00	80	0000	06	6	TERMINAL I/O TERMIO
0E	14	1C00	80	0000	07	7	FILE I/O DISKFIO-I
0F	15	1E00	80	0000	08	8	SYSTEM SUBS SYSTEM-SUBS-I
10	16	2000	EF	0000	01	1	DEBUGGER DB1
11	17	2200	EF	0000	02	2	TCL TCL-II
12	18	2400	EF	0000	03	3	FILE I/O DISKFIO-II
13	19	2600	EF	0000	05	5	TCL TCL-I
14	20	2800	EF	0000	09	9	SYSTEM SUBS SYSTEM-SUBS-II
15	21	2A00	EF	0000	0A	10	WRAPUP PROCESSOR WRAPUP-I
16	22	2C00	80	0000	0B	11	WRAPUP PROCESSOR WRAPUP-II
17	23	2E00	EF	0000	0C	12	WRAPUP PROCESSOR WRAPUP-III
18	24	3000	EF	0000	11	17	DEBUG PROCESSOR DB2
19	25	3200	EF	0000	12	18	DEBUG PROCESSOR DB3
1A	26	3400	EF	0000	13	19	DEBUG PROCESSOR DB4
1B	27	3600	EF	0000	14	20	DEBUG PROCESSOR DB5
1C	28	3800	EF	0000	15	21	DEBUG PROCESSOR DB6
1D	29	3A00	FF	0000	-	-	AVAILABLE
1E	30	3C00	FF	0000	-	-	AVAILABLE
1F	31	3E00	FF	0000	-	-	AVAILABLE

The first three buffers are not accessed by the firmware in the virtual mode of operation; thus there is absolute memory protection of core locations 0-X'07FF'. The status fields for these buffers must be set to X'80' (core-locked). There are two buffers containing the PIB's (process identification blocks) that follow the above; their status must also be set to X'80'. Buffers 6 and 7 have additional monitor object code, and must be locked in core. The dummy FID's assigned to buffers 4, 5, 6 and 7 allow software to access these buffers, but provide some measure of access protection.

TABLE OF PRIME NUMBERS LESS THAN 1000

2	101	211	307	401	503	601	701	809	907
3	103	223	311	409	509	607	709	811	911
5	107	227	313	419	521	613	719	821	919
7	109	229	317	421	523	617	727	823	929
11	113	233	331	431	541	619	733	827	937
13	127	239	337	433	547	631	739	829	941
17	131	241	347	439	557	641	743	839	947
19	137	251	349	443	563	643	751	853	953
23	139	257	353	449	569	647	757	857	967
29	149	263	359	457	571	653	761	859	971
31	151	269	367	461	577	659	769	863	977
37	157	271	373	463	587	661	773	877	983
41	163	277	379	467	593	673	787	881	991
43	167	281	383	479	599	677	797	883	997
47	173	283	389	487		683		887	
53	179	293	397	491		691			
59	181			499					
61	191								
67	193								
71	197								
79	199								
83									
89									
97									

SECTION III

DICTIONARIES

INTRODUCTION

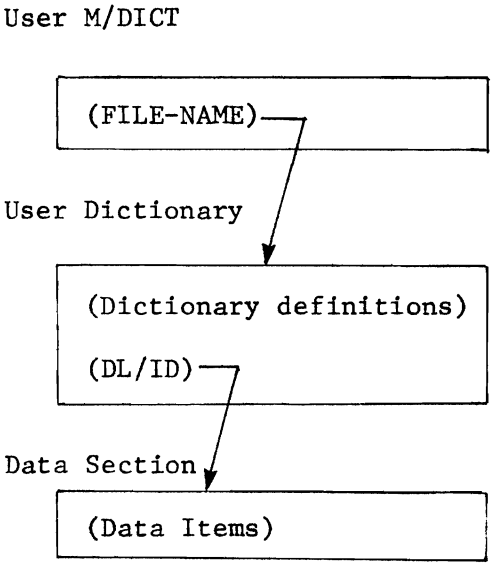
Dictionaries define and describe data within their associated file. Dictionaries exist at several levels within the Reality system; the highest level dictionary is called the System Dictionary (SYSTEM). This dictionary is used for system control, and contains only a pointer to the Accounting file and the names of users who may logon to the Reality system. The next level dictionary is called the Master Dictionary (M/DICT); each user's account has a Master Dictionary associated with it. The Master Dictionary points to (or defines) lower level dictionaries within the user's account.

File Structure

The term "file" as used in the context of the Reality system, refers to a mechanism for maintaining a set of like items logically together. The data in a file is normally accessed via the dictionary associated with it. Since the dictionary is itself a file, the mechanism for accessing items in a dictionary is identical to that for a file.

At the user M/DICT level, a file-definition item (the file name) is a pointer to the dictionary. The dictionary may contain pointers to one or more lower-level data files; thus a two-level structure is usually implied by a file definition item at the M/DICT level. A special item in a dictionary, whose item-id is DL/ID, serves as the file pointer to the data.

If there is no data section corresponding to the dictionary entry in the M/DICT, the DL/ID item may be absent in the dictionary, or it may be present and may point to the dictionary itself; in the latter case the "data" section overlays the dictionary.



This diagram shows the two-stage relationship of the file-name in the master dictionary to the data section. The file-name is a file-definition item (see next section) that points to the dictionary; one of the items in the dictionary is the DL/ID, which again is a file-definition item that points to the data section of the file. Items at each level thus serve to define the structure at the next lower level: the M/DICT describes the user-dictionaries, and each user-dictionary describes its corresponding data section.

Dictionary Inter-relationships

The table on the next page describes the four-level file structure, and dictionary inter-relationships of the Reality system. In addition to the required system dictionaries and data-files, one user account is shown for illustration.

The boxes represent the dictionaries "containing" items; shown are file-definition and file synonym definition items (defined in the next two sections). The full lines from items to boxes represent the "file pointer" nature of file-definition items; the dashed lines represent the linkage between file synonym definitions and their equivalent files.

DICTIONARY ITEM DEFINITIONS

File Definition Items

Each item in a dictionary is classified according to a single character dictionary code (D/CODE) in attribute one of the item. A file-definition item has a D/CODE of "D", and is a pointer to the actual physical location of the file in the virtual (disc) memory. File-definition items are set up during the system File-restore process, and by the CREATE-FILE processor. Values in these items that define the physical file extents should not be altered by the user. There may be more than one file-defining item in a dictionary that points to the same file; but there should not be any such items in other dictionaries that point to the same file--these should be file synonym definitions (see below). If duplicate "D" items exist in other dictionaries, duplicate copies of the file will result on a filesave and restore.

Attributes two through four define the physical extents of the file:

Attribute two:	Contains the BASE FID of the file.
Attribute three:	Contains the MODULO of the file.
Attribute four:	Contains the SEPARATION of the file.

The values for BASE, MODULO and SEPARATION are stored as decimal numerics; the meaning of these fields is described under Data Structures.

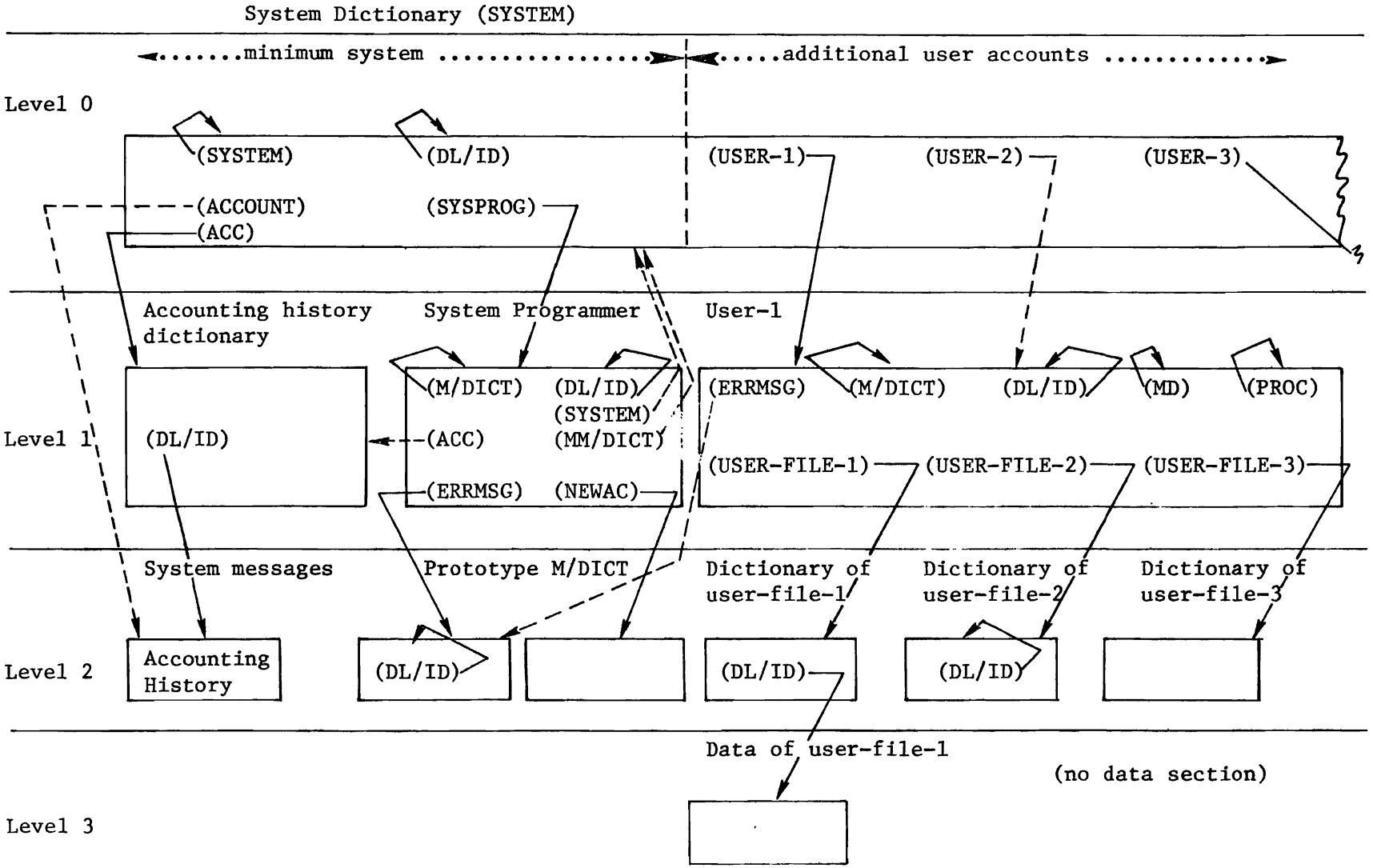
Attributes 5 through 12 of a file-definition item are identical to those described for attribute definition items (see later).

Attribute 13 is an optional Reallocation specification, which allows the reallocation of the physical extents of a file during a system File-restore process. This is the only way in which the physical extents of a file can be altered. The format of this specification is as follows:

(m,s) Where "m" and "s" are decimal numerics specifying the new modulo and new separation parameters of the file. Restrictions on the values of "m" and "s" are as follows:

$0 < m$
$0 < s < 128$

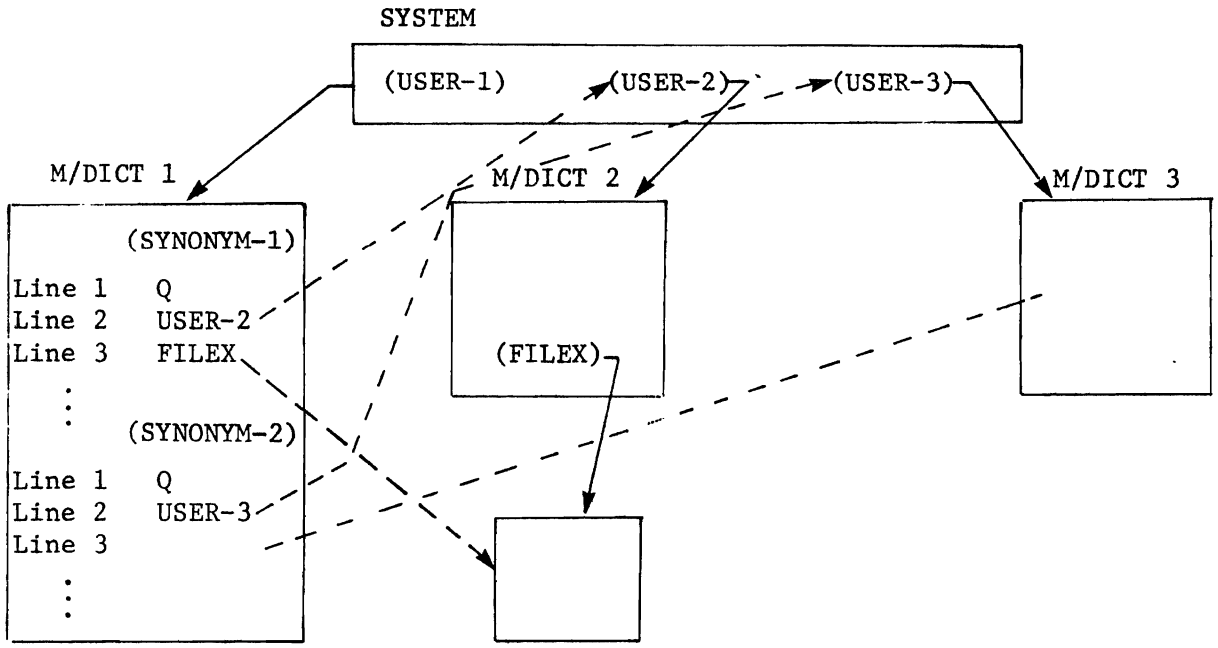
Dictionary Inter-relationships in the Reality System.



Entries in parentheses represent file-definition items in the appropriate master-dictionary or dictionary. Note that USER-FILE-2 and USER-FILE-3 both have no data sectors defined, but that the former has a DL/ID pointing to the dictionary. The entry USER-2 in the SYSTEM, and ERRMSG in the M/DICT of USER-1, are file synonym definitions.

File Synonym Definition Items

A file synonym definition item is distinguished by its having a D/CODE of "Q", and it allows access to files in another user's account. Attribute two of a file synonym definition item contains the name of the account in which the actual file definition is to be found (the account name is an entry in the System Dictionary, SYSTEM); attribute three contains the file-definition item-id to which the synonym equates. If this attribute is null, it is implied that the synonym file is the user's M/DICT. Examples are shown below:



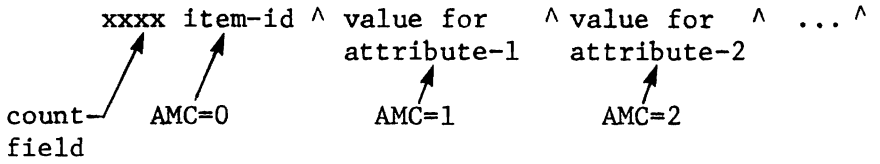
The entry SYNONYM-1 in user-1's M/DICT is equivalent to the file-definition item FILEX in user-2's M/DICT; the entry SYNONYM-2 is equivalent to user-3's M/DICT itself, since it has a null value in attribute 3.

Attributes four through twelve are as defined under Attribute Definition Items (see below).

Attribute Definition Items

These items define the meaning of the various attributes, or fields, in the data items. Each attribute definition item has a value, called the attribute mark count (AMC), which acts as a pointer to the data field defined by it. The AMC indicates the number of attribute marks which precede the value(s) for the attribute being defined by the item. Recalling that the

physical item format consists of the count field immediately followed by the item-id field, followed by an attribute mark, and then the attribute values, each delimited by another attribute mark, it will be seen that the item-id itself may be referenced as having an AMC = 0, the first attribute as having an AMC = 1, and so on:



An attribute-defining item in the dictionary has a D/CODE of "A"; attribute two contains the decimal AMC value described above; attributes three and four are not defined for these items.

The values for attributes five through twelve are as follows; values are optional except where specified:

Attribute 5: Retrieval security lock; used to restrict the retrieval access to this attribute.

Attribute 6: Update security lock; used to restrict the update access to this attribute. These two fields are described in the section under Security.

Attribute 7: Conversion specification; used to perform table look-ups, masking functions, etc. Described under Conversion.

Attribute 8: Correlative specification; used to describe inter-file, and intra-file data relationships. Described under Correlatives.

Attribute 9: Type and Justification; describes the type (alphabetic or numeric), and justification (left or right) for output. A value is mandatory, and may be one of the following:

L	Left justified, no specified type.
LA	Left justified, alphabetic.
LN	Left justified, numeric.
R	Right justified, no specific type.
RA	Right justified, alphabetic.
RN	Right justified, numeric.

Attribute 10: Maximum length; describes the maximum length of values for the attribute; an entry is a decimal numeric, and is mandatory.

Attribute 11: Minimum length; describes the minimum length of input values acceptable on updates to this attribute.

Attribute 12: Pattern edit; describes a pattern editing mask that input values must check against, on updates to this attribute.

Attribute Synonym Definition Items

These items have specific meaning to the ENGLISH processors; they are more fully described in those sections. A synonym definition has a D/CODE of either an "S" or an "X"; attribute two is not used, but normally contains the AMC value of the attribute being defined (mainly to allow sorting by AMC of items in a dictionary). Attribute three contains a "synonym name", a value which lists as a header on ENGLISH LIST or SORT statements; attribute four contains the AMC. Attributes five through twelve are as described above for attribute definition items.

DICTIONARIES

The System Dictionary (SYSTEM)

There is one and only one SYSTEM Dictionary for each Reality system. Other than a pointer to the Accounting file, the SYSTEM should contain only "D" code items, representing user accounts. The LOGON processors use these "D" code items to verify users attempting to logon to the system. Only one "D" code item should be present for each account; if more than one user-name is to be established for the same user-account, the additional names should be file synonym definition ("Q" type) items. The meaning of attributes five through eight is different for both "Q" and "D" Code entries in the SYSTEM; these are described under LOGON/LOGOFF. Entries in this dictionary also completely control the file-save process, whereby the data base is saved on a secondary storage medium.

The Master Dictionary (M/DICT)

There is one M/DICT for each account. The M/DICT, like any other file or dictionary, is made of up items. Some of these items define the attribute format for all dictionaries (D/CODE = "A") and their formats are identical to those for file-dictionaries. The file defining items (D/CODE = "D") point to (or define) the various dictionaries defined for the account.

In addition to those elements in the M/DICT identical to a file dictionary, there are entries which define VERBS, PROCS and various ENGLISH language elements (connectives and BATCH STRINGS). Each of these entries has a coding structure which uniquely identifies it. Please refer to the chapters on TCL, PROC and ENGLISH language for their respective definitions.

All names used as item-id's in the M/DICT must be unique not only within the M/DICT, but also among all file dictionaries.

INITIAL SYSTEM FILES

Certain files are essential to the operation and maintenance of the Reality System. These files are described below.

The Account File

This file contains the accounting history for the system, as well as the entries that describe currently active (logged-on) users. The formats of these entries are described under the LOGON/LOGOFF section. The Accounting file should be cleared periodically to prevent overflow of the file (refer to LOGON/LOGOFF).

The SYSPROG Account

The SYSPROG (System Programmer) account is the only account needed to maintain the Reality System. The system message file (ERRMSG) and the prototype M/DICT (NEWAC) are defined from this account; the former is accessed by all users of Reality to obtain error and informative messages, while the latter is used to create new user M/DICT's.

Also contained in the SYSPROG account are the system-level PROC's which perform the File-save and File-restore functions, the initialization of the accounting file on a cold-start, etc. For this reason, the following two file synonym definition items must be present in the SYSPROG M/DICT.

MM/DICT : synonym to the SYSTEM dictionary.

ACC : synonym to the accounting history file.

See System Maintenance for a full description of entries in this account.

The ERRMSG File

This dictionary defined from the SYSPROG account, contains the system messages. It is mandatory that every user account have a "Q" type entry called ERRMSG which points to the ERRMSG file in the SYSPROG account. (This is accomplished by the CREATE-ACCOUNT PROC.)

Entries in the ERRMSG file are listed in the System Messages section; they consist of both error messages as well as informative messages.

The NEWAC File

This dictionary is defined from the SYSPROG account, and is a prototype M/DICT that is used as a model from which a new user's M/DICT is created.

It contains the standard set of VERBS, PROCS, and ENGLISH language elements. Entries are listed in the System Maintenance section.

Summary of Dictionary Item Definitions

Attribute Number	M/DICT Name	File Definition	Synonym to a File-Definition	Attribute Definition	Synonym to Attribute Definition
1	D/CODE	D	Q	A	S or X
2	F/BASE or A/AMC	Base FID of file	Account-name	AMC	[AMC]
3	F/MOD or S/NAME	Modulo of file	Synonym file-name	Not used	Synonym name
4	F/SEP or S/AMC	Separation of file	Not used	Not used	AMC
5	L/RET	- - Retrieval lock code(s) - - - - - - - - - - - -			
6	L/UPD	- - Update lock code(s) - - - - - - - - - - - - - -			
7	V/CONV	- - Conversion specification(s) - - - - - - - - -			
8	V/CORR	- - Correlative specification(s) - - - - - - - -			
9	V/TYPE	- - Justification & type-code - - - - - - - - -			
10	V/MAX	- - Maximum field length - - - - - - - - - - - - -			
11	V/MIN	- - Minimum field length - - - - - - - - - - - - -			
12	V/EDIT	- - Pattern edit for updating values - - - - - -			
13	F/REALLOC	Reallocation Specification - - - - -		Not Used - - - - -	

EXAMPLES--

```
:COPY DICT M/DICT M/DICT MURTHI PREMIUM 16 (P) (r)

    M/DICT                          Item-id (file-name).
001 D                               D/CODE = "D"; File Definition Item.
002 14933                           Base FID of file.
003 13                              Modulo of file.
004 1                               Separation of file.
005 /;P123                          File access protect code; retrieval
006 UPDATE*LOCK!                    Update lock-code.          lock-code
007                                 Conversion (null).
008                                 Correlatives (null).
009 L                               Left justified dictionary items.
010 13                              Maximum field length.
011 1                               Minimum field length.
012                                 Pattern edit mask (null).
013 (11,1)                          Reallocation parameters.

    MURTHI                          Item-id (file-name).
001 Q                               File Synonym Definition Item.
002 CHANDRASHEKAR                   User name in MM/DICT.
003                                 Null file-name; therefore M/DICT.
004
005
006
007
008
009 L
010 8
011 1

    PREMIUM                         Item-id (attribute name).
001 A                               Attribute Definition Item.
002 99                              Attribute Mark Count (99-th. field).
003
004
005
006
007 MD26                            Conversion specification.
008 F;7;8;*                         Correlative specification.
009 R                               Right justified field.
010 10
011 1

    16                              Item-id (attribute synonym).
001 S                               Attribute Synonym Definition Item.
002 16                              For sorting purposes only.
003 AGENCY NAME....                 Synonym name (header name).
004 16                              Attribute Mark Count.
005
006
007 TAGENT-NO;V;;2                  Conversion specification.
008
009 LA                              Left justified; alphabetic field.
010 4
```

```
:SORT DICT AGENCY-NO D/CODE A/AMC S/NAME S/AMC V/CONV V/CORR  V/TYPE V/MAX V/MIN V/EDIT F/REALLOC BY
        D/CODE BY A/AMC DBL-SPC (r)

AGENCY-NO. D/CODE.. A/AMC S/NAME..... S/AMC V/CONV..... V/CORR.... V/TYPE V/MAX V/MIN V/EDIT F/REALLOC

RATE       A       01                          MD4                  R      7     7
DESC       A       02                                               L      50    1
DESCRIPTIONA       02                                               L      50    L
TAX        A       03                          MD23                 R      9     1
DL/ID      D       03259 0001          020                          L      4     4
1          S       01    RATE          1       MD4                  L      7     7
2          S       02    DESCRIPTION   2                            L      50    L
3          S       03    TAX           3       MD23                 R      11    1
```

The above is a listing of the AGENCY-NO dictionary; the fields L/RET and L/UPD have been suppressed. Note that the Attribute Definition Items 'DESC' and 'DESCRIPTION' reference the same field (both have an A/AMC of two); thus they can be said to be "synonyms" to each other. Though the items DESC and DESCRIPTION are identical, they may have different entries under, say, the V/MAX for formatting or other purposes; there is no restriction on the number of such synonyms in the dictionary. The values under the columns A/AMC, S/NAME and S/AMC for the File Definition Item 'DL/ID', are actually the values of the base FID, modulo and separation of the data-file referenced by this dictionary. Leading zeroes in numeric fields are not necessary; they are present mainly for formatting purposes.

SECTION IV

TERMINAL CONTROL LANGUAGE

INTRODUCTION

The Terminal Control Language (TCL) is the primary interface between the terminal user and the various Reality processors. Most processors are evoked directly from TCL by a single statement, and return to TCL after completion of processing. Some processors, the EDITOR for example, retain control of the terminal until explicitly exited, at which point they return control to TCL. TCL prompts the user by typing a colon(:). This is referred to as the "TCL prompt character". Statements are constructed by typing a character at a time from the terminal until the "CARRIAGE-RETURN" or "LINE-FEED" key is depressed. At that time the entire line is processed by TCL.

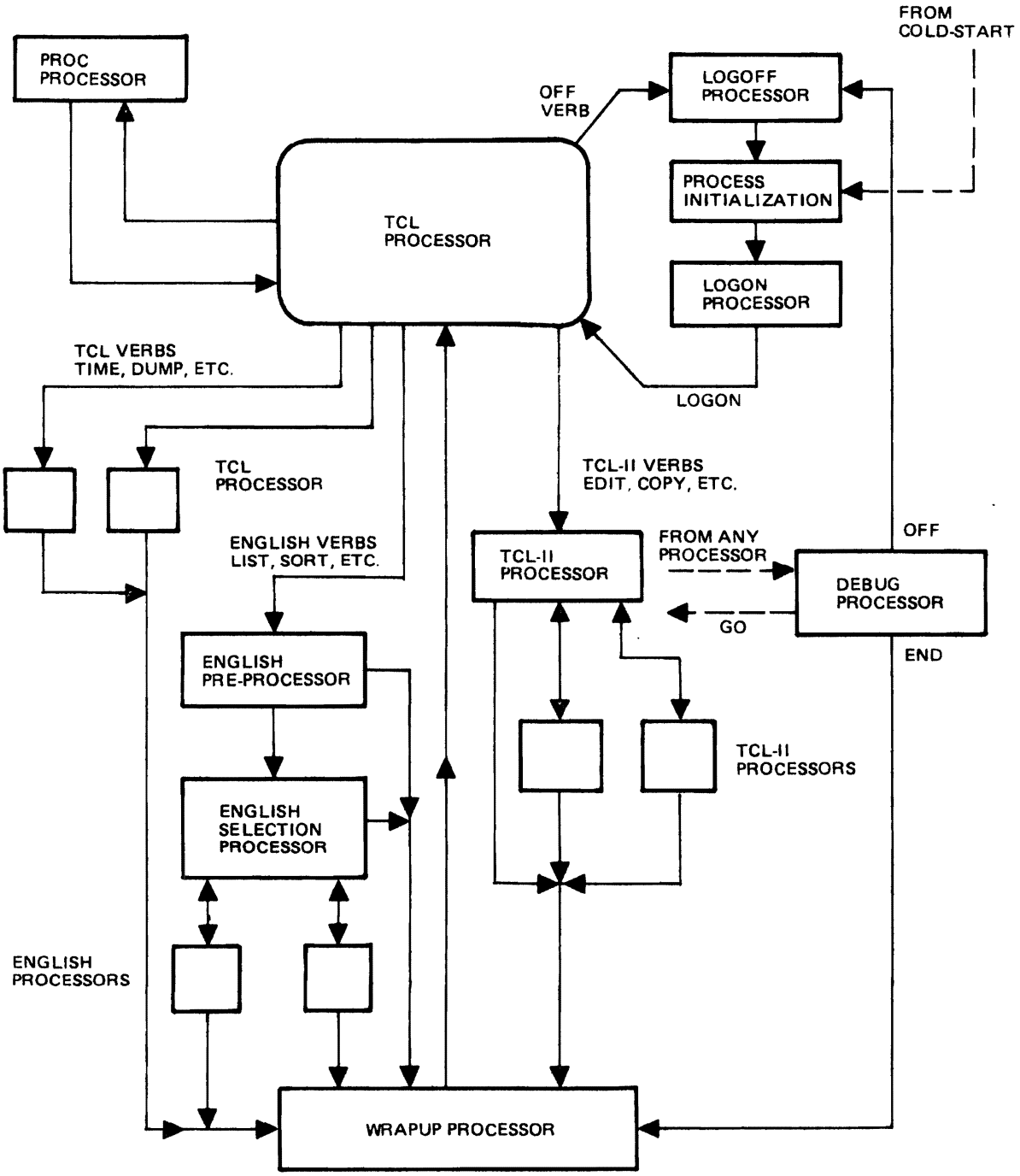
EXAMPLE--

```
:COUNT EVERY ITEM IN THE ACCOUNT FILE (r)
```

Input Statements

TCL works on one statement at a time. A statement may be comprised of multiple lines. However this statement must begin with a verb and may contain only one verb per statement. Reality operates in the full-duplex mode of communication with each user's terminal. Full-duplex means that data is being transmitted in both directions simultaneously between the terminal and the computer. Additionally, Reality operates in what is known as an "Echo-Plex" environment. This means that each data character input by the terminal is echoed back to the terminal by the computer. The user is assured therefore that the data character displayed on the terminal is identical to the data character stored by the computer. TCL passes only complete input lines to be processed by the software. The user fully composes his input statement with no action being taken until TCL detects either a Carriage-Return or a Line-Feed. If no Carriage-Return (or Line-Feed) is detected data characters will be assembled into a statement in the user's input buffer up to 140 characters at which time TCL will automatically generate a Carriage-Return. In addition to the standard ASCII (96) character set recognized by TCL, special operations are performed when control characters are detected. The control characters listed below perform editing functions; all other control characters are deleted from the input line that is passed to lower level processors, but remain in the original input line.

OVERALL VIEW OF SYSTEM SOFTWARE LINKAGE



Control Character	Function
Carriage Return or Line Feed	Terminates the input statement and initiates processing.
Backspace (Control H)	Deletes the last character typed from the input buffer.
Cancel (Control X)	Deletes the entire line currently being typed from the input buffer.
Retype (Control R)	Causes the entire line currently being built in the input buffer to be retyped.
Continuation (Control-Shift 0)	Permits continuation to a second input line; must be immediately followed by a carriage return or line feed.

Note: The continuation character is only effective from TCL; all other characters may be used at any time for any processor requesting input.

TCL Processing

The TCL expects the first parameter of a statement to be a verb. There are three types of verbs in Reality:

- ENGLISH verbs
- TCL-II verbs
- TCL-I verbs

A summary of standard Reality verbs is provided below. One of the powerful features of Reality is the ability to customize the vocabulary for each user. Since verbs reside in the individual user's Master Dictionary (M/DICT) the vocabulary may be added to or deleted from without affecting the other users. (In addition to adding or deleting verbs, an unlimited number of synonyms may be created for each verb.)

Verb	Type	Function
ADDD	TCL-I	Add decimal.
ADDX	TCL-I	Add Hexadecimal.
AS	TCL-II	Assembles source code.
ASSIGN	TCL-I	Assign print spooler device.
B/ADD	TCL-II	File update via batch-string.
B/DEL	TCL-II	File delete via batch-string.
BLOCK-PRINT	TCL-I	Send block characters to spooler.
BLOCK-TERM	TCL-I	Print block characters on terminal.
C-READ	TCL-II	Read cards and append them to an existing file item.
CLEAR-FILE	TCL-I	Remove all file items from a file or dictionary.
COPY	TCL-II	Copy data files and dictionaries.
COREDUMP	TCL-I	Produce formatted output from a binary dump of core.
COUNT	ENGLISH	Count occurrences of file items.
CREATE-FILE	TCL-I	Create a new file.
CROSS-INDEX	TCL-II	Create a cross index of assembly language programs.
DELETE-FILE	TCL-I	Delete an entire file.
DIVD	TCL-I	Divide decimal.
DIVX	TCL-I	Divide Hexadecimal.
DTX	TCL-I	Convert from decimal to hexadecimal.
DUMP	TCL-I	Dump virtual frames to terminal.
EBTPRD	TCL-II	Read records from tape into file items.
ED	TCL-II	Same as EDIT.
EDIT	TCL-II	Evoke the editor processor.
EJECT	TCL-I	Eject line printer pages.

Verb	Type	Function
FORM	TCL-I	Set form alignment for print spooler.
GROUP	TCL-II	Provide file usage statistics on groups.
I-DUMP	ENGLISH	Dump to terminal in T-DUMP format.
ISTAT	ENGLISH	Histogram file hashing.
ITEM	TCL-II	Provide usage statistics on file items.
KILL	TCL-I	Abort current spooler output.
LIST	ENGLISH	Print selective report output.
MESSAGE	TCL-I	Inter-user communication.
MLIST	TCL-II	List assembly source code.
MLOAD	TCL-II	Load assembly object code.
MSG	TCL-I	Same as MESSAGE.
MULD	TCL-I	Multiply decimal.
MULX	TCL-I	Multiply hexadecimal.
MVERIFY	TCL-II	Verify assembled program against loaded program.
OFF	TCL-I	Terminate session-logoff the system.
P	TCL-I	Inhibit printing at terminal.
P-ATT	TCL-I	Attach line printer.
P-ATT-KILL	TCL-I	Unconditionally detach line printer from any line.
P-DET	TCL-I	Detach line printer.
P-STAT	TCL-I	Print line printer status.
POVF	TCL-I	Print overflow parameters.
PRINT-HOLD	TCL-I	Send hold file to line printer.
PRINT-QUE	TCL-I	Print hold file queues.
SEL-RESTORE	TCL-II	Selective restore from save tape.
SELECT	ENGLISH	Select file items for subsequent command.

Verb	Type	Function
SORT	ENGLISH	Print ordered report output.
SSELECT	ENGLISH	Select and sort file items for subsequent command.
STAT	ENGLISH	Print attribute statistics.
SUBD	TCL-I	Subtract decimal.
SUBX	TCL-I	Subtract Hexadecimal.
SUM	ENGLISH	Total attribute values.
T-ATT	TCL-I	Attach magnetic tape unit.
T-BCK	TCL-I	Backspace tape.
T-DET	TCL-I	Detach magnetic tape unit.
T-DUMP	ENGLISH	Dump file items to tape.
T-FWD	TCL-I	Forward-space tape.
T-RDLBL	TCL-I	Read tape Label.
T-LOAD	TCL-II	Load file items from tape.
T-READ	TCL-I	Read one record from tape.
T-REW	TCL-I	Rewind magnetic tape.
T-WEOF	TCL-I	Write EOF on tape.
TERM	TCL-I	Set terminal characteristics.
TIME	TCL-I	Print time and date.
WHO	TCL-I	Print the line number and account name to which the terminal is logged on.
X-REF	TCL-I	Create a cross-reference of assembly programs.
XTD	TCL-I	Convert from hexadecimal to decimal.

TCL Statement Parsing

TCL copies characters from the terminal into a second buffer performing the following processing:

- The first word is assumed as the VERB and looked-up in the user's Master Dictionary (M/DICT), but not copied.
- Redundant blanks surrounding all words in the statement are deleted.
- Character strings surrounded by single or double quotes (' ") are identified and copied verbatim, including redundant blanks.

Statement Formats

All statements processed by TCL must begin with a verb. The syntax of the statement is dependent on the type of verb used.

ENGLISH Verbs

Statements are free-form and may use any combination of conditional constraints such as relational and Boolean operators. The form is:

Verb file-name item-list selection-criteria output-specification

In ENGLISH the words after the verb may be arranged in any sequence that makes sense to the user. (See ENGLISH section for further details.)

TCL-II Verbs

Statements are more restricted. Selection-criteria and output-specification are not allowed by TCL-II verbs. The file name (or DICT file-name) must immediately follow the verb. Item selection is restricted, since each item-id must be uniquely named in the statement, or, alternately all items may be specified (by use of the asterisk*). The advantage gained by this restricted format is an enhancement in processing speed since statement parsing is quicker.

TCL-II verbs use the following formats:

verb file-name item-list (option parameter string)

verb DICT file-name item-list (option parameter string)

Item-list format: The item-list is made up of one or more item-ids, separated by one or more blanks. If an item-id contains embedded blanks or parentheses it must be surrounded by single quotes. All items in a file may be specified by using an asterisk (*) as the item-list.

Option-string format: The option parameter string is enclosed in parentheses. This string is passed to the TCL-II processor and its contents are a function of the particular verb.

Option parameters are either single characters, A through Z, or the numeric option n-m; multiple options are separated by commas.

Interaction of TCL-II Verbs with the SELECT Verb

The full ENGLISH selection criteria may be used in conjunction with TCL-II verbs. This may be done by using the "SELECT" verb to select items from a file (refer to ENGLISH section); when the message indicating the number of items selected is returned, the TCL-II statement may be entered, omitting the item-list. The previously selected list of items will then be used by the TCL-II verb. This capability permits, for instance, selective editing or copying of a file.

EXAMPLE--

```
:SELECT SYSTEM-MODES WITH CLASS = "%SYSTEM MODE" (r)

18  ITEMS SELECTED.
:MLIST SYSTEM-MODES (P,M) (r)          (Note item-list missing)
```

TCL-I Verbs

Verbs which have a code of other than "2" or "35" in line two of their M/DICT entry are known simply as TCL-I verbs. When TCL identifies a verb it exits immediately to the entry point specified in line two of the verb defining item.

Interrupting Processing

The CPU processing can be interrupted at any time by depressing the BREAK key on the terminal (INT on some terminals). This causes an interrupt in the current processing, and an entry to the DEBUG state. This entry is signalled by the message "I x.d" where "x" and "d" describe the location of the point of interruption (refer to DEBUG documentation for details); and input is then requested by the DEBUG prompt character, the exclamation point (!).

For users with system privileges level zero and one, the following are the only DEBUG facilities available.

P Print on/Print Off;

Each entry of P switches from print suppression to print non-suppression and back; the message OFF is returned if output is now suppressed; ON if it is now resumed. Useful to limit output at the terminal. (Also refer to P verb in TCL).

G Go; causes resumption of process execution from the point of interruption.

Note: G cannot be used if a process ABORT condition caused the entry to DEBUG.

END Terminates current process and causes an immediate return to TCL.

OFF Terminates current process and causes the user to be logged off the system.

Note that depressing the BREAK key when in the terminal input or in the output mode will cause a loss of up to 16 characters. If in the input mode, the retype-line character (control-R) should be used to check the loss of data after returning from DEBUG via the G command.

```
:LIST SYSTEM-MODES FRAME HDR-SUPP (r)        ENGLISH LIST - statement.

SYSTEM-MODES..... FRAME.....                 Listing output from system

WSPACES                FRAME 172
EDIT-I                 FRAME 013
PQUEUE/1200            FRAME 164
WRAPUP-II                                    BREAK key depressed
I 6.1A3                                      Interrupt message
!P (r) OFF                                   Turn Print off
!G (r)                                       Go (resume execution without
                                             printing) BREAK key pressed
I 3.FB                                       BREAK key pressed
!P (r) ON                                    Turn Print back on
!G (r)                                       Go
DB3                    FRAME 018
DB4                    FRAME 019
TAPEIO-II              FRAME 036             BREAK key pressed
DB5                                          Terminate execution
I 6.137
!END (r)
                                             Back to TCL
:
```

Processing Aborts

On encountering one of the hardware abnormal conditions, the system will trap to the DEBUG state with a message indicating the nature and location of the abort. If the user has system privileges, level zero or one, he must type END or OFF to exit from the DEBUG state. The hardware abnormal conditions are described in the section DEBUG.

TCL Verb Definition

All verbs are defined as an item in the M/DICT. These items have as their item-id the name of the verb. The second attribute in the item defines the processor entry point to which TCL passes control. The attributes used and their meanings are given below.

Attribute	Description
0	item-id (verb name)
1	"Pc" where "P" identifies the M/DICT item as a verb. The single character "c" is passed to the defined processor. (Note: if c = "Q" the item is a PROC and not a verb)
2	Transfer control program identification (mode-id or address) 2 = TCL-II verb 35 = ENGLISH verb XXXX = TCL verb
3	Secondary transfer point (address)
4	(Tertiary transfer point (address)
5	TCL-II parameter string. These parameters govern treatment of the items retrieval by TCL-II to be passed to the processor whose entry point is defined in attribute three. Parameter meanings are: C-copy item to a work area P-print item-id if item string = "*" (all items) U-items will be updated by processor N-okay if item is not on file Z-final entry required on eof F-pick up file parameters only (ignore item-list)

EXAMPLES--

ENGLISH Verb

```
     LIST
001  PA
002  35
003  4D
```

TCL-II Verb

```
     MLIST
001  PY
002  2
003  20
004
005  CP
```

TCL Verb

```
     TIME
001  PZ
002  3033
```

Section V

STORED PROCEDURES (PROC)

INTRODUCTION

An integral part of the Reality Computer System is an ability to define stored procedures called PROCs. A PROC provides the applications programmer a means to catalog a highly complex sequence of operations which can be evoked from the terminal by a one word command. Any operation that can be executed by the Terminal Control Language (TCL) can be performed in a PROC. This usage of a PROC is quite similar to the use of a Job-Control-Language (JCL) in some large-scale computer systems. The PROC language in Reality is more powerful however since it has "conditional" capabilities and can be used to inter-actively prompt the terminal user. A PROC can test and verify input data as they are entered from the terminal keyboard. These input data are stored in a stack which is ultimately passed to some other processor such as the EDITOR or the BATCH-STRING processor.

A PROC is stored as an item in a dictionary or data file. The first attribute value (first line) of a PROC is always the literal PQ. This specifies to the system that what follows is to be executed by the PROC processor. All subsequent attribute values contain PROC statements that serve to generate TCL commands or insert parameters into the stack for the interactive processors. PROC statements consist of an optional numeric label, a one or two character command and optional command arguments. PROC statements are executed interpretively by the PROC processor.

PROC Execution

A PROC stored as an item in the user's Master Dictionary (M/DICT) is executed or evoked in the TCL environment by typing: the item-id of the PROC, any optional parameters and a carriage return.

EXAMPLE--

```
:LISTU (r)
CHANNEL PCB-FID NAME..... DATE..... TIME..

THREE   0260    KARDEX   14 FEB 1974   12:38

ELEVEN  0360    EARL     14 FEB 1974   14:01

TWO     0240    EARL     14 FEB 1974   14:02

ZERO    0200    SYSPROG  14 FEB 1974   14:15
```

PROCs also have the ability to pass arguments to the TCL level process.

EXAMPLE--

```
:LISTDICTS POLICY (r)
POLICY.............. D/CODE.. A/AMC V/CORR.... V/CONV..... V/TYP V/MAX

AUDIT-PERIOD          A        01                          L     4
POLICY-PERIOD-FROM    A        02             D            L     10
POLICY-PERIOD-TO      A        03             D            L     11
EXPIRES               A        03             D            L     12
```

Where "LISTDICTS" is the name of a PROC and "POLICY" is the argument.

PROCs can be used to interactively prompt data entry and to verify the format of the data.

EXAMPLE--

```
:ENTER-DATA (r)
PART-NUMBER   ? = 3215-19 (r)
DESCRIPTION   ? = TRANSISTOR (r)
QUANTITY      ? = FIFTY (r)
ERROR-NUMERIC DATA ONLY!!
QUANTITY      ? = 50 (r)
BIN-LOCATION  ? = (r)
```

In the previous example the input data is stored in a stack which will be passed to the BATCH-STRING processor to update the file.

Once a PROC is evoked, it remains in control until it is exited. When the PROC temporarily relinquishes control to a processor such as the EDITOR or a user supplied subroutine, it functionally remains in control since an exit from the called processor returns control to the PROC. TCL only regains control when the PROC is exited explicitly, or when all of the lines in the PROC have been exhausted.

PROC Link Command

One PROC can evoke another PROC with the PROC link command. Control is passed to the first statement of the requested PROC, which may be in any dictionary or data file. This allows the storage of PROC (except the LOGON PROC) outside the M/DICT. Also, large PROCs can be broken into smaller PROCs to minimize processing time.

The format of the PROC link command is:

```
( [DICT] filename [item-id])
```

where the filename and item-id indicate the PROC to be evoked. If the item-id is null, the item-id is taken from the current position of the current input buffer.

EXAMPLE--

Assume that the PROC 'LISTU' has been moved to a file called PROCLIB; then the LISTU PROC in each user's M/DICT can be:

```
LISTU

001 PQ
002 (DICT PROCLIB LISTU)
```

and 'LISTU' in the file PROCLIB is exactly as it used to be originally in the M/DICT.

EXAMPLE--

Assume that a PROC called 'EXECUTE' will be used to execute any one of a series of PROCs in a file called PF, as specified by the user typing in a single-character alphabetic code:

```
EXECUTE

001 PQ
002 OPLEASE INPUT CODE +
003 IN?
004 IF A = (1A)  (PF)
005 XILLEGAL RESPONSE.
```

If for example, the user's response to line 3 is "D", Line 4 will transfer control to the item 'D' in the file PF.

All input and output buffers remain unchanged when the linkage takes place. Note that line one of the linked-to item is always skipped, even if the item is not in the M/DICT, to maintain compatibility with M/DICT PROCs.

THIS PAGE INTENTIONALLY LEFT BLANK

Summary of PROC Commands

INPUT COMMANDS

IN	Input from terminal to secondary input buffer.
IP	Input parameters from terminal to either input buffer.
IT	Input from tape to primary input buffer.

OUTPUT COMMANDS

D	Display parameters from either input buffer to terminal.
O	Output string to terminal.

BRANCHING COMMANDS

IF	Conditional statement.
GO,G	Unconditional branch.

BUFFER COMMANDS

A	Move argument from input buffer to output buffer.
B	Back up the pointer of the input buffer.
BO	Back up the pointer of the output buffer.
F	Forward space the pointer of the input buffer.
H	Move a Hollerith string to the output buffer.
IH	Move a Hollerith string to the input buffer.
RI	Clear input buffer.
S	Position the Input Buffer pointer and optionally select the primary input buffer.
STON,ST ON	Stack-on selects secondary output buffer
STOFF,ST OFF	Stack-off selects primary output buffer.
+	Add decimal number to parameter in input buffer.
-	Subtract decimal number from parameter in input buffer.

EXIT COMMANDS

U Exit to user supplied subroutine.

X Exit back to TCL level.

PROCESS COMMANDS

P Process the primary and output buffer.

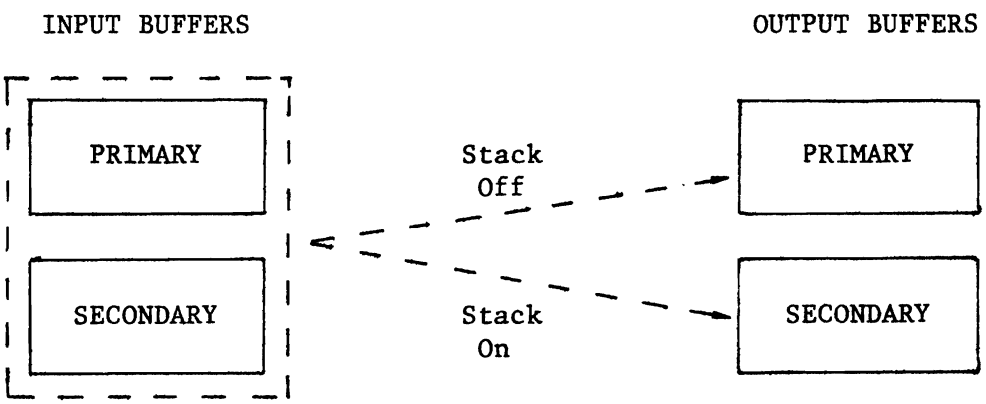
PP Print the primary and secondary output buffers to the terminal before processing.

COMMENT COMMAND

C Comment, statement is not executed by the PROC Processor.

Input/Output Buffer Operation

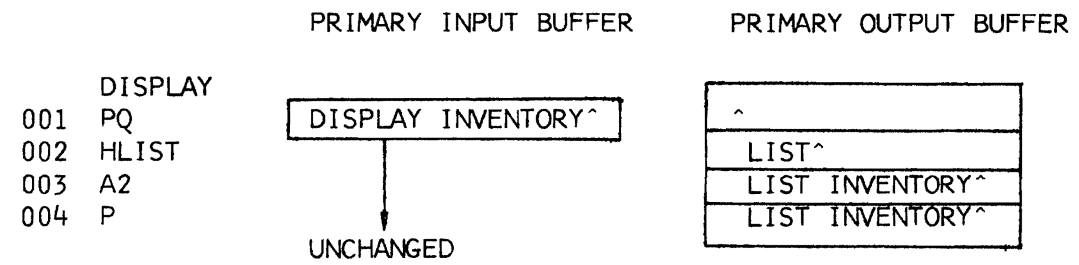
The operation within a PROC is essentially a process of moving data from either of two input buffers to one of two output buffers.



The primary input buffer holds the data which evoked the PROC. These data are the PROC name and any optional arguments.

EXAMPLE--

:DISPLAY INVENTORY (r)



The primary output buffer stores the command which is ultimately passed to the TCL processor for execution. Consequently, any operation which can be performed at the TCL level from the terminal can be performed inside a PROC.

The primary input buffer contains the PROC name and any optional parameters, exactly as they were entered when the PROC was evoked. The contents remain the same throughout execution of the PROC unless explicitly modified using one of the following commands: IP, IT, IH, RI, +, -.

The secondary input buffer is used to hold input from the terminal as a result of the "IN" command. The data in this buffer is volatile and is not available after an "S(m)" or subsequent "IN" command. Usually the data in this secondary input buffer will be checked for correctness and moved to the secondary output buffer which is also known as the "STACK". Parameters will be placed in the stack until the data input for the item is complete, at which time control will be passed to the primary output buffer. The statement which resides in the primary output buffer will be executed at the TCL level and the data in the secondary output buffer (if any) will be used to feed processors such as BATCH-STRING or EDITOR. The stack is used to store parameters for those processors which explicitly request input. The stack may contain one or more lines of data. When the process specified by the primary output buffer is completed, control returns to the PROC at which time new data may be removed to the output buffers.

EXAMPLE--

```
:NEW-PART (r)
PART-NO.
?33451 (r)
VENDOR
?SMITH\CO. (r)
BIN LOCATION
?A2777 (r)
BAD BIN NUMBER
BIN LOCATION
?AA277 (r)
```

PRIMARY INPUT BUFFER

```
NEW-PART^
```

SECONDARY INPUT BUFFER

```
^
33451^
SMITH\CO.^
A2777^
AA277^
```

UNCHANGED

PRIMARY OUTPUT BUFFER

```
^
33451
33451 ^
33451 SMITH\CO.^
33451 SMITH\CO. ^
33451 SMITH\CO. AA277^
33451 SMITH\CO. AA277←^
33451 SMITH\CO. AA277←←^
B/ADD MD :ENT^
B/ADD MD :ENT←^
```

SECONDARY OUTPUT BUFFER

```
^
```

```
      NEW-PART
001   PQ
002   OPART-NO.
003   IN?
004   IF A = (5N) GO 21
005   X-BAD PART NUMBER-
006   21 ST ON
007   A
008   HⒷ
009   22 OVENDOR
010   IN?
011   IF A GO 23
012   GO 22
013   23 A
014   HⒷ
015   24 OBIN LOCATION
016   IN?
017   IF A = (2A3N) GO 25
018   OBAD BIN NUMBER
019   GO 24
020   25 A
021   H<
022   H<
023   ST OFF
024   HB/ADD MD :ENT
025   P
```

PROC Commands

Typically, PROCs are created using the EDITOR. PROCs are executed interpretively and therefore require no compilation phase. Care should be taken not to modify a PROC if another user is currently executing that PROC, since his PROC instruction pointer will remain the same, while the PROC itself may be in a different position after it is filed, causing unpredictable results.

The first line of any PROC must contain the code "PQ". The following lines may contain any legitimate PROC commands. There is no limit to the number of lines in a PROC. However, each line may contain only one statement and must begin in column one. Statement labels are optional and consist of a numeric string (leading zeros are ignored); for example, 01,02 . . . 20,21,22 . . . 100,200,300,400 . . . 1050,1051 . . . Any PROC statement may contain a label. This label serves to uniquely identify its associated PROC command. Labels are used for branching and looping within the PROC. The PROC command begins one blank beyond the label identifier. Multiple label definitions cause no errors; however, only the first definition will be used as the destination of any control transfers.

PROC Command Format

- A

A Ac Ap or Acp A(m,n) or Ac(m,n) A(m) or Ac(m) A(,n) or Ac(,n)	Where p is parameter count c is non-numeric character and not left parenthesis m is column number n is character count

Parameter Insertion Command - This command picks up a parameter from the input buffer and moves it to the output buffer. Either the primary or secondary INPUT buffer may be the source and either the primary or secondary OUTPUT buffer may be a destination. The buffers selected depend on events prior to the occurrence of the "A" command.

A single character "c" immediately following the "A" will be used to surround the parameter if the stack is off (primary output buffer selected). This is useful for picking up item-id's which require single quotes and values which required double quotes for processing by the ENGLISH Language Processor.

EXAMPLE--

```
A'        '1242-L0'        '5996-4'            'JONES'
A"        "250.00"         "100"       "15"        "999.99"
```

However, the character "c" may not be a left parenthesis, or a numeric.

If the parenthetical specification "(m,n)" is not present, the parameter is obtained from the current position of the input buffer pointer. Leading blanks are deleted from the parameter in the input buffer and the insertion terminates on the first blank it encounters.

If the optional "(m,n)" is specified, the parameter is always obtained from the primary input buffer. If the secondary input buffer was selected, use of this option causes a switch back to the primary input buffer. The m parameter positions the input buffer pointer to be set to the m'th column in the primary input buffer. "m" may be used by itself using the form A(m). In this case the parameter is obtained from the m'th column and continues until the first blank is encountered. If the "n" parameter is specified, exactly "n" characters are used in the parameter including any embedded blanks.

"n" may be used by itself. Using the form A(,n) the next "n" characters are taken starting at the current position of the input pointer.

Using the form "Ap" the data is obtained from the p'th field, where the fields are separated by blanks.

Multiple data parameters may be moved into the primary output buffer with a single "A" command if these data are separated by semi-colons. This implicit loop can be used only with the primary output buffer (not the stack).

EXAMPLE--

A'

Either input buffer	Primary output buffer
351;427;926;852 ...	'351''427''926''852'

A"

Either input buffer	Primary output buffer
250;1.95;29.95;1000.50;1	"250""1.95""29.95""1000.50""1"

- B

Causes the input pointer for the currently active input buffer to be spaced backward over one parameter. If the input buffer pointer is currently at the beginning of the buffer, this command has no effect.

- BO

Causes the output pointer for the current output buffer pointer to be moved backward over one parameter. If the output buffer pointer is currently at the beginning of the buffer, this command has no effect.

- C

Comment - The remainder of the statement following the "C" is ignored by the PROC. This provides a means of placing comments within the body of the PROC.

- D
  D(m)
  Dp

"D" displays the field from the current buffer pointer to the next blank on the terminal, "D(m)" displays the field starting at the mth column of the current input buffer. "Dp" displays the pth parameter. DO displays the entire contents of the buffer. The buffer pointer is not changed. A plus sign (+) may be appended to the command to suppress a carriage-return/line-feed.

- F

Causes the input pointer for the currently active input buffer to be spaced forward over the next parameter. If the input buffer pointer is currently at the end of the buffer, this command has no effect.

- GO n
  G n

Provides the facility for transferring control within the PROC to a different PROC line. The transfer is to the statement referenced by the label.

- H

Causes the body of text immediately following the "H" to be placed in either the primary or the secondary output buffer, depending upon whether the stack is currently on or off. When the last parameter has been moved to the secondary output buffer, an end-of-line must be placed in the stack. The form is "H ...<" This pertains only to the stack, not the primary output buffer.

- IF

The "IF" command provides a facility for validating the parameters in the input buffers prior to moving them to the output buffers with the A command. There are three basic forms of the "IF" command. In each form, when the test criteria is satisfied, the associated PROC statement is executed. All forms of the "A" command are legal in the "IF" command except the form using the character surround ("Ac"). The "IF" command does not move or disturb data in the input buffer; however, it does move the pointer just like the "S" command.

Form 1: IF A proc-statement
IF #A proc-statement

The first form of the "IF" command simply tests for existence of data in the input buffer.

EXAMPLE--

```
IF A GO 100
```

In the example, if there is data in the input buffer, control transfers to the PROC statement labeled 100; otherwise the next line after the "IF" will be interpreted.

Form 2: IF A op string proc-statement

The second form of the "IF" command compares the contents of the input buffer with a literal "string" according to one of the following relational operators:

Relational operators (op) in the "IF" command

IF A = ...	Test for equal values
IF A #	Test for unequal values
IF A <	Test if parameter is less than value
IF A >	Test if parameter is greater than value
IF A [....	Test if parameter is less than or equal to value
IF A]	Test if parameter is greater than or equal to value

In the case of <, [, >,], the test is on a left-to-right character basis only; thus leading zeroes in numeric fields may not compare correctly (003 will test as less than 2, for instance).

EXAMPLE--

```
IF A = YES GO 15
```

In the example, should the data in the input buffer match the string "YES", the PROC will continue executing at statement 15.

Form 3: IF A op (format) proc-statement

The third form of the "IF" command allows a pattern match for numeric alpha, alpha-numeric characters and literals. The "format" is any combination of:

```
nN      Test for "n" numeric characters.
nA      Test for "n" alpha characters.
nX      Test for "n" alpha-numeric characters.
```

Any other characters in the "format" test for the literal character specified. If n=0, then data will be skipped until a non-matching character type is encountered.

EXAMPLE--

```
IF A = (2N/2N/2N)  ODATE OK
IF A = (3N-2N-4N)  F
IF A - (PT7N3X)    GO 10
```

The second example "IF" could be used to validate a social security number. The third could be used to test for input such as: PT5550301AX9. The following command will test the input parameter to see if it is in the range 10 through 19, and go to label 99 if it is:

```
.
.
.
IF A = (2N) IF A ] 10 IF A [19 GO 99

Error condition here
```

- IH

Causes the line of text (including blanks) immediately following the "IH" to replace the current parameter in the currently active input buffer.

- IN
  INc

The "IN" command causes input to prompted at the terminal. Unless specified, the TCL prompt character (colon :) will be displayed on the terminal. The second form prompts with a user defined prompt character, which will remain in effect till changed.

EXAMPLE--

```
          Prompt

IN?       ?
IN=       =
IN        =      (if specified after the preceding IN=)
```

Data input from the "IN" statement is placed into the secondary input buffer. Subsequently, this data may be read from the secondary input buffer by using the "A" statement. However, when the primary input buffer is specified through the "A" statement or if the "S(m)" statement is used, the data in the secondary input buffer is lost. Also, any time the "IN" statement is executed, input from the terminal will overwrite any previous data in the secondary input buffer.

- IP

IP may be used like IN to accept input from the terminal. Data input with the IP command replaces the current parameter of the current input buffer (a parameter is a string of characters up to a blank, or a string of characters enclosed in quotes). If the input buffer pointer is at the end of the current buffer, the data is appended to the current buffer. IP does not move the input buffer pointer. Note that several parameters will replace one parameter. The normal A,Ap,A(m,n),Sp,S(m)F, and B may be used to position and copy the arguments to the appropriate output buffers. It should be noted the arguments remain in the primary input buffer after the P command has been executed for "re-use". Input buffer overflow is defined as the sum of the bytes in the current primary and secondary input buffers exceeding 280 bytes. If this condition occurs, IP will return error 269: INPUT BUFFER OVERFLOW AT PROC STATEMENT: statement.

Additional forms:

IPc where c is the prompt character

IPB where embedded blanks will be replaced with \

IPBc

- IT
  ITA
  ITCA

Input from magnetic tape. One record (max. size 140 characters) is read from the magnetic tape unit, and the data is stored in the primary input buffer. (The original primary input buffer is lost.) If the letter "C" follows the "T" in this specification, an EBCDIC to ASCII conversion is performed on the tape record. The letter "A" masks the 8 bit ASCII character to 7 bits (MSB=0).

A null record constitutes an end-of-file condition; an all-blank record will also give this same condition.

- O

Causes the entire body of the text following the "O" to be output to the terminal. If the last character of this message is a plus sign (+), the carriage will not be returned at the end of the message.

EXAMPLE--

```
OPART-NUMBER =+
IN?
```

- P

The "P" command allows the PROC to proceed via TCL. A single letter "P" submits the primary output buffer to TCL for processing. Optionally, the P may be followed by a mode-ID and in this case, control is transferred to the specified modal entry point. After execution in TCL or the user's defined program or processor, the PROC regains control at the statement immediately following the P command.

- PP

Same as in the "P" command, except that the data in both the PROC output buffers are printed on the terminal, and terminal input requested using a question mark (?) as a prompt character. If the input is anything other than an "N", the generated statement is passed to TCL; if an "N" is entered, PROC execution is aborted, and an exit taken to TCL.

- RI
  RI(m)
  RIp

"RI" resets both input buffers to an empty or null condition. "RI(m)" clears data from the primary input buffer from column "m" to the end of the buffer. "RIp" clears data from the primary input buffer from the pth parameter to the end of the buffer.

EXAMPLE--

If the input buffer was:

```
(start) 11020 D 'JOHN SMITH' XX(end)
```

and the command "RI3" is executed, the result and buffer is:

```
(start) 11020 D (end)
```

The buffer pointer is always at the end of the truncated buffer.

- S(m)
  Sp

Used to both set the position of the input pointer and/or to select the primary input buffer as the current input buffer. The "m" parameter is used to determine at what column to set the input pointer. S(m) moves the pointer to the mth column of the primary input buffer. Sp moves the pointer to the pth field of the currently active input buffer, where the fields are separated by blanks. Subsequent references via the parameter insertion code "A" will extract parameters from the current position of the input buffer set by the "S" command.

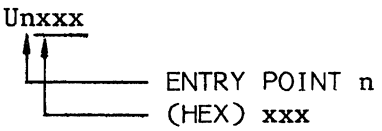
- ST ON, STON
  ST OFF, STOFF

Used to turn the stack on or off. The stack is the secondary output buffer. When the stack is on, all data picked up by the "A" command moves to the secondary output buffer. When the stack is off, these data move to the primary output buffer. The stack may be turned on and off at any point in time during the processing of a PROC. As it is turned on and off, output is then routed to either the primary or the secondary output buffer.

- U

Used to provide an exit to a user defined subroutine. The format for this statement is identical to the second format of the P command; however, the U command is meant to be used for a simple subroutine call. TCL is not involved and assuming the user does not modify his buffers they will remain unchanged during this exit. Upon return from the subroutine, control is passed to the statement immediately following the U command. (Refer to program documentation, "User exits from PROC", Section XX).

EXAMPLE--



- X

The "X" command is used to exit from the PROC. Normally, a PROC control is terminated with execution of the final PROC statement. The X command may be used at intermediate points in the coding to cause termination of the PROC. Any characters following the X will be output as a message, prior to termination of the PROC.

EXAMPLE--

```
X-EXIT TO TCL-
IF A = END X-HURRY BACK-
```

- +n
  -n

The decimal number "n" is added to or subtracted from the parameter at the current input buffer position. If the input buffer pointer is currently at the end of the buffer, this command has no effect.

The updated parameter is written back in place; therefore, care must be taken to ensure that the updated value is the same length as the previous parameter, since no check for this is made. Also, it is assumed that the parameter is numeric.

EXAMPLE--

(loop control within a PROC)

```
IF A1 #(2N) XERROR
100 C START LOOP HERE
.
.
body of loop
.
.
S1
-1
C TEST END OF LOOP
IF A # 00 GO 100
```

(note test for "00", not "0")

Section VI

LOGON/LOGOFF

INTRODUCTION

The LOGON processor provides a facility for initiating a user's session by identifying valid users and their associated passwords. The LOGOFF processor is used to terminate the session and should always be evoked via the verb OFF when the user wishes to terminate. These processors accumulate accounting statistics for billing purposes and also associate the user with his privileges and security codes.

Logging On to the System

The user may log on to the Reality system when the message:

```
LOGON PLEASE:
```

is received. The user then enters the name, or identification, established for him in the system. If a password has also been established, he may follow his identification with a comma, and then his password. If the password is not entered as a response to the LOGON PLEASE: message, the system will then output the message:

```
PASSWORD:
8888
```

The blacked-out area allows the user to enter his password and not have it readily observable (on a printing terminal, for example). Reality validates the user's identification against the entries in the Master Master Dictionary; if the identification is illegal, the message:

```
USER-ID?
```

is returned, and the LOGON PLEASE: message repeated. If the user identification is valid, but the password is not acceptable, the message:

```
PASSWORD?
```

is returned, and the LOGON PLEASE: message repeated. Assuming that both the user identification and the password are accepted, the user has successfully logged on to the system. The message:

```
*** WELCOME TO MICRODATA REALITY ***

*** Time      RELEASE 1.X     Date ***

:
```

will be returned; the user is then prompted for input with a colon (:), the TCL prompt character.

The terminal characteristics (see System Commands) are set to an initial condition by LOGON; these initial values are:

```
Page width        :   110 characters
Page body         :    44 lines
Page skip         :     7 lines
Line-feed delay   :     0
Form-feed delay   :     0
Backspace echo    :    X'08' (ASCII backspace)
```

These correspond to an 8-1/2" x 11" page size.

The Logon PROC

When the user has logged on to his account, Reality permits the automatic execution of a PROC whose item-id is the same as the user identification. That is, the Master Dictionary of the account will be searched for a PROC matching the user identification which was used to log on to the account, and, if it is found, it will be executed. Typically, this is used to perform standard functions that are always associated with the particular user's needs, resetting terminal characteristics as an instance.

EXAMPLE--

Assume that a PROC called REF-MAN has been established on the M/DICT of Ref-man's account; a dump of the PROC is as below:

```
:COPY M/DICT REF-MAN (T) (r)

   REF-MAN
001 PQ
002 HTERM 118,44,7,6
003 P
004 X** TERMINAL CHARACTERISTICS ARE SET. **
```

The Logon sequence is shown below:

```
LOGON PLEASE: REF-MAN (r)                    (user name entered)
PASSWORD:
XX X (r)                                     (password obscured)

*** WELCOME TO MICRODATA REALITY ***
*** 15:30 RELEASE 1.0 24 OCT 1973 ***

** TERMINAL CHARACTERISTICS ARE SET. **      (message from PROC
                                             REF-MAN)
:                                            (TCL prompt character)

                                             (User is now logged on
                                             to the system)
```

General System Message

There is the facility to send the same message to each user as he logs on to the system. The item-id 'LOGON' in the ERRMSG file, may define such a message, which is typically used to transmit information pertaining to system up-time, and the like. It should be noted that the LOGON message item must be present in the ERRMSG file even if no general system message is to be sent; in this case, the item should have no attribute values (i.e., an item-id only). The format of the LOGON message item is the same as any other message item in the ERRMSG file; see System Maintenance.

Logging Off the System

Logoff is achieved by entering the word OFF, either at the TCL level, or at the DEBUG level. A message indicating the connect time (number of minutes that the user was logged on), and the charge-units used, will be returned, before the system re-enters the LOGON state and waits for the next user session to be initiated.

The charge-units represent usage of the CPU. It is normally in tenths of a CPU second, though this can be changed since it is a monitor software convention.

EXAMPLE--

```
:OFF (r)

*** CONNECT TIME = 16 MINS.; CHARGE-UNITS = 295 ***
*** LOGGED OFF AT  15:33  ON  24 OCT 1973.      ***

LOGON PLEASE:
```

Clearing the ACCOUNT File

To avoid overflowing the entry in the ACCOUNT file for a specific account-name, the file should be cleared periodically by the following procedure:

```
        LOGON to SYSPROG
Type:   LIST ACC 'account-name' LPTR
        DELETE ACC account-name
```

The point of overflow is determined by the activity of the account; however, approximately 200 LOGON/LOGOFFs are allowed. This point can be calculated by the following procedure:

Type: CT SYSTEM account-name

Attribute 8 gives the System Level Privileges and the amount of work space; if no number is specified, 6 is assumed. Multiply this number by 500 and subtract 100 for the approximately maximum size.

To determine the actual size, type:

```
STAT ACC ACC-SIZE 'account-name'
```

This will produce a message in the following format:

```
STATISTICS OF ACC-SIZE :
TOTAL = 52 AVERAGE = 52.ØØ COUNT = 1
```

If the value for TOTAL approaches the size calculated above, the account is approaching the overflow point.

If the ACCOUNT file does overflow, the response to OFF will not be the standard message described in the previous section, but will be in the following format:

```
FORW LNK ZERO; REG = 0.E
ABORT @ 50.16D
```

Subsequent OFFs will produce the same result. To recover from this situation, follow this procedure:

```
        LOGON to SYSPROG on another terminal
Type:   LIST ACC 'account-name' LPTR (to save the ACCOUNT history)
        DELETE ACC account-name
```

User Identification Items

All User Identification Items are stored in the System Dictionary (SYSTEM), and are either file definition items or file synonym definition items.

The items in the System Dictionary therefore define the set of users who can log on to the system. Attributes two through four of these items are identical to the corresponding attributes in file definition or file synonym definition items; attributes five through eight contain data associated with the user's lock (security) codes, password, and system privileges as under:

Attribute 5: Contains the set of retrieval lock-codes associated with the user. Multiple values (separated by value marks) are allowable; there is no restriction as to the format of individual lock-codes. This attribute may be null, indicating no lock-codes. Lock-code usage is described under Security.

Attribute 6: Contains the set of update lock-codes associated with the user. As for retrieval lock-codes.

Attribute 7: Contains the user's password, which is a single value; this attribute may be null; there is no restriction as to the format of the password.

Attribute 8: Contains a code which indicates the level of "system privileges" (see below) assigned to the user. Also contains a code which indicates additional work-space requirements for this user (see below).

Attributes nine through thirteen are as defined for file definition or file synonym definition items.

System Privileges

Three levels of system privileges are available; they are referred to as zero (lowest), one, and two (highest) respectively. Lower levels of system privileges restrict usage of certain facilities of the system:

Facility	Lowest Privilege Level Required
Updating of M/DICT	One
Use of magnetic tape	One
Use of DEBUG (other than P, OFF, END and G commands)	Two
Dump Processor	Two

Facility	Lowest Privilege Level Required
Use of ASSEMBLER & LOADER	Two
Use of verbs : SET-TIME; SET-DATE; :INIT-LINES; File save and restore processors	Two

System privileges are assigned by the code in attribute eight of the user identification item in the System Dictionary; the following codes are used:

SYS2 : assigns level two privileges.

SYS1 : assigns level one privileges.

Anything else : assigns level zero privileges.

These codes may be immediately followed by the additional work-space assignment parameter, described below.

Additional Work-Space Assignment

As noted under Data Structures, there are three "linked" work-spaces, symbolically named the HS, the IS and the OS, that are set to an initial size of six frames (3000 bytes) each at logon time. It is possible that particular users require additional work-space (principally for assembling source programs); this requirement may be specified in attribute eight of the user identification item, immediately following the system privileges code.

The format of this parameter is:

(n) where "n" is a decimal numeric; $6 < n < 128$

"n" specifies the work-space requirement, in number of frames, for each of the three linked work-spaces HS, IS and OS. "n" must be greater than six, since six frames are initially made available by the system.

The additional space is obtained from the pool of contiguous overflow space; if such space is not available, the message:

```
[334] REQUESTED NUMBER OF ADDITIONAL WORK-SPACE FRAMES:  XXXX
      IS NOT AVAILABLE.
      ADDITIONAL WORK SPACE HAS NOT BEEN ASSIGNED.
```

will be returned after the "WELCOME etc" message (XXXX represents the actual attribute value from attribute eight of the user identification item).

If the format of the work-space parameter is illegal, ("n" out of range, missing right parenthesis, non-numeric "n" field), the message:

```
[333] THE FORMAT OF THE ADDITIONAL WORK-SPACE PARAMETER:  XXXX
      IS ILLEGAL.
      ADDITIONAL WORK-SPACE HAS NOT BEEN ASSIGNED.
```

will be returned after the "WELCOME etc" message.

EXAMPLE on the next page.

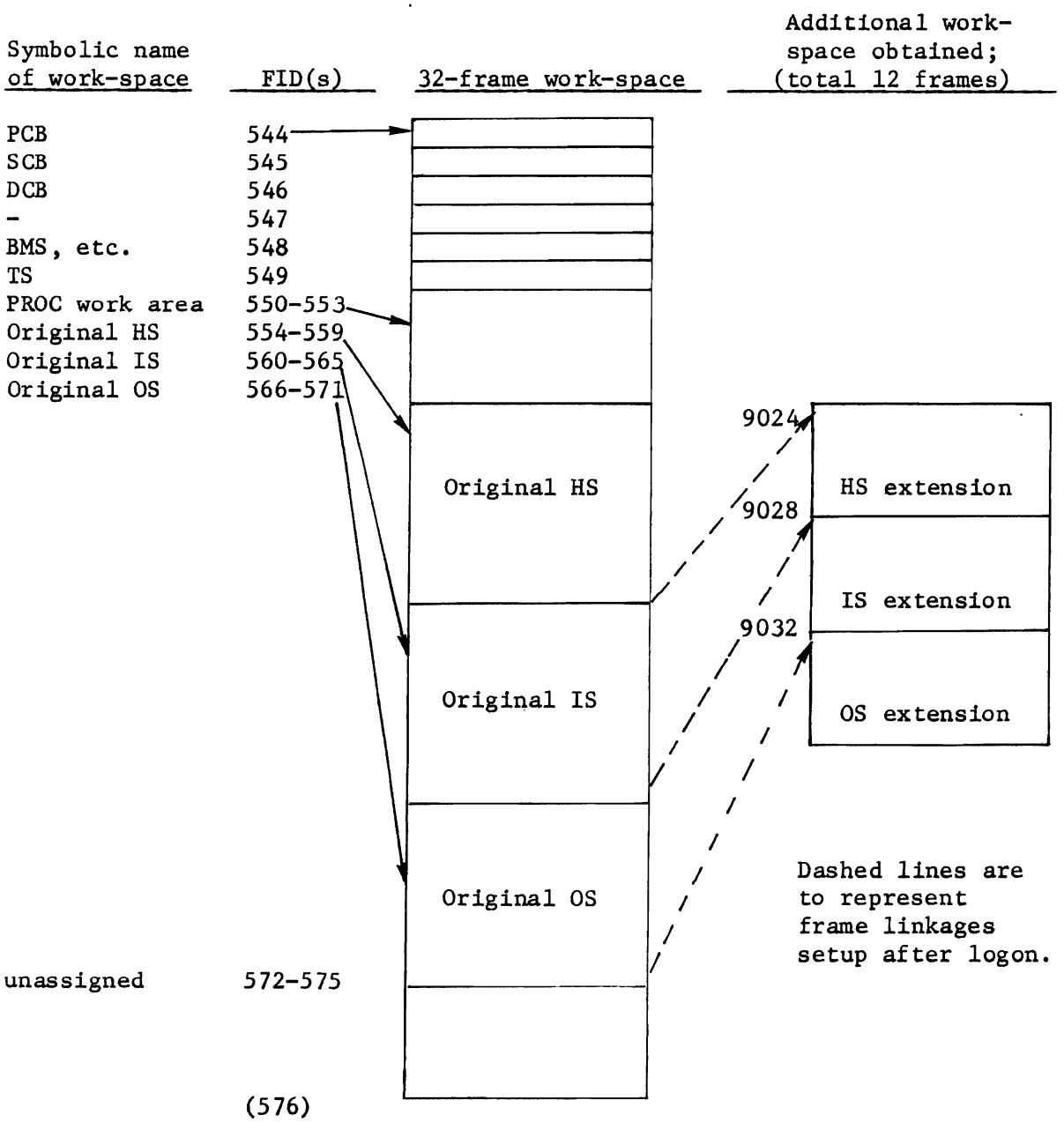
Updating System Dictionary Entries

Entries in the System Dictionary should only be updated, from the SYSPROG account, when no other users are logged on to the system. This is because the system software maintains pointers to data in the System Dictionary when users log on, and updating the System Dictionary will invalidate the pointers. An exception to this rule is when creating a new account, or a synonym to an existing account, which can be done at any time, since new items are added to the end of the existing System Dictionary data, and thus do not disturb any pointers to it.

EXAMPLE--

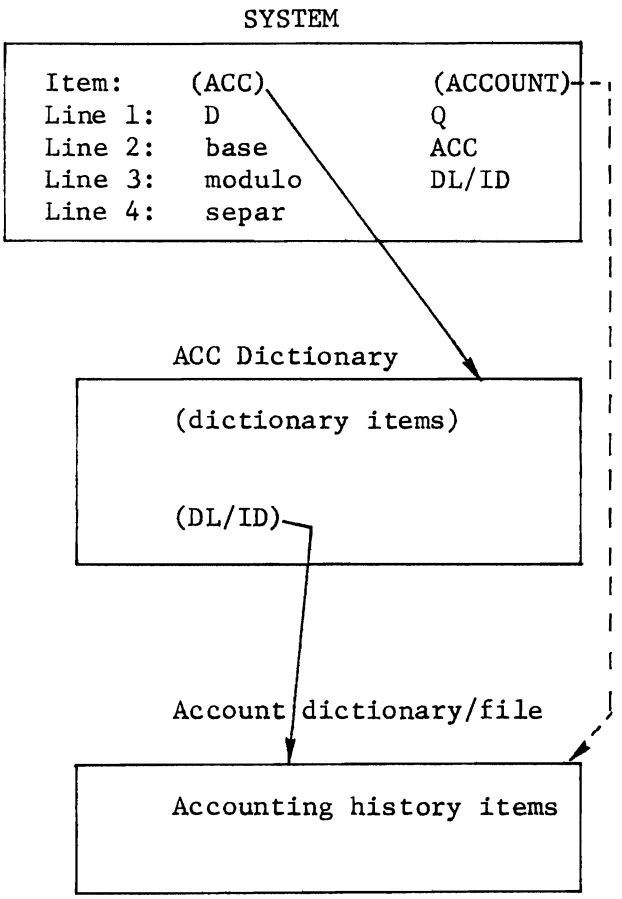
The entry in the eighth attribute is : SYS1(10)

This gives the user system privileges, level one, and ten frames of work-space per linked set. Thus 12 additional frames have to be obtained from contiguous overflow space. It is assumed that the user logs in on process one (communications channel one on device address X'18'), which corresponds to a PCB FID of X'220', 544_{10}.



The Accounting History File

This is one of the mandatory files in the Reality system; the file definition item, or file synonym definition item, 'ACCOUNT', must be present in the SYSTEM. It is treated as a dictionary-level file by the LOGON/LOGOFF processor; that is, it updates directly into the file defined by the ACCOUNT entry, not going one level further down as for a data file.



The actual file linkages set up for the ACCOUNT file and ACC dictionary are as shown on the left. The ACC dictionary is set up for convenience in examining and listing data in the Account file.

There are two types of entries in the ACCOUNT file--those that represent active (logged-on) users, and those that keep track of accounting history.

Active Users Entry

The item-id of an active user entry is a four character, hexadecimal FID of the PCB of the user's process. Recalling that the PCB's start at FID = 512, and proceed in steps of 32 frames from there on, a user logged on to process zero will have an entry with an item-id '0200' (512_{10}); on to process one with an item-id '0220' (544_{10}), etc. Attribute one contains the name of the user (user identification item-id); attribute three the date logged on; and four the time. Active user entries are created when a user logs on, and deleted when he logs off.

Accounting History Entry

The item-id of an accounting history entry is the name of the user; attributes one, two, and three are not used. The remainder of the attributes are described below:

Attribute 4: Date(s) logged on - each unique date is stored; value marks are tagged on to the value in this attribute if multiple logoffs on the same date, for LIST alignment purposes. Date is stored in Reality date format.

Attribute 5: Time(s) logged on- an entry is made on each log-off, representing the time at which the user had logged on. Time in seconds past midnight, 24-hour clock.

Attribute 6: Connect time(s) - an entry representing the time in seconds between the logon and the logoff.

Attribute 7: Charge-units - a number representing the CPU usage is added on each logoff.

The accounting history entry is updated every time a user logs off the system; thus the entry stores the history of every session (logon to logoff).

ACCOUNT File Attribute Description

Attribute Number	ACC dictionary Name	Active User Entry	Accounting History Entry
0	(item-id)	Four-character hexadecimal PCB-FID	User name
1	A1	User name.	Not used
2	A2	Date logged on	Not used
3	A3 or DISK-SPACE	Time logged on	Not used
4	DATE		Dates logged on
5	TIME		Times logged on
6	CONN		Connect time
7	UNITS		Charge-units
8*	CHARGES		Computed charges
9*	TOT-CONN		Computed charges connect time
10*	TOT-UN		Computed total charge-units
11*	TOT-CHRGS		Computed total charges

*Functional relationships; no data stored in these attributes. Data values shown for these attributes in the examples are for illustration only.

EXAMPLES--

The statement below is a sorted listing of active users (users "WITH A1").

```
:SORT ACC WITH A1 A1 A2 A3 HDR-SUPP (r)

ACC..... A1..... A2..... A3....

0240        PICK       24 OCT 1973 15:16
0280        MVC        24 OCT 1973 11:22
02A0        REF-MAN    24 OCT 1973 15:01
02C0        ROCS       24 OCT 1973 14:55
02E0        E          24 OCT 1973 14:19
```

User name
Time logged on
Date logged on
PCB-FID = X'0240';
Channel two (578_{10})

The example below is a listing of the user REF-MAN's accounting history item:

```
:LIST ACC 'REF-MAN' HDR-SUPP (r)

ACC..... DATE..... TIME. CONN. UNITS CHARGES TOT-CONN TOT-UN TOT-CHRGS

REF-MAN      13 OCT 1973 19:11  0:21  2335    5.80     1:22    4111     12.73
                         20:00  0:54   987    4.89
             15 OCT 1973 10:55  0:02   452    1.05
             24 OCT 1973 11:51  0:03   308    0.81
                         12:49  0:02    29    0.18
```

Below is the same item dumped so as to show its internal storage format:

```
:I-DUMP ACC    'REF-MAN' (r)

REF-MAN^^^^2113]]2115]2124]^69079]72000]39330]42693]46184^1266]3245]
169]224]141^2335]987]452]308]29^
```

Dates
Times logged on
Connect times
Charge-units
Attribute mark
Value mark

Section VII

FILE MANAGEMENT PROCESSORS

•CREATE-FILE

The CREATE-FILE Processor provides the capability for generating files and dictionaries in the Reality system. This processor is used to create file dictionaries which exist as "D" entries in a users Master Dictionary (M/DICT). The CREATE-FILE processor is also used to reserve disk space for the data portion of the new file and automatically puts a DL/ID (Data-List/Identifier) entry in the file dictionary, which points to the data. CREATE-FILE will automatically locate and reserve a contiguous block of disk frames from the available space pool. The user therefore needs only to specify the modulo and the separation of both the file dictionary and the data. Refer to the Data Structures section for the definition of modulo and separation.

- Modulo - Must be a non-zero integer

  $o < m$

- Separation - Must be a non-zero integer less than 128.

  $o < s < 128$

For users with no system level privileges or with Level One system level privileges the following restriction applies:

- The product of modulo times separation may not exceed 512

  $m * s \leq 512$

Note: A user with SYS2 privileges is limited only by his disc space.

Refer to notes in the Data Structures section regarding the selection of values for modulo and separation.

The dictionary (DICT) of the file must always be created first and the name given to this new file must not exist in the users M/DICT. The general form for creating a file dictionary is:

```
CREATE-FILE (DICT   file-name   m,s)
```

EXAMPLE--

```
:CREATE-FILE (DICT SAMPLE 1,1) (r)

(417) FILE 'SAMPLE' CREATED; BASE = 14946, MODULO = 1. SEPAR = 1.
```

Once the DICT has been created the disk space for the data area of the file can be reserved. The general form is:

```
CREATE-FILE (DATA   file-name   m,s)
```

EXAMPLE--

```
:CREATE-FILE (DATA SAMPLE 11,3) ⓡ

(417) FILE 'DL/ID' CREATED; BASE = 14947, MODULO = 11, SEPAR = 3.
```

An alternate form of the CREATE-FILE processor is shown below. This enables the creation of both the DICT and the DATA areas with one statement. The general form is:

CREATE-FILE (file name m_1,s_1,m_2,s_2)

where m_1 = DICT MODULO

s_1 = DICT SEPARATION

m_2 = DATA MODULO

s_2 = DATA SEPARATION

EXAMPLE--

```
:CREATE-FILE (SAMPLE  1,1  11,3) ⓡ

(417) FILE 'SAMPLE' CREATED; BASE = 14980, MODULO =  1, SEPAR = 1.
(417) FILE 'DL/ID' CREATED; BASE = 15132, MODULO =  11, SEPAR = 3.
```

CLEAR-FILE

The CLEAR-FILE Processor clears the data from a file; that is, it sets the file "empty" by placing an attribute mark in the first data position of each group of the file. "Overflow" frames that may be linked to the primary frame space of the file will be released to the System's overflow space pool.

Either the DATA section or the DICT section of a file may be cleared using the CLEAR-FILE command. If the DICT section is cleared, and a DATA section exists (as implied by the presence of the DL/ID item in the dictionary), the DL/ID will be maintained in the dictionary.

To clear the DATA section of a file, the following command is used:

```
CLEAR-FILE (DATA file-name)
```

To clear the DICT section of a file, the following command is used:

```
CLEAR-FILE (DICT file-name)
```

Files that are defined by file-synoym definition items in the user's M/DICT cannot be specified in a CLEAR-FILE command.

DELETE-FILE

The DELETE-FILE Processor allows the deletion of a file, either the DATA section, or both the DATA and the DICT section. The DICT section of a file which has a DATA section cannot be deleted. The formats of the DELETE-FILE commands are exactly as those for the CREATE-FILE commands.

To delete the DATA section of a file, the following command is used:

```
DELETE-FILE (DATA   file-name)
```

which will delete the DATA section, and also delete the DL/ID entry from the DICT section.

To delete the DICT section of a file, the following command is used:

```
DELETE-FILE   (DICT file-name)
```

To delete both the DATA and DICT sections, use DELETE-FILE (file-name).

In both the above cases, the file-definition item in the M/DICT (the file-name) is also deleted.

Files that are defined by file-synonym definitions in the user's M/DICT cannot be specified in a DELETE-FILE statement.

COPY

The COPY processor allows the user to copy items from a file to the terminal, line printer, or to another file, either in his account, or in some other users account. However, in order to COPY to another users account, a file synonym definition ("Q" entry in D/CODE) which points to that specific account and file, must already exist. (Refer to the section on Dictionaries.)

The COPY processor is a TCL-II verb and consequently has the same general form as a TCL-II verb. (Refer to the section on TCL-II Processing.) The general form of the COPY command is:

```
                                item-id's
COPY   [DICT]   file-name         or        (options)
                                   *
```

The format for copying existing items in a dictionary to new dictionary items within the same file:

```
COPY   DICT   file-name   dict-item-name(s)
```

EXAMPLE--

```
:COPY DICT SAMPLE COST (r)
TO  :WORTH (r)

1 ITEMS COPIED
```

In this example the COPY processor creates a new attribute called "WORTH" for the file "SAMPLE", by copying the existing attribute "COST" from the file named "SAMPLE".

The format for copying data from one item to another item within the same file is:

```
COPY   file-name   item-id
```

EXAMPLE--

```
:COPY SAMPLE 1242-01 (r)
TO  :1242-99 (r)

1 ITEMS COPIED
```

In using the COPY verb, multiple items may be specified as the source and as the destination. They must be separated by blanks.

A one-to-one correspondence between source-file and destination-file item lists is not mandatory. If the source-file list is exhausted first the COPY terminates. If the destination-file list is exhausted first, the remainder of the items are copied with no change in item-id.

EXAMPLE--

```
:COPY FLAVORS RED WHITE BLUE (r)
TO:  :ALPHA BETA GAMMA (r)

3 ITEMS COPIED
```

This example copies data items RED, WHITE and BLUE back into the file called FLAVORS but gives them item-ids of "ALPHA" "BETA" "GAMMA".

If it is desired to copy all existing items, this is done by using an asterisk (*) following the file name.

EXAMPLE--

```
:COPY DICT SAMPLE * (r)
TO :(DICT FLAVORS) (r)

2 ITEMS COPIED
DL/ID NOT COPIED
```

This will copy all the attribute definition items from the file name "SAMPLE" to the file name "FLAVORS". Note that when a destination of other than the source file is desired, the destination file must be enclosed in parentheses. If the parentheses are not used, the items will be copied into the original file.

When copying from one file to another the COPY processor does not bring dictionary entries which have a D/CODE of "D", such as the DL/ID entry. If this restriction is not in effect, a "FILE-SAVE" operation will save the data in the file more than once and a subsequent file restore will cause multiple copies of the data to exist on the disk. DL/ID entries must be only created by the CREATE-FILE processor.

To recreate both the dictionary and the data of one file in a new file, the following sequence of COPY commands may be used:

```
:CREATE-FILE (NEW-SAMPLE 1,1 3,1) (r)

[417] FILE 'NEW-SAMPLE' CREATED; BASE = 15417, MODULO = 1, SEPAR = 1.

[417] FILE 'DL/ID' CREATED; BASE = 15418, MODULO = 3, SEPAR = 1.
```

A new file called NEW-SAMPLE has now been created.

```
:COPY DICT SAMPLE * (r)
TO :(DICT NEW-SAMPLE) (r)

3 ITEMS COPIED
```

This copies all the dictionary items from SAMPLE to NEW-SAMPLE except the DL/ID

```
:COPY SAMPLE * (r)
TO :(NEW-SAMPLE) (r)

2 ITEMS COPIED
```

This copies all the data items from SAMPLE to NEW-SAMPLE.

Copying to the Magnetic Tape, Line Printer or Terminal

Items can be copied to the tape if, when the TO: message appears, the response is "(TAPE)". Note that file definition items (D/CODE = "D") will not be copied. The tape so created can be read using the T-LOAD verb.

EXAMPLE--

```
:COPY M/DICT * (r)
TO:  (TAPE) (r)

157 ITEMS COPIED.
```

Items can also be copied to the terminal or to the line-printer either in character format, or as a hexadecimal dump; see options below.

Options

Several options are available for use with the COPY processor. The desired options are specified by one or more single characters which are separated by commas. The entire options list must be enclosed in parentheses, and must follow the item-id list (or *).

Options are described below:

Option	Description
D	Deletes items from the source-file after they are copied to the destination file.
O	Overwrites destination file items with source file items.
P	Copies items to the line-printer.
T	Copies items to the user's terminal.
X	Specifies a hexadecimal dump (to the terminal or line printer), when used with P or T option.
L	Suppresses line-numbers (on a copy to the terminal or line printer), when used with the P or T option.

EXAMPLE--

```
:COPY FLAVORS ALPHA BETA (D,O) (r)
TO :BLUE YELLOW (r)

2 ITEMS COPIED
```

The above example copies two items from the file named FLAVORS back into FLAVORS changing the item-ids from ALPHA to BLUE and from BETA to YELLOW.

The items ALPHA and BETA are deleted. If either item previously existed (BLUE or YELLOW), it is overwritten.

Note: Not all combinations of option characters are meaningful. D and O options only apply on a file-to-file copy. X and L options apply only on a copy to the terminal or to the line printer. None of the options apply on a copy to the tape.

Note: If a null line (carriage return) is supplied to the destination prompt "TO:" the copy will be sent to the terminal.

EXAMPLE--

```
:COPY M/DICT LISTACC (r)
TO : (r)

     LISTACC
001  PQ
002  HLIST ACC WITH NO A1
003  F
004  10 IF A GO 11
005  P
006  X
007  11 IF A = LPTR GO 12
008  IF A = PAGE GO 12
009  A'
010  GO 10
011  12 A
012  P
```

Copies the PROC "LISTACC" from the Master Dictionary to the terminal.

File Management Verbs

ITEM filename item-id [(options)]

Displays the base FID of the group into which the item-id hashes. If the item is not already on file, the message: ITEM NOT FOUND is printed on your screen. In addition every item-id in that group is listed along with a character count of the item in hex. At the end of the list the following message is printed: n ITEMS m BYTES p/q FRAMES

where:

n is the number of items in the group

m is the total number of bytes used in the group

p is the number of full frames in the group

q is the number of bytes used in the last frame of the group.

Valid options are

P direct output to lineprinter.

S suppress item list.

GROUP filename [(options)]

Same as ITEM except each group in the file is used and no specific item can be tested.

POVF [x]

POVF displays the number of frames in each contiguous block. There are two ways to use POVF. If POVF x (r) is keyed in where x is any character, the linked chain is not counted. In this case the first line of output will contain only the FID of the first frame in the chain. If POVF (r) is keyed in, the linked chain is counted and the number of frames in the chain is displayed in parenthesis on the first line of output. Regardless of which way POVF is used, every line of output after the first is of the format m n (p) where m is the first frame of a contiguous block, n is the last frame of that block and p is the number of frames in the block.

SEL-RESTORE Verb

The SEL-RESTORE verb may be used to selectively restore a file, or specific item within a file, from a file-save tape. Only one file at a time can be restored. SEL-RESTORE may be run from any account, not necessarily from the account whose file is to be restored, as long as the file is defined in the M/DICT of the account from which the restoration is done.

The general form of the SEL-RESTORE statement is as follows:

SEL-RESTORE file-name account-name 'item-id-list'

where

file-name is the name of the file to be restored; it must match the name on the file-save tape. It may be preceded by "DICT" specifying dictionary.
"DICT" must be used when restoring an M/DICT.

account-name is the name of the account from which the file was defined (as a D-CODE item) when the file-save tape was created.

item-id-list is optional; if omitted, all items are restored; otherwise, specific item-ids may be entered, each enclosed in single quotes.

Each item-id restored will be listed, and a message indicating the total number of items restored will be returned.

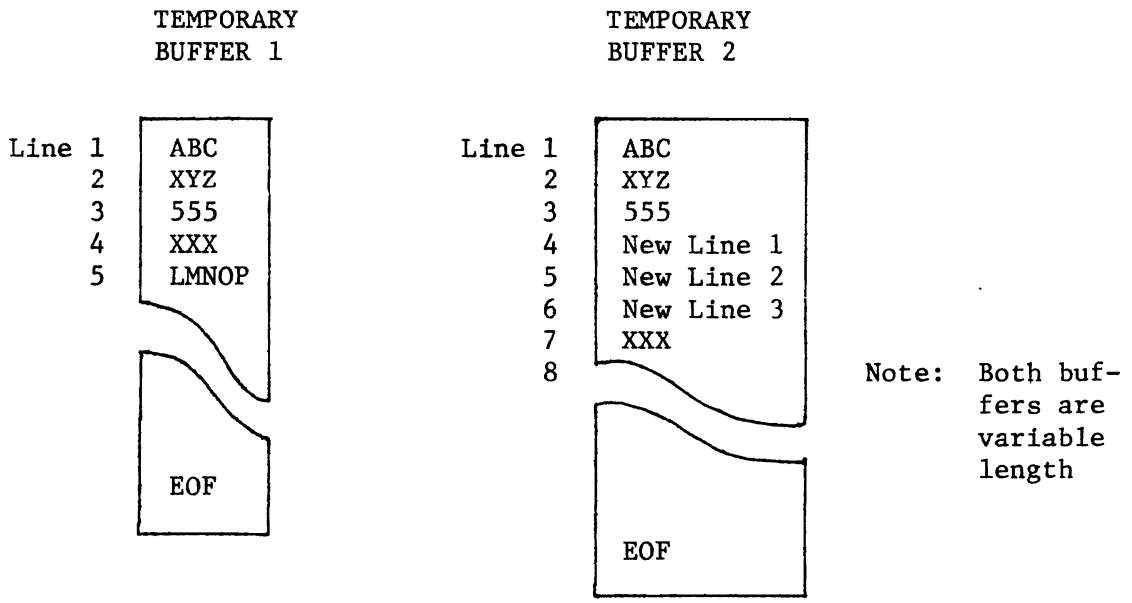
Note: SEL-RESTORE will exit to LOGON if end of tape is reached due to (1) account-name not found, or (2) the very last file on the tape is restored.

Section VIII

EDITOR

INTRODUCTION

The EDITOR is a processor which permits on-line interactive modification of data and file dictionaries of the Reality Computer System. This means that PROC's and assembly source can also be modified by the EDITOR. The EDITOR allows line-at-a-time modification of any item in the data base. When the EDITOR is entered the item being edited is copied to one of two temporary buffers. The item is permanently updated on the disk file only after a file command.



The EDITOR uses the current line concept; that is, at any time, there is a current line that can be listed, altered, deleted, etc. As one moves through the item from TOP to EOF (end-of-file), the current line pointer changes accordingly. If the current line pointer is at EOF, an "L" command will place the pointer at the top before processing. Initially, the item is in Buffer 1. As an edit is performed the modified line and all previous lines are copied to Buffer 2. The editing process continues working on Buffer 1. As lines in the item are changed, or new lines inserted the EDITOR builds a new, updated item in Buffer 2. Updating must be done in ascending line-number sequence, until the F command is used to merge updated with the previously existing item; and automatic resequencing of the item then takes place. Updated lines cannot be listed until such a merge has taken place. Functionally, the EDITOR stores the updates, and only applies them to the item when the F command is entered. An F command does not permanently file an item. It completes the copy to the alternate buffer causing all lines to be resequenced and the EOF marker to be repositioned. It then switches

the function of the buffers. Editing occurs in Buffer 2 with the new modification assembled in Buffer 1. This toggling of buffers can go on indefinitely until the item is permanently filed away with an "FI" or "FS" command.

Entering The Editor

1. EDIT file-name item-id [(options)] ⓡ

2. EDIT DICT file-name item-id [(options)] ⓡ

Example 1. Edits the data referenced by item-id, in the reference file

Example 2. Edits the item in the dictionary of the referenced file

Multiple items may be specified and are delimited by blanks or all items may be specified by the use of an asterisk (*). In the latter case, each item-id lists on the terminal as the EDITOR is entered.

If the item exists on the file, the message:

TOP

will be returned; the current line pointer is set to zero, the INPUT environment is entered, and an EDIT command is requested with the EDIT prompt character (.).

If the item does not exist on the file, the message:

NEW ITEM

will precede the TOP message; the EDIT environment is entered as before.

Valid options are:

Z Suppress the TOP message the first time. This feature is useful when EDIT is called at the PROC level.

Edit Command Structure

EDIT commands are one or two letter mnemonics which must appear as the first non-blank input character. Command parameters, or operands, follow the command; blanks may be inserted between parameters for clarity if desired; embedded blanks in parameters are not permitted. Any EDIT command may be optionally preceded by a period (.), which will suppress all output.

except error messages, resulting from the command. EDIT commands cannot be continued to a second line.

"String" Format

Where EDIT commands require one or more character strings as parameters, the "string" may be delimited by any non-numeric character (except a blank or a minus sign), that does not appear within the body of the string itself. The colon (:) is a reserved delimiter; if used, it indicates that a column-dependent correspondence between characters in the string and characters in the line is necessary for a match. The up-arrow (↑ or ^) is a reserved character within the body of the "string"; if used, it indicates that any character in the corresponding position in the line is acceptable as a match. The terminal delimiter of the string(s) is necessary only if further parameters follow the string specification, or trailing blanks are to be included as part of the string.

Two consecutive delimiters define a "null" string.

Editor Error Messages

Message	Description
CMND?	Illegal EDIT command.
STRING?	Illegal string specification, or missing string. e.g.: Required string missing for ME; Second string missing for R; double null string specified for R. This message may also occur as a result of an illegal numeric parameter specification, which causes a part of the numeric to appear as if it were a string.
COL#?	Illegal characters follow the recognized end of the command, or illegal format for the column-number limit specification.
SEQN?	Out-of-sequence update; updating must be done in an ascending sequence until an F command is entered.
EOF:m	End-of-item reached at line-number m; this message may appear with a number of commands, and is not usually an error.
TOP	Top-of-item reached; informative message that is not usually an error.

The Input Environment

The INPUT environment, used for data entry, is entered when lines are to be added to the item, or existing lines are to be replaced. This environment is entered by the "I" or "R" commands. The EDITOR requests data by prompting with the line-number to which data is being entered. The continuation character (0^{cs}) may be used to continue data input to the same line. All characters input, including trailing blanks, will be stored without change. A null input (carriage return or line feed only) will terminate the data request, and cause a return to the EDIT environment for the next EDIT command. If null lines are required, it is necessary to create these lines with a fill character such as "&" then prior to permanently filing the item replace with a null using the replace operator such as R99/&//.

Edit Commands

Parameters in brackets are optional.

Command	Description	Message
A	AGAIN, Repeats the last LOCATE command that had a "string" specification.	CMND? if no such command had been previously issued. EOF:m if EOF is reached.
B	BOTTOM of item - the current line pointer is set to the last line in the item.	EOF:m "m" is the last line of the item.
DE [n] "string" [p [-q]]	DELETE. This command causes a search for characters matching the "string"; the search is not column-dependent unless the delimiter used in the "string" is a colon (see comment under "string" format). The search is restricted to column p, or columns p through q, if specified. If q < p, q=p is assumed. If n is specified, n lines, starting from the current line plus one, will be scanned for a matching "string", and every line with a match is deleted. Lines that are deleted are listed. The current line pointer is unchanged.	EOF:m if EOF is encountered.

Command	Description	Message
DE [n]	DELETE. n lines, (one if n is not specified) are deleted from the item, starting from the current line. The current line pointer is not changed.	EOF:m if EOF is reached
EX	EXIT. The EDIT is terminated; the item is not updated to the diskfile; control is returned to TCL.	EXIT
F	Updates are merged with the previously existing item, and the current line pointer is set to zero.	TOP
FD	FILE-DELETE. The item is deleted from the disk-file; the EDIT is terminated, and control returned to TCL.	'item-name' DELETED
FI	ITEM-NAME-SAVE. The item is updated to the disk-file, and control is returned to TCL.	'item-name' FILED
FS	FILE-SAVE. The item is updated to the disk-file, and EDIT is re-entered.	TOP
G n	GO TO. The current line pointer is set to n, and that line is listed.	EOF:m if n > m TOP
I	INPUT. The input environment is entered. All subsequent lines are considered as data input lines to the item, and are inserted following the current line. (If the current line pointer is at zero (TOP), lines are inserted before the first line of the item). Input is prompted with the current line-number and a plus sign (+). A null input terminates the data request, and the EDIT environment is re-enterd.	

Command	Description	Message
I data	INSERT. One line may be inserted following the current line, by entering I, one blank, and the data to be inserted. Note that the data is separated from the I by one blank only: all other blanks will be considered as part of the line that is inserted. The line-continuation character cannot be used to continue data beyond one physical line.	
L [n]	LIST. n lines (one if n is omitted), starting from the current line plus one, will be listed. The current line pointer is set to the last line that is listed on the terminal.	EOF:m if EOF is encountered.
L[n]"string"[p[-q]]	LOCATE. This command causes a search for characters matching the "string"; the search is not column-dependent unless the delimiter used in the "string" is a colon (see comment under "string" format). The search is restricted to column p, or columns p through q, if specified. If q<p, q=p is assumed. If n is not specified, the next occurrence of "string" is located, and that line is listed; the current line pointer is set at the line that is listed. If n is specified, n lines, starting from the current line plus one, are scanned for the occurrence of "string"; all lines in which the "string" is found, are listed. The current line pointer is incremented by n, and may not be at the last line listed. Note that the scan always begins from the current line plus one.	EOF:m if EOF is encountered.

Command	Description	Message
ME[n]"string"[p]	MERGE. n lines (one if n is omitted), starting from line-number p (one if omitted) of the item specified in "string" are merged into the item being edited. Lines are inserted following the current line. The item specified in "string" must be the same file as the item being edited. If "string" is null, lines are merged from the item being edited itself.	NOT ON FILE If the item from which lines are to be merged is not on file. EOF;m If EOF is encountered in the item from which the merge is taking place.
N[n]	NEXT. The current line pointer is incremented by n (one if n is omitted), and that line is listed.	EOF:m If EOF is encountered.
P	PRE-STORED command call. The EDIT command that has been pre-stored is called into effect (see below). Note:Pre-stored item will remain through successive innovations of the EDITOR.	CMND? If no command had been pre-stored; any EDIT error message may be caused by error in the format of the pre-stored command.
P data	PRE-STORE command. Any EDIT command can be pre-stored by entering P, one blank, and the EDIT command. This pre-stored command can then be called into effect by entering P only (the pre-stored command call). Only one command can be pre-stored.	
R[n]"string$_1$"string$_2$"[p[-q]]	REPLACE. The first occurrence of "string$_1$" is replaced by "string$_2$" in n lines (one if n omitted), starting from the current line. The search is restricted to column p, or columns p through q, if specified. If q < p, q=p is assumed. The format of "string$_1$" and "string$_2$" is as described under "string" formats; note that only one delimiter separates the strings.	EOF:m If EOF is encountered.

Command	Description	Message
R[n]"string$_1$"string$_2$"[p[-q]] (Continued)	Lines that are changed are listed in their updated form. The current line pointer is unchanged.	
R[n]	REPLACE. The INPUT environment is entered, and input requested for data to replace n lines (one if n is omitted), starting from the current line. The EDIT environment is re-entered when either: 1) data for the specified number of lines has been entered, or 2) a null line is entered. In the latter case, the remainder of the lines remain unchanged. The current line printer is unchanged.	EOF:m If EOF is encountered.
S	SUPPRESS line numbers on/off. Each entry of S acts as an alternate-action switch to either suppress or list line-numbers when lines are listed. On initial entry to EDIT, line-numbers are listed.	
T	TOP. The current line pointer is set to zero.	TOP
TB	TABS. This command can be followed by up to 16 tab settings in ascending order. For example, TB 15 20 31 40 would set four tab stops. Tabbing is invoked whenever the EDITOR is in the INPUT environment and a control-I (I^C), or on some terminals a tab key, is hit. The I^C will cause a series of blanks to be output, moving the cursor to the next specified tab stop. A backspace (H^C) and cancel (X^C) will backspace over tabs.	

Command	Description	Message
U[n]	UP. The current line pointer is decremented by n (zero if omitted), and that line is listed. Note that, if n is omitted or is zero, the current line is listed.	TOP If top-of-file is reached.
X	X deletes the effects of the last update command (DE,I,R) that had been entered. Only one update, the last, can be deleted. This is useful when an erroneous update command has been used, since, in order to repeat the command without the X, an F is needed.	L:n n is the last update line-number.
Z[n[-m]]	ZONE. Sets print column limits for listing output of lines. (Does not affect the search in LOCATE/REPLACE commands). If no parameters are used, the zone is reset so that the entire line will be listed on output. If m<n, m=n is assumed.	
?	Interrogates the current line-number.	L:n n is the current line-number.
null	A null line (carriage return or line feed only) lists the next line. This is equivalent to an L command with no parameters.	EOF:m if EOF is encountered.

Section IX

SYSTEM COMMANDS

INTRODUCTION

A set of standard commands are available to the user which provide system capabilities. These commands include tape handling, output spooling, card reading, interuser communication and other miscellaneous system utilities.

Arithmetic Commands

The following is a list of commands which provide simple arithmetic capabilities.

Command			Function
ADDD	n_1	n_2	Add n_1 and n_2 in decimal.
ADDX	h_1	h_2	Add h_1 and h_2 in hex.
SUBD	n_1	n_2	Subtract n_2 from n_1 in decimal.
SUBX	h_1	h_2	Subtract h_2 from h_1 in hex.
MULD	n_1	n_2	Multiply n_1 by n_2 in decimal.
MULX	h_1	h_2	Multiply h_1 by h_2 in hex.
DIVD	n_1	n_2	Divide n_1 by n_2 in decimal.
DIVX	h_1	h_2	Divide h_1 by h_2 in hex.
DTX	n_1		Convert n_1 from decimal to hex.
XTD	h		Convert h from hex to decimal.

Negative numbers are designated by appending a minus sign immediately prior to the first digit of decimal numbers. Hexadecimal numbers are considered negative when in a range from FFFFFFFF to 80000000. (Note: if fewer than 8 hex characters are keyed in, high order zeroes are assumed.) The largest positive number which can be handled is 2,147,483,647 in decimal or 7FFFFFFF in hexadecimal.

Card Reader Command

C-READ file-name item-id [(options)]

This verb reads cards and adds them to the end of the existing item (new items are also acceptable). Each card becomes a new attribute at the end of the item. The item is filed away in the event that the hopper becomes empty, an EBCDIC error occurs, a mechanical error occurs, or the item becomes greater than 32,000 bytes. Appropriate messages accompany each of these terminating conditions. The normal termination is hopper empty, implying end of card deck. If the card reader is taken out of ready manually, the message "CRD RDR NOT RDY" is output to the terminal once every ten seconds until the card reader is readied.

If an item becomes larger than 32,000 bytes, the command must be repeated using a different item-id. Linkage between the items must be supplied by the user. A maximum size of 32,000 was chosen to allow room in the item to add such linkage.

Valid options are:

n-m	to limit the use of card columns n through m for data to the item.
L	to remove leading blanks.
T	to remove trailing blanks.

If either the L or T option is used, a blank card is stored as a null attribute.

Note that an item larger than 3,000 bytes will cause a forward link zero unless additional work frames are requested in the LOGON item in the SYSTEM Dictionary. If items as large as 32,000 bytes are expected, your LOGON item should request 66 frames.

It is the user's responsibility to make certain that the card reader is not used from two or more terminals simultaneously. If this should occur, the most common result will be that the second terminal attempting to use the card reader will be told that the card reader is not ready. The least common result is that the first terminal may lose activation and not regain activation until after a boot load.

Tape Commands

The following is a list of commands which provide basic capabilities with the magnetic tape unit. Each is described in more detail below.

Command	Function
EBTPRD	Read records from tape into file items.
T-ATT	Attach magnetic tape unit.
T-BCK	Backspace tape.
T-DET	Detach magnetic tape unit.
T-DUMP	Dump file items to tape.
T-FWD	Forward space tape.
T-LOAD	Load file items from tape.
T-RDLBL	Read tape label.
T-READ	Read one record from tape.
T-REW	Rewind tape.
T-WEOF	Write end-of-file on tape.

EBTPRD File-name

This command reads sequential records from tape, converts each record from EBCDIC to ASCII, creates a sequentially numbered item-id for each record converted and stores it in the new file item.

The user is prompted for the record length in bytes. The maximum allowable record length is 32,000 bytes.

T-ATT

This command causes the processor to allocate the magnetic tape unit to the terminal issuing the command. It must be issued before tape is used. Other users are locked out. If the tape is attached to some other use, the message IN USE will be returned. Logging off the system (OFF command) automatically detaches the magnetic tape unit.

T-BCK [n]

This command causes the processor to back the tape n records. If n is not specified, default is a backspace to the position immediately preceding the previous EOF mark, or to the load point. Before reading the next record, a T-FWD must be issued to position the tape after the EOF mark.

T-DET

This command causes the processor to make the magnetic tape unit available to the system. The unit is automatically detached when the user logs off.

T-DUMP file-name ('item-list') (selection-criteria)

Allows the user to selectively dump his dictionaries or data files to the magnetic tape. (ENGLISH verb).

The 'item-list' and "selection-criteria" are described under the ENGLISH section, and cause a selected sub-set of the items or the file to be dumped. Absence of both causes all items to be dumped. The file name may be preceded by the DICT modifier to dump dictionary data. File definition items (D/CODE=D) will not be dumped, an EOF mark is written to the tape after the dump.

Note: As in other ENGLISH statements, each item-id must be enclosed in single quotes.

T-FWD n

This command causes the processor to move the tape forward n records (maximum value for n = 32,767). If n is not specified, the tape forward spaces to the position immediately beyond the next EOF mark.

T-LOAD file-name [(N)]

This command allows the user to load dictionaries or data saved by the T-DUMP verb. The data from the tape is loaded to the file "file-name". Items on the file with item-id's corresponding to those on the tape will be over-written. The file name may be preceded by the DICT modifier to load dictionary data. The tape is positioned at the EOF mark at the conclusion of the load. If the (N) is present, item-id's will not be listed as they are loaded.

T-RDLBL n

This command will read and store the label from tape reel number "n" ("n" in hexadecimal), if the tape is at load point. This verb must be used to initialize the internal label storage area, and is needed under either of the following conditions:

a) Data is to be written to a tape (T-DUMP, etc.) starting at other than the load point of reel number one.

b) Data is to be read from a tape (T-LOAD, SEL-RESTORE, etc.) starting at other than the load point of reel number one.

The label data is maintained during a logon session, and therefore, T-RDLBL need be used only once when working on any particular multi-reel tape set. As an example, suppose there is a three-reel file-save, and it is desired to start a SEL-RESTORE from reel #2 (since SEL-RESTORE can be started in the middle of the file-save):

```
  Reel 1              Reel 2          Reel 3
BOT . . . . . .EOT BOT . . . .EOT BOT. . . . . .EOF
                   start
```

If tape #2 reaches EOT, the system must be able to prompt for real #3; therefore, it is imperative that prior to the SEL-RESTORE, reel #2 is mounted at load point, and the command T-RDLBL 2 Ⓡ is used to initialize the label save buffer.

T-READ [(options)]

This command dumps tape to terminal, or to the line printer. The T-READ command terminates when the specified number of records are read, or when an EOF mark is detected.

Valid options are listed below; multiple options are separated by commas.

X — dump in hexadecimal instead of character format.

P — dump to the line printer

n_1-n_2 — dump only specified tape records, counting from the current position of the tape. If $n_1 > 1$, n_1-1 records are bypassed before displaying the N_1 record. The tape stops positioned at record n_2+1.

If n_2 is omitted, n_2=n_1 is assumed.

If (n_1-n_2) is omitted, tape records 1 through EOF (or EOT) are displayed.

EXAMPLE--

T-READ (4-6) bypasses 3 records, displays the 4th, 5th, and 6th records, and the tape is positioned at the beginning of the 7th record.

T-REW

A rewind command is issued to the tape unit and control returns immediately to TCL.

T-WEOF

Write and end-of-file (EOF) on the tape.

Tape Labels

A tape label may be written at the beginning of each tape reel; the label may consist of up to sixteen characters, and the system adds the time, date, and the reel number. Labels are specified when invoking either the FILE-SAVE or T-DUMP processor; in the case of a FILE-SAVE, the label is written after the cold-start section of the tape, if specified, is written. T-DUMP will cause the label to be written only if the tape is at the load point; it will be ignored otherwise.

Labeled tapes are created as follows:

1) T-DUMP

   T-DUMP ..... HEADER "xxxxxx" ⓡ

   The data enclosed in double-quotes following the 'HEADER' connective is used as the label. Omitting the 'HEADER' connective causes unlabeled tapes to be generated.

2) FILE-SAVE

   :DDUMP xxxxx ⓡ

   The data following the :DDUMP verb is used as the label. If no data appears, unlabeled tapes are generated.

   Notes: The characters ' " ^] [\ should not appear as part of the label. Labels can only be up to 16 characters; data beyond 16 characters will be truncated.

Tape labels are read as follows:

The tape label is read by the FILE-RESTORE, T-LOAD and T-RDLBL processors. The first will always read the first record presented to it (when an 'A', 'AF', or 'F' option is entered); the latter two will attempt to read the label only if the tape is at the load point. In the case of unlabeled tapes, the processor will read the first tape record, determine that the tape is unlabeled, and backspace the tape by one record before continuing.

Multiple Reel Tape Files

When a reel reaches the end-of-tape marker, the system will issue a 'tape rewind' command, and print a message requesting the next reel of tape. An asterisk is printed as a prompt for input. When the next

reel has been mounted, the process may be continued by entering a control-shift-0 character; all other entries will be ignored. (The process is roadblocked on a READ instruction, and will therefore echo up to eleven non-control characters before re-issuing the asterisk prompt; any control character other than the 0^{cs} will cause an asterisk prompt immediately). If the tape unit is not ready or not on-line, the asterisk will be re-issued, and another 0^{cs} character should be entered when the tape is ready. The prompts cannot be suppressed, and the input to the asterisk prompt cannot be "stacked" by a PROC.

Messages Relating to Multiple-Reel Files:

1. MOUNT REEL # xx; LABEL = label time date*

   The requested tape is to be mounted, and a control-shift-0 character typed when the tape unit is ready. If unlabeled tapes are in use, the "LABEL =" etc. part of the message will be suppressed. "xx" is the two-digit hexadecimal reel number required.

2. INCORRECT REEL# *

   This message is returned when the reel number on the labeled tape does not match the requested reel number (or if the first tape mounted is not reel #1). Mount the correct tape and type an 0^{cs}. This message cannot appear if unlabeled tapes are in use.

3. INCORRECT LABEL*

   Self-evident; action as above

4. REEL #1 WAS UNLABELED *

   A labeled tape has been mounted, when the first tape reel was unlabeled. Action as above.

5. REEL #1 WAS LABELED *

   Converse of (4); action as above.

Output Spooler Commands

The following is a list of commands which provide output spooling capabilities. Each is described in more detail below.

Command	Function
ASSIGN	Assign output spooler device.
EJECT	Eject line printer pages.

Command	Function
FORM	Set form alignment for output spooler.
KILL	Abort current spooler output.
LOAD-SPOOLER	Load spooler from SYSTEM-MODES.
P-ATT	Attach line printer.
P-ATT-KILL	Unconditionally detach line printer from any line.
P-DET	Detach line printer.
P-STAT	Display line printer status.
PRINT-HOLD	Print hold file on line printer.
PRINT-QUE	Display hold file queues.
PRINT-TAPE	Print tape file on line printer.

DEVICE ASSIGNMENT Verb

ASSIGN options

The default device is always the line printer. Device assignment is on a user line-by-line basis. Each user can issue an ASSIGN verb at any time, which will remain in effect until a new ASSIGN verb has been issued for that line or until LOGON resets it.

The spooler can spool to multiple devices simultaneously. The options for ASSIGN are summarized below:

C -Output to the terminal connected to the communication line on which the spooler is running.

H -Hold the file on disc. The device is the disc and the file is retained for future printing.

N -No spooling to disc. Output is line-at-a-time directly to the line printer. If N is set, all other options are ignored.

S -Suppress line printer printout.

T -Output to tape from wherever the tape is currently positioned. Write an end-of-File (EOF) mark when a close command is received (TCL-II closes all files).

The ASSIGN options are input in free form format. Blanks and unrecognizable characters are ignored.

EXAMPLE--

```
ASSIGN
```

(Resets all the option flag bits the same as LOGON does. Output is spooled to the line printer only and is not held.)

EXAMPLE--

```
ASSIGN N T C
```

(Output line-by-line to line printer. Ignore T and C).

EXAMPLE--

```
ASSIGN H T S C
```

(Hold the file on disc, output to tape, suppress line printer printout, and output to console connected to the communication line on which the spooler is running).

EXAMPLE--

```
ASSIGN S
```

(Any file will be deleted when it is next "printed").

Each of these options is explained in more detail below.

CONSOLE OUTPUT (C)

The spooler can spool output to a hard-copy terminal or CRT connected to the communication line assigned to the spooler. The "C" option will enable console output. The spooler will output the data regardless of whether a console is actually connected. The line printer parameters (lines per page, etc.) will apply to the console.

Note that the spooler will hang permanently if the "C" option is used and an asynchronous communications channel (2613 or 2614) does not exist for the spooler line. The first character output will cause the spooler to wait forever for an interrupt.

HOLD FILE OPTION (H)

The spooler has two main queues. The first is a 32 entry queue which contains information on each currently open print file. An open file is one that has not been closed and is still having frames of print output added to it.

The second queue contains entries for up to 32 closed files; i.e., those files that are ready for printing. After a file has started printing it is normally purged from the second queue unless it is flagged as a Hold file. A Hold file will occupy one of the 32 queue entries until it is printed without the hold option being set by ASSIGN. When a Hold file is entered in the print queue it is assigned a decimal entry number which is printed as follows:

```
ENTRY NO. 7
```

A Hold file is kept on the disc until released. A verb to interrogate and edit the hold queue entries is the PRINT-QUE verb. The verb is input without any parameters. A series of messages will then be output, requiring a response. They are as follows:

ENTRY 5	(Queue entry flagged as hold)
LIST FRAME (Y/N)	("Y" reply will list first Frame of the file on your terminal)
PRINT (Y/N)	("Y" reply will print the file according to the presently set ASSIGN parameters. If H is not set the file will be released as it is printed. If only the parameter S is set then the file will be deleted.)
ENTRY 7	(Next queue entry flagged as hold).

After the entire queue has been searched for hold entries the message

```
[283] END OF PRINT QUEUE
```

is output to the user.

NO SPOOL OPTION (N)

The "N" option specifies no spooling. The tape test, etc., will be bypassed. The data is output a line-at-a-time directly to the line printer. All other options are ignored. This option should normally only be used when the disc is getting full and space is a problem or for forms alignment.

The "ASSIGN N" causes an automatic attachment to the line printer the first time WRTLIN is called. The user must use the P-DET verb to detach the line printer when finished. The line-at-a-time mode is a special mode where the user usually wants to gain exclusive control of the printer for a time in order to print several files or do forms alignment.

If the line printer is attached to another line the first time WRTLIN is called, the following message will be output to the user console over and over until the other line is detached:

LPTR ATTACHED TO ANOTHER LINE

The line-at-a-time mode and the normal spooler mode will blank out the user buffer from OBBEG through OB each time WRTLIN is called unless the NOBLNK option is used when calling WRTLIN.

SUPPRESS LINE PRINTER OPTION (S)

The "S" option on an ASSIGN will suppress line printer output. Normally, the "S" is used in conjunction with "T" and/or "C" options. However, if the "S" appears alone, the file will be effectively deleted since the output device is null and the frames of the file are released.

TAPE FILES OPTION (T)

If the "T" option is used on an ASSIGN, then the file spooled will be written to magnetic tape in 500 byte records, starting wherever the tape is positioned. After the last record is written, an end-of-file (EOF) mark is written.

The magnetic tape is never automatically rewound because files may be "stacked" on the tape.

The user must mount the tape (with write-ring), attach the tape unit, and rewind or position the tape where the file is to be written. The tape must be left attached because the spooler will automatically write on the tape without attaching. This means that the spooler will use the tape in this situation even if another process is currently using it. It is the user's responsibility to see that this does not occur.

EJECT Verb

EJECT n

The line printer will be attached, n pages ejected and the line printer detached. The number n cannot exceed 10 or else the message [286] NO. OF PAGE EJECTS GREATER THAN 10 will be output to the console and an exit taken to TCL.

FORM Verb

FORM n m

FORM is a verb which is used for forms alignment on a Hold File queue entry n. The first m lines are output to the line printer. The message

AGAIN? (Y/N)-

is output after each m lines. The response "Y" will output the m lines again. The only restriction is that the m lines must be on the first frame of the file.

The forms alignment uses the line-at-a-time feature of the spooler. The present ASSIGN bits are saved before the alignment and restored afterwards. The spooler routine PPUT is called to output each line.

The PPUT routine will attach the line printer when it is available. The FORM routine detaches from the line printer after "N" is typed in response to AGAIN?

Four error messages can be output with FORM:

[284] INPUT ENTRY MUST BE 1-32

[285] ENTRY IS NOT A HOLD FILE

[286] NO. OF LINES MISSING

[287] FILE BUSY BEING PRINTED

KILL Verb

KILL

This verb unconditionally aborts the current spooler job in execution or the next one to be executed. A Hold file will be preserved while a normal print file will have its frames released.

LOAD-SPOOLER PROC

LOAD-SPOOLER

This PROC loads eight frames of the spooler from SYSTEM-MODES. This usually followed by a START-SPOOLER PROC. The LOAD-SPOOLER does not destroy the print queue containing Hold files. However, the START-SPOOLER PROC will clear the Hold file queue if the reply is 'Y' to the reset question. The disc space will not be released in this case.

P-ATT Verb
P-ATT

The P-ATT is a verb which attaches the user to the line printer. If the user is attached, only the TCL prompt comes back. If the printer was already attached to another communication line, the following message is returned:

[289] LINE PRINTER ATTACHED TO LINE X

P-ATT-KILL Verb
P-ATT-KILL

The P-ATT-KILL verb unconditionally detaches the line printer from whatever line is currently attached to it. The spooler attaches and detaches from the line printer at the beginning and end of each print file. Thus, a user can normally get attached between spooler print files.

The spooler assumes that it remains attached to the printer once it becomes attached. The spooler unconditionally outputs to the line printer each frame without testing to see if it is still attached.

P-ATT-KILL should only be used under the SYSPROG account and only when a line has inadvertantly remained attached to the line printer.

P-DET Verb
P-DET

The P-DET detaches the user from the line printer if the line is presently attached and an exit taken to TCL.

The LOGOFF routine will automatically detach a line from the line printer if it is attached. A user can cause the spooler to wait if he attaches to the line printer and never detaches or logs off.

P-STAT Verb
P-STAT

P-STAT performs a printer status. One of four messages will be output:

1) *PRINTER READY*

2) *LPTR OFF-LINE*

3) *LPTR PRINTING*

4) *LPTR CABLE OFF*

PRINT-HOLD Verb

PRINT-HOLD n ["string"]

The Hold file queue entry n will print according to the current ASSIGN options. The Hold file will be deleted unless the H option is set. For example, to copy a Hold file to magnetic tape without releasing the Hold file, use ASSIGN T H followed by PRINT-HOLD.

Three error messages can be output at this point:

[284] INPUT ENTRY NO. MUST BE 1-32

[285] ENTRY IS NOT A HOLD FILE

[287] FILE BUSY BEING PRINTED

If the option string is used, a search for the string will be made and the output started at the beginning of the string. If the string is not found, the following error message is printed:

290 STRING NOT FOUND IN HOLD FILE

Thus, a Hold file can be reprinted starting at any point.

PRINT-QUE Verb

PRINT-QUE

Used to interrogate and print hold file queues (see ATTACH HOLD FILE option).

PRINT-TAPE PROC
PRINT-TAPE ["string"]

This PROC does the following:

1) Attaches to the tape unit.

2) Rewinds the tape.

3) Attaches to the line printer

4) Prints one tape file. An optional string can be specified as a starting point.

5) Detach the line printer.

6) Detach the tape unit.

The rewind command can be removed from the PROC if a manual rewind is used. Thus, a user could print multiple files stacked on the tape.

An assembly language routine is called to print the tape. It also attaches to the line printer unit and will spin until attached. The attach in the PROC will give a message if another user is presently attached to the line printer or mag tape.

Summary of Spooler Error Messages

SPOOLING TO LPTR:

LPTR ATT. TO ANOTHER LINE

The line printer is attached to another line. The spooler will wait until the printer is free before printing. This message is output when the print file has been closed and is ready for printing.

LPTR CABLE OFF

The line printer connector is loose or the cable is bad. The spooler will spin and wait for good status.

LPTR OFF-LINE

Line printer is powered off, the off-line switch is in the off-line position, or the paper has broken. The message is printed just as the file is queued for printing. The spooler will process the file as normal, then it will spin and wait for the line printer to go ready.

LPTR PRINTING

The line printer was busy at the time of status. Another spooled file is being printed. This message does not come out every time because its timing is dependent on line printer state. Thus, the line printer may be printing without getting the message.

SPOOLING TO MAG TAPE:

MAG TAPE ATTACHED TO ANOTHER LINE TYPE (A) TO ABORT OR (G) TO TRY AGAIN (A/G)

The user should go to another terminal and detach the tape unit (or hit break and detach by using debugger) and type 'G' to try again.

The 'A' response will release all the spooled frames and exit to TCL.

MOUNT WRITE RING ON MAG TAPE AND TYPE C/R TO GO

This message allows the user to put a write ring on the tape without aborting the job and losing spooled output. Merely, mount the write ring, put the tape at load point and type carriage return to try again.

MISC ERROR MESSAGES:

OPEN FILE PRINT QUE FULL

The 32 entry queue of open print files is full. These are files that have not been closed and are not ready for printing. Exiting to TCL will close all the files for a line. Loading the program OPNPF from SYSTEM-MODES will clear the queue.

This should only be done when loading the spooler. The LOAD-SPOOLER proc will re-load OPNPF.

PRINT QUE FULL

The 32 entry queue of closed print files is full. The spooler will do a release-quantum for 40 times, then try to put an entry in the queue again. Consequently, the message will be repeated until the entry is made.

This queue also holds all of the hold files. Some of the hold files may have to be deleted. It is not recommended to leave hold files on the disc for any long periods of time. Five hundred print characters (including blanks) occupy one frame on the disc.

Miscellaneous Commands

BLOCK-PRINT, BLOCK-TERM

These verbs will print characters in a 9 by n block form on either the line printer or the user's terminal respectively. Any ASCII character may be printed. Each word in the input line will cause a carriage return and three line feeds prior to printing the word in block form. Any word containing single quotes (') must be contained within double quotes ("), and vice versa. The surrounding quotes will not appear in the output.

The program uses a file named BLOCK-CONVERT to create the blocked characters. A BLOCK-CONVERT file already exists which contains the conversion specifications for all printable ASCII characters (no lower case alphas, however). With this file characters will be printed as 9 by 12 to 9 by 20 blocks. If it is desired to change the way any character is printed, it is necessary to change the corresponding item in the BLOCK-CONVERT file. The item-id of the item is the character to be converted. Each item in the file must consist of exactly ten (10) attributes; the first must specify in decimal the number of horizontal bytes in the blocked character to be output, (i.e., n of the 9 by n block mentioned above). The second and subsequent attributes provide the conversion specification. These attributes contain one or more values; each value except the last is separated from the preceding by a value mark (VM), hex "FD". The first character of the first value in each attribute must be "C" or "B". These signal that the output matrix line of the blocked character begins with a character or a blank respectively. Immediately following must be the number of characters or blanks in decimal. The presence of a value mark indicates a switch from character to blank status or vice versa and must be followed by the number of bytes to be output. The process continues until the attribute mark at the end of the current line.

For example an "X" might be specified as follows:

001	7	blocked character is 7 bytes wide
002	C1]5]1	output 1 char, 5 blanks, and 1 char
003	B1]1]3]1]1	output 1 blank, 1 char, 3 blanks, 1 char, 1 blank
004	B2]1]1]1]2	output 2 blanks, 1 char, 1 blank, 1 char, 2 blanks

005	B3]1]3	output 3 blanks, 1 char, 3 blanks
006	B2]1]1]1]2	output 2 blanks, 1 char, 1 blank, 1 char, 2 blanks
007	B1]1]3]1]1	output 1 blank, 1 char, 3 blanks, 1 char, 1 blank
008	C1]5]1	output 1 char, 5 blanks, 1 char
009	B7	output 7 blanks
010	B7	output 7 blanks

Words to be blocked cannot have more than nine (9) characters, and in addition, the total number of bytes (including three (3) blanks separating each blocked character in a word cannot exceed the current line length set by the last TERM verb.)

The following error message will be produced if the corresponding error occurs.

[520] NO DATA FOR BLOCK OUTPUT (A string of characters to be blocked did not follow the verb)

[521] TOO MANY CHARACTERS IN WORD TO BLOCK (more than nine characters specified in a word)

[522] BLOCK CONVERT FILE MISSING OR IMPROPERLY DEFINED

[523] BLOCK OUTPUT WOULD EXCEED PAGE WIDTH (see discussion above)

[524] INPUT CHARACTER 'x' IS NOT IN BLOCK CONVERT FILE

[525] INPUT CHARACTER 'x' IS IMPROPERLY FORMATTED IN BLOCK CONVERT FILE

An example of BLOCK-PRINT appears on the next page.

Debug

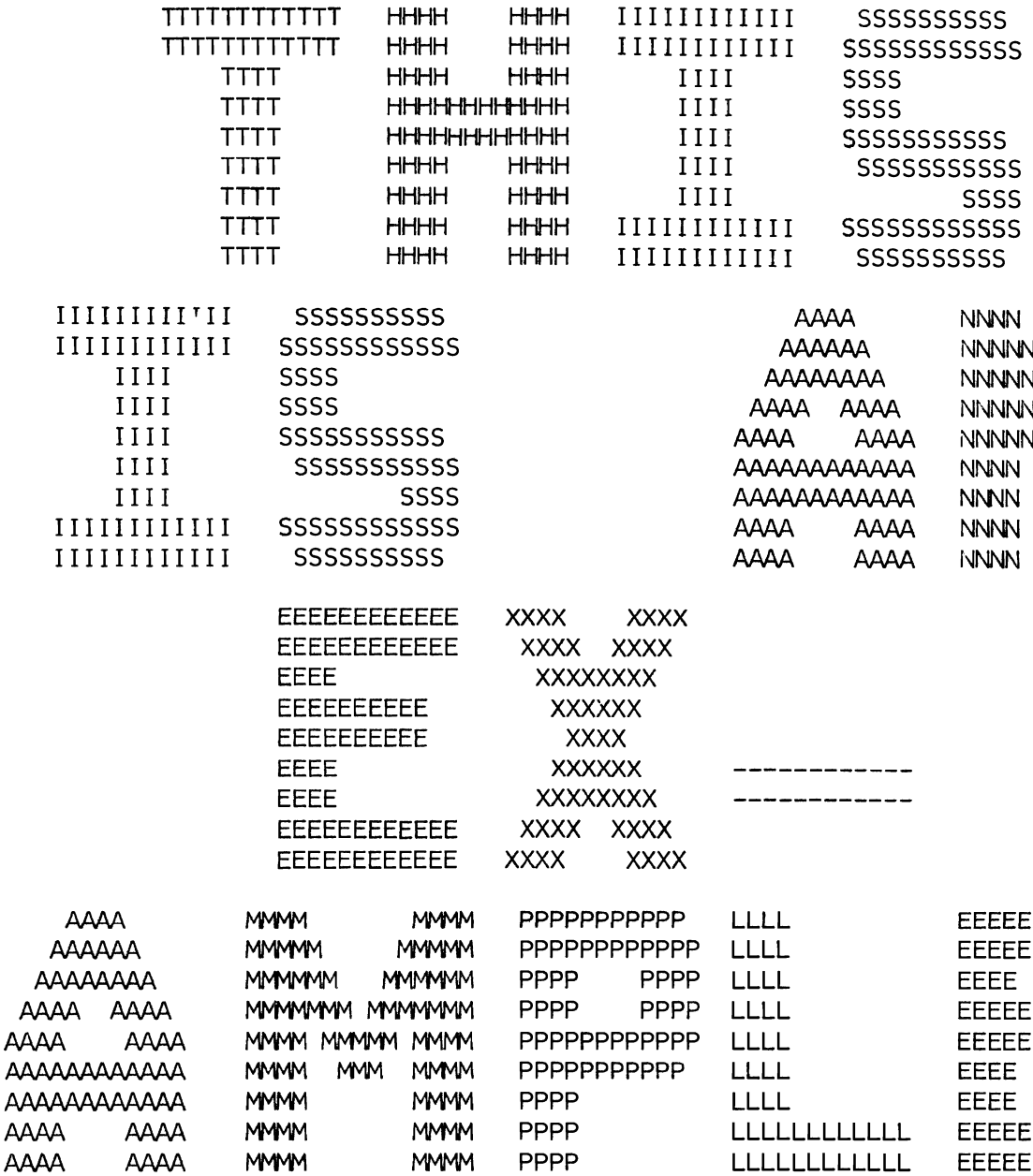
The user can always terminate execution by depressing the break-key. This interrupts the process and enters the debug package. The following alternatives are available:

G Resume processing

END Discontinue processing return to TCL

OFF Discontinue processing exit to log-off

P Disable printing (normally followed by G)


TTTTTTTTTTTT HHHH HHHH IIIIIIIIIIII SSSSSSSSS
TTTTTTTTTTTT HHHH HHHH IIIIIIIIIIII SSSSSSSSSSS
TTTT HHHH HHHH IIII SSSS
TTTT HHHHHHHHHHHH IIII SSSS
TTTT HHHHHHHHHHHH IIII SSSSSSSSSS
TTTT HHHH HHHH IIII SSSSSSSSSS
TTTT HHHH HHHH IIII SSSS
TTTT HHHH HHHH IIIIIIIIIIII SSSSSSSSSS
TTTT HHHH HHHH IIIIIIIIIIII SSSSSSSSS
IIIIIIIII'II SSSSSSSSS
IIIIIIIIIIII SSSSSSSSSS
IIII SSSS
IIII SSSS
IIII SSSSSSSSSS
IIII SSSSSSSSSS
IIII SSSS
IIIIIIIIIIII SSSSSSSSSS
IIIIIIIIIIII SSSSSSSSS
AAAA NNNN NNNN
AAAAAA NNNNN NNNN
AAAAAAAA NNNNNN NNNN
AAAA AAAA NNNNNNN NNNN
AAAA AAAA NNNNNNNNNNNN
AAAAAAAAAAAA NNNN NNNNNN
AAAAAAAAAAAA NNNN NNNNNN
AAAA AAAA NNNN NNNNN
AAAA AAAA NNNN NNNN
EEEEEEEEEEEE XXXX XXXX
EEEEEEEEEEEE XXXX XXXX
EEEE XXXXXXXX
EEEEEEEEEE XXXXXX
EEEEEEEEEE XXXX
EEEE XXXXXX
EEEE XXXXXXXX
EEEEEEEEEEEE XXXX XXXX
EEEEEEEEEEEE XXXX XXXX
AAAA MMMM MMMM PPPPPPPPPPP LLLL EEEEEEEEEEEE
AAAAAA MMMMM MMMMM PPPPPPPPPPPP LLLL EEEEEEEEEEEE
AAAAAAAA MMMMMM MMMMMM PPPP PPPP LLLL EEEE
AAAA AAAA MMMMMMMM MMMMMMMM PPPP PPPP LLLL EEEEEEEEEE
AAAA AAAA MMMM MMMMM MMMM PPPPPPPPPPPP LLLL EEEEEEEEEE
AAAAAAAAAAAA MMMM MMM MMMM PPPPPPPPPPP LLLL EEEE
AAAAAAAAAAAA MMMM MMMM PPPP LLLL EEEE
AAAA AAAA MMMM MMMM PPPP LLLLLLLLLLLL EEEEEEEEEEEE
AAAA AAAA MMMM MMMM PPPP LLLLLLLLLLLL EEEEEEEEEEEE

At the TCL level, the session is terminated by typing the command OFF.

DUMP

Reality provides a means of examining the virtual memory frames from the disc via the dump command. The user types the verb DUMP plus a beginning and an ending FID. The appropriate frames are displayed on the terminal.

The dump command has the following format:

DUMP fid_1, fid_2 (options)

Valid options are listed below; multiple options are separated by commas:

P — Dump to the line printer instead of the terminal

n-m — Character count to restrict dump to characters "n" through "m"; only valid if "L" option is not specified

L — Dumps "link" fields only (no data dump)

G — Dumps "group" data - all data for the beginning of a linked frame set through the last frame in the group. Only fid_1 is valid if G is present.

If the optional G parameter is not specified the data in frames Fid through fid_2 will be dumped; preceding each 500 bytes of data dumped for each frame, the links are displayed. Specifying the L option inhibits the data dump and dumps only the links. If only the first fid_1 is specified, fid_1 onward will be dumped.

MESSAGE, MSG

All Reality system users may communicate with the other users on the system. To transmit a message to another user, type the verb MESSAGE, or MSG, followed by the user's account name, then the text of the message. The maximum message is 108 characters long. Anyone currently active on the account referenced will receive the message. If the user is not presently logged on, the system will respond with "USER NOT LOGGED ON". Users with systems privileges level-two may broadcast to all active users by typing MESSAGE * followed by the message text.

EXAMPLE--

```
MESSAGE * GOOD MORNING
```

TERM

Different terminal characteristics may be accommodated through the use of the TERM command. The format of the command is as follows:

```
:TERM    p1,p2,p3,p4,p5,p6
```

Individual parameters may be null and, if null, the previously defined parameter remains in force. (Refer to LOGON/LOGOFF section). The parameter definitions are:

p1 terminal line length (characters/line) 16<p1<140

p2 number of print lines per page

p3 number of blank lines per page (sum of p2 and p3 equals page length)

p4 number of delay or idle characters following each carriage return/line feed; this is used for terminals that require a pause after a CR/LF.

p5 number of delay characters following each top-of-form; if non-zero, a form-feed (X'0C') character is also output before each page.

p6 backspace character; an ASCII backspace (control-H) is always input to backspace over, or erase, the last character that was input; however, the user may set the actual character echoed to his terminal. This accomodates terminals that cannot physically backspace, or that have a backspace character other than the ASCII backspace.

EXAMPLE (for standard Microdata (Adds) terminal)--

```
:TERM  79,23,1,3,1,21
```

This example performs the following functions:

Carriage width set to 79 characters

Number of print lines per page = 23

Blank lines after top-of-page = 1

Idle characters after carriage return = 3

Idle characters after top-of-form = 1

Character echoed to a "Control-H' input = 21 = X'15'

TIME

System time and date can be displayed on the terminal by typing the verb TIME. Time is printed in the form hh:mm:ss, as on a 24-hour clock.

WHO

WHO is a TCL-I verb which returns the line number and the account name to which you are logged on. The line number is computed by subtracting the PCB FID of line zero from your PCB FID and dividing by 32. The account name is obtained by looking your PCB FID up in the ACC file and returning attribute one. If not found "UNKNOWN" is returned as account name.

Section X

ENGLISH LANGUAGE

INTRODUCTION

ENGLISH is a user oriented language used for accessing files within the system. An input sentence is constructed and terminated by a carriage return (r) . This sentence directs the appropriate ENGLISH processor to perform some specific retrieval function. The language is limited natural English and the formats for an input sentence are both simple and very general. The ENGLISH processors, together with the use of dictionaries, permit inputs to be stated directly in the technical terminology natural to each application area. The ENGLISH language uses the lineal format natural to prose text, accepts any number of variable length words, and permits a limited freedom of word order and syntax. An extension of descriptive error messages are used to inform the user of illegal constructs.

The general form of an ENGLISH sentence is as follows:

VERB file-name 'item-list' selection-criteria output-specification

The item-list specifies those items eligible for consideration, the absence of an item-list implies all items. An item-list consists of specifically enumerated item-ids, each enclosed within single quotes, additionally constrained by relational operators and logic connectives. A selection-criteria is optional and further limits the items for output to those meeting the specified conditionals. An output-specification enumerates those attributes desired for output; absence of an output-specification implies all attributes as defined with an S or X code in the file dictionary.

EXAMPLE--

```
:SORT ACCOUNT '10000' WITH CURR-BALNC "19.75" NAME ADDRESS
CURR-BALNC (r)

PAGE 1                                          13.45  25 OCT 1974

ACCOUNT... NAME..............  ADDRESS.............  CURR-BALNC

22030      F E CABRON          101 BEGONIA                20.50
22070      A A ALTHOFF         119 BAY STREET             22.60
23090      W J HIRSCHFIELD     230 BEGONIA                20.45

END OF LIST
```

ENGLISH Input Rules

The following rules apply to the use of ENGLISH language input sentences.

1. The first word of any input sentence must be a verb defined in the Master Dictionary (M/DICT).

2. Exactly one file-name defined in the M/DICT must appear in each sentence.

3. A sentence is terminated by a carriage return. A sentence may be continued to another line by typing a shift-control-O

4. File-names may consist of any sequence of non-blank characters and must be unique within the M/DICT and within all file dictionaries.

5. Any number of attribute-names may be used in a sentence.

6. Attribute-names may consist of any sequence of non-blank characters and must be unique within their associated dictionary and the Master Dictionary (M/DICT).

7. Any number of modifiers, connectives, and relational operators may be used which have been pre-defined in the Master Dictionary (M/DICT).

8. Verbs, file-names, attribute names, modifiers, connectives and relational operators must be immediately followed by a blank or language delimiter (i.e., single quote, double quote or carriage return).

9. Specified item-ids are enclosed within single quotes (e.g. 'XYZ') and may appear anywhere within the sentence.

10. Specified values are enclosed within double quotes (e.g. "ABC") and are attributed to the previous attribute-name.

ENGLISH Verbs

Verbs are action oriented words which evoke a specific processor. One and only one verb must begin each ENGLISH sentence. Verbs are defined in the Master Dictionary (M/DICT) as described in the chapter on Terminal Control Language (TCL). A set of verbs are provided, but the user may define any number of additional synonyms by copying the verb definition into a M/DICT item with the desired synonym name as the item-id.

LIST and SORT Verbs

The verbs LIST and SORT are used to generate a formatted output. The only difference is the LIST verb orders the output in the sequence specified and SORT orders the output in some specified sorted order. SORT always sequences on the item-ids and optionally by any number of specified attributes using the modifier BY. LIST will sequence on the order in which the item-ids have been specified in the input sentence. If no item-ids have been specified in an input sentence, all item-ids are implied and LIST will present these items in the hash sequence in which they are stored on the file. Generated output will be formatted into a columnar output if possible taking into account the maximum defined size of the specified attributes and their associated names along with the width of the terminal platen as defined by the TCL verb TERM. If more attributes have been specified than will fit across the page, a non-columnar output will be generated with the attribute names down the side and the associated attribute values to the right.

EXAMPLE--

```
:LIST ACCOUNT NAME ADDRESS (r)

PAGE 1                                          14:31  25 OCT 1974

ACCOUNT... NAME..................... ADDRESS.............

11080      E M AWAD                 107 BAY STREET
23070      L R MARCHANT             219 COVE STREET
23095      W E ZUMSTEIN             224 BEGONIA
35060      J A SCHWARTA             331 DOCK WAY
35085      J F SITAR                301 DOCK WAY
11025      R S MARCUS               107 BEGONIA
11105      C C GREEN                112 AVACADO
23000      H T LEE                  200 BAY STREET
23025      D C BINGAMAN             230 BAY STREET
35015      W F GRUNBAUM             318 COVE
11050      J R MARSHECK             125 R
11075      T F LINDSEY              11
23040      P B SCIPMA
23065      J WOSK
23110      J L VANGOTHEN
35030      F M HUGO
           J W ROMEY
           R
```

EXAMPLE--

```
:SORT ACCOUNT NAME ADDRESS (r)

PAGE 1                                          14:31  25 OCT 1974

ACCOUNT... NAME..............  ADDRESS.............

11000      M H KEENER          100 ANCHOR PL
11015      L K HARMAN          118 ANCHOR PL
11020      J T O'BRIEN         124 ANCHOR PL
11025      P R BAGLEY          130 ANCHOR PL
11030      F E CABRON          101 BEGONIA
11035      R S MARCUS          107 BEGONIA
11040      E G MCCARTHY        113 BEGONIA
11045      F AZ-DRESCH         119 BEGONIA
11050      J R MARSHECK        125 BEGONIA
11055      W H KOONS           131 BEGONIA
11060      F T NATORI          131 BAY STREET
11065      C V RANDALL         125 BAY STREET
11070      A A ALTHOFF         119 BAY STREET
11075      T F LINDSEY         113 BAY STREET
11080      E M AWAD            107 BAY STREET
11085      A B SEGUR           101 BAY ST
11090      J W JENKINS         130
11095      J B STEINER
11100      E F CHALMERS
  105      C C GREEN
    0      D L WEISBRO
           D R MAS
```

The SORT verb interacts with both the Function Correlative processor and the Conversion processor. Before the sort-key is built, the "F", "G" or "T" correlative functions will be performed, and the "T" and "U" conversions. Note that the "M" and "D" conversions are not performed, as they do not affect the results of sorting, and not performing those conversions will save processing time.

Multiple ascending and descending sort-keys may be intermixed at will. If the descending sort is required on the item-id alone, a descending sort-key with an attribute synonymous to the item-id must be specified.

An attribute with an AMC of 9999 will cause the size of the item (as defined in the count-field of the item) to be retrieved and used as the value referenced. Previously such an attribute with an AMC of 9999 may also be used as a selection criterion, thus allowing one to LIST or SORT items conditionally on their size. For example, the attribute MODE-SIZE is defined in the dictionary of SYSTEM-MODES as below

EXAMPLE--

```
MODE-SIZE
001 A
002 9999
003
004
005
006
007 MDO,
008
009 R
010 9
```

An example of a SORT statement using MODE-SIZE is:

```
SORT SYSTEM-MODES MODE-SIZE WITH MODE-SIZE<"7000"
```

COUNT Verb

The COUNT verb counts the number of items meeting the conditions as specified by the combination of item-list and selection criteria. The output generated by this verb is simply the number of items counted.

Up to $2^{31}-1$ (2147483647) items can be counted.

EXAMPLE--

```
:COUNT ACCOUNT (r)

67 ITEMS COUNTED.

:COUNT ACCOUNT WITH BILL-RATE ".30" (r)

23 ITEMS COUNTED.

:COUNT ACCOUNT >'10000' WITH CURR-BALNC AND WITH BILL-
RATE ".30" (r)

6 ITEMS COUNTED
```

SUM and STAT Verbs

The SUM and STAT verbs provide a facility for summing one specified attribute. In addition to the sum, the STAT verb provides a count and average for the specified attribute. If no attribute is specified then STAT and SUM operate on the entire item, providing a facility for summing the number of bytes in an item or set of items. The output generated by these verbs are the derived statistics.

```
:SUM ACCOUNT CURR-BALNC (r)

TOTAL OF CURR-BALNC IS : 169.40

:STAT ACCOUNT CURR-BALNC (r)

STATISTICS OF CURR-BALNC :
TOTAL = 169.40 AVERAGE = 15.400 COUNT = 11
```

SELECT Verb

The SELECT verb provides a facility to select a set of items using the item-list and the selection-criteria. These selected items are available one at a time to TCL-II processors. The output from the SELECT verb is a message signaling the number of items extracted or selected. The user then responds by typing in a single TCL-II sentence. In the processing of the sentence, all items will be processed from the previously selected item-list.

The following paragraph describes the use of SELECT with the TCL-II verbs B/ADD and B/DEL. This use differs from the use of SELECT with all other TCL-II verbs, described in Section IV.

When using the BATCH Processor after a SELECT, the item-id currently being processed is available to the BATCH-string. This must be specified in the (only) input line to BATCH after a SELECT; which must now contain at least one item-id substitution code, an asterisk (*). More than one asterisk is permitted; each will be replaced by the item-id currently being processed.

EXAMPLE 1--

```
:SELECT ACCOUNT WITH SEWER-ASMT (r)

10 ITEMS EXTRACTED.
:B/ADD M/DICT UPDATE-ACCOUNT

>* 2.00
'23070' UPDATED
'35025' UPDATED
'23065' UPDATED
'23080' UPDATED
'35000' UPDATED
'35025' UPDATED
'23075' UPDATED
'35020' UPDATED
'23060' UPDATED
'35005' UPDATED
```

THIS PAGE INTENTIONALLY LEFT BLANK

EXAMPLE 2--

```
:SELECT DICT MD WITH D/CODE "D" (r)

12 ITEMS SELECTED.
:B/ADD DICT BS JUNK (r)
>* DATA * ABCDEF (r)

'TSYM' UPDATED.
ETC.
```

ERROR MESSAGE 282 :DATA INPUT LIKE TO BATCH AFTER A SELECT MUST CONTAIN AT LEAST ONE ITEM-ID SUBSTITUTION CODE (ASTERISK *). WILL BE RETURNED IF NO ASTERISKS ARE FOUND.

SSELECT Verb

The SSELECT verb combines the ENGLISH SORTing capability with the SELECTion capability. SSELECT allows the same selection criteria as SELECT and allows the selected items to be sorted, just like with SORT.

File-Name Specification

Each ENGLISH sentence must contain one and only one file-name. The file-name defines the primary file on which the sentence operates and must be appropriately defined in the Master Dictionary (M/DICT). The modifier "DICT" may be included anywhere in the sentence (normally just preceding the file-name) to specify operation on the file dictionary rather than the file.

EXAMPLE--

```
:COUNT DICT ACCOUTNT (r)

79 ITEMS COUNTED.
```

Item List

An item list defines those items desired for processing. Absence of an item-list implies all items on the file. An item-list consists of any number of specified item-ids surrounded by single quotes (e.g. 'XYZ'). These item-ids may be interspersed at will throughout the ENGLISH sentence. Complex item-lists may be constructed using relational operators and logical connectives. For example, the item-list:

```
'ABC' OR > = 'DEF' AND < 'GHI'
```

selects item 'ABC' as well as all items greater than or equal to 'DEF' and also less than 'GHI'. The hierarchy of the logical connectives in an item-list is left to right. For example, the following item-list shows the left to right hierarchy, using explicit parentheses to show the implicit grouping.

```
(((<'A') OR > 'B' AND  <'C') OR >'D' AND <'E')
```

The OR connective is always implied and may be left out. Therefore, the list: <'A' > 'B' and <'C' > 'D' AND <'E' is equivalent to the preceding one.

The left to right hierarchy is shown in the following table:

Result / Connection	True	False
AND	Continue Comparison	Stop comparison; list false
OR	Stop comparison; list true	Continue comparison

```
:SORT ONLY EACH ACCOUNT>='11000' AND<='11020' OR >='11040'
AND <='11050' (r)

PAGE 1                                        21:09  25 OCT 1973

ACCOUNT...

11000
11015
11020
11040
11045
11050

END OF LIST
```

```
:SORT ONLY EACH ACCOUNT >='11040' AND <='11050' OR >='11000'
AND <='11020' (r)

PAGE 1                                          21:10  25 OCT 1973

ACCOUNT...

11040
11045
11050

END OF LIST
```

Selection Criteria

The selection-criteria specifies a set of conditions which must be met by an item before it is eligible for output. The selection-criteria is made up of one or more selection-criterion. Each selection-criterion must begin with the connective WITH followed by a single attribute-name. The attribute-name may be followed by a value-list. The rules for value-lists and the usage of relational operators is identical to that for item-lists except that double quotes surround the actual values. For example, the following selection-criterion is met by those items which have at least one value for the attribute DESCRIPTION which is either equal to "ABC" or is both greater than "DEF" and less than "GHI".

```
WITH DESCRIPTION "ABC" OR > "DEF" AND < "GHI"
```

If a selection-criterion has no value-list then it is true for those items which have at least one value for the specified attribute-name. The selection-criterion may be further modified by using either or both of the modifiers EVERY or NO immediately following the WITH. The modifier EVERY requires that every value for the attribute meet the specified conditional while the modifier NO reverses the sense of the entire conditional.

```
:LIST ACCOUNT TRNS-DATE TRNS-CODE WITH EVERY TRNS-DATE
BEFORE "3/18/72" (r)

PAGE 1                                          18:22  25 OCT 1973

ACCOUNT... TRNS-DATE.. TRNS-CODE

11075      17 MAR 1972          B
           17 MAR 1972          T
           17 MAR 1972          P
           13 MAR 1972          R
           15 JAN 1972          B
           14 JAN 1972          T
           10 JAN 1972          R

END OF LIST
```

```
:COUNT ACCOUNT WITH EVERY TRNS-CODE NOT "P" (r)

7 ITEMS COUNTED.
```

Selection-criterion may be used at any point within an ENGLISH sentence; however they are all logically grouped together (in fact the ENGLISH pre-processor reorganizes the input sentence, grouping the selection-criterion together). This logically grouped set of selection-criterion constitute the selection-criteria. A selection-criteria may consist of up to nine AND clauses. An AND clause is made up of any number of selection-criterion bound by AND connectives. The AND clause is terminated when an OR connective is found in the left to right scan (note: the absence of an AND connective implies an OR connective). For an item to pass the selection-criteria the conditions specified by any one of the AND clauses must be met. An example of the logical hierarchy of AND clauses is shown below, the parentheses have been included for clarity but do not appear in an actual ENGLISH input sentence.

```
(WITH DESC "ABC" AND WITH VALUE "1000") OR (WITH DESC "ABC"
 AND WITH NO VALUE)
```

```
:LIST ACCOUNT AVG-USAGE SEWER-ASMT BILL-RATE_ (r)
:WITH AVG-USAGE "20" OR "25" AND WITH SEWER-ASMT "150"_ (r)
:OR WITH AVG-USAGE "20" OR "25" AND WITH BILL-RATE ".30" (r)

PAGE 1                                          17:36  25 OCT 1973

ACCOUNT... AVG-USAGE SEWER-ASMT BILL-RATE

11050            20                  0.30
23080            20        150       0.35
11020            20                  0.30
11085            25                  0.30

END OF LIST
```

```
:COUNT ACCOUNT WITH CURR-BALNC AND WITH BILL-RATE >".25" AND <".45"_
:AND WITH DEPOSIT AND WITH AVG-USAGE > "10" (r)

11 ITEMS COUNTED.
```

An ASCII up-arrow or circumflex (shift N) may be used as an ignore character in any value or item-id. All comparisons made against the file ignore the characters in the corresponding position.

EXAMPLE --

```
SORT ACCOUNT NAME ADDRESS WITH ADDRESS "  BEGONIA" SORT-ON
ADDRESS (r)

PAGE 1                                          16:43  25 OCT 1973

ACCOUNT... NAME.............. ADDRESS.............

11030      F E CABRON          101 BEGONIA
11035      R S MARCUS          107 BEGONIA
11040      E G MCCARTHY        113 BEGONIA
11050      J R MARSHECK        125 BEGONIA
11055      W H KOONS           131 BEGONIA
23115      T F PIATKOSKI       200 BEGONIA
23110      J L VANGOTHEN       206 BEGONIA
23105      B G PAUL            212 BEGONIA
23100      G J PACE            218 BEGONIA
23095      W E ZUMSTEIN        224 BEGONIA
23090      W J HIRSCHFIELD     230 BEGONIA
```

Comparison are performed in the following manner:

If the fields are specified as left-justified, the comparison will be alpha. For example, sorted items:

```
09
3
31
A
AA
Z
```

If the fields are specified as right-justified and the items are all numeric, it will be a true numeric compare.

If the fields are right-justified and the items are mixed, the comparison will be alpha-numeric. For example:

```
   9
  ZZ
 AAA
0000
1234
QWIZ
```

Output Specification

All attribute-names not a part of a selection-criterion (i.e. preceded by the modifier WITH) or not modified by certain control modifiers (e.g. EVERY, BY) are part of the output-specification. Attribute-values from

those items passing both the item-list and the selection-criteria will be displayed in an automatically generated system format. This format will include a heading line displaying the date, time and page number (unless suppressed) at the beginning of each new page. The page size is set through the use of the TERM verb (see System Commands). The LIST processor will attempt to format the output into a columnar format with the attribute name as a column heading. This column-format is attempted using as a column width for each specified attribute either the attribute max-size from the dictionary or the attribute-name whichever is larger. If the sum of the column widths, adding one blank separator for each specified attribute-name, does not exceed the page width as set by the TERM verb; a columnar-format will be generated. In a columnar-format the attribute-names, as specified, are displayed in a single line header across the top of the page. The values for each of the items are then displayed in their respective columns. The attribute-name header is repeated at the top of each new page. If the requested output exceeds the page width, the attribute-names are listed down the side of the output with their respective values immediately to the right. A significant difference between the formats is that for the columnar-format all headings are listed only once for each page whether or not values exist for the columns; while in the non-columnar-format headings are displayed over for each item only if there are values for the associated attributes.

```
:LIST ACCOUNT '11000' '11015' NAME ADDRESS START-DATE (r)

PAGE 1                                              15:43  25 OCT 1973

ACCOUNT... NAME.............. ADDRESS.............. START-DATE..

11000      M H KEENER         100 ANCHOR PL         25 OCT 1971
11015      L K HARMAN         118 ANCHOR PL         01 JAN 1968

END OF LIST

:TERM 60 (r)

:LIST ACCOUNT '11000' '11015' NAME ADDRESS START-DATE (r)

PAGE 1                                              15:43  25 OCT 1973

ACCOUNT : 11000
NAME    M H KEENER
ADDRESS    100 ANCHOR PL
START-DATE    25 OCT 1971

ACCOUNT : 11015
NAME    L K HARMAN
ADDRESS    118 ANCHOR PL
START-DATE    01 JAN 1968
```

If no attribute-names are specified all are assumed and are generated by successively retrieving the attributes 1,2,3 ... from the file dictionary until no more can be found. These special attributes must have either an S or X for their dictionary code. These special synonyms have a special format (see below) specifying the heading-name to be used for output.

```
:LIST ACCOUNT '11000' (r)

PAGE 1                                          15:35  25 OCT 1973

ACCOUNT : 11000
NEXT-ACCNT    11010
CSTMR-NAME    M H KEENER
SERVC-ADDR    100 ANCHOR PL
MAIL-CITY.    THE CITY
MAIL-STATE    CA
ACCNT-STAT    A
DEPOSIT-=.    10.00
START-DATE    25 OCT 1971
BILL-RATE.    0.30
TRASH-CHGS    2.00
AVG-USAGE.    27

END OF LIST
```

The item-id as specified by file defining item in either the file dictionary or the M/DICT is always included in output-specification unless the modifier ID-SUPP is used. If an output is to be restricted to only the item-ids the modifier ONLY must precede the file-name to inhibit the appending of the special synonyms.

The following table summarizes the various dictionary attributes as they apply to the formatting of an output specification.

Name	A/AMC	Value	Meaning
D/CODE	1	A	attribute defining item
		S	special synonym
		X	special synonym to maintain order, but ignore for output
A/AMC	2	attribute-no	for A-code attributes, defines attribute number.
S/NAME	3	text-name	for S-code attributes, use this name for heading (note: these names may be padded with blanks to align non-columnar output)

Name	A/ANC	Value	Meaning
S/AMC	4	attribute-no	for S-code attributes, defines attribute number.
V/TYPE	9	L	for columnar-output only, left justify output in column, value size greater than column width, value is folded
		R	for columnar-output only, right justify output in column. If value size greater than column width value overlay previous columns.
V/MAX	10	n	for columnar-output only, number of characters to reserve for the column width. Column width will be increased if attribute-name is larger than V/MAX
		Ln	for non-columnar output, left justify output and reserve characters for each repetition of a multi value. Multi-values will fold at end of line and repeat aligned with the start of the first value.
		Rn	for non-columnar output, as for L but right justify in the reserved value area

Modifiers, Relational Operators and Connectives

Modifiers, connectives and relational operators may be used to further modify the meaning of an ENGLISH sentence or to add naturalness. These special words are defined as items in the Master Dictionary (M/DICT) and to that extent are reserved words. However, (with the exception of DICT) a user may define any number of synonyms for these words and even remove the system defined entries thereby creating his own semantics for the language.

MODIFIERS

BREAK-ON. Defines a control break as any change in value for the immediately following attribute. Up to four control breaks are permitted and the left-most defined control break is the highest level. On the occurrence of a specified control break the attribute causing the break displays asterisks ** in its corresponding column. Also the current sub-totals (see TOTAL) are printed and the totals are rolled to the next level.

```
:SORT ACCOUNT WITH CURR-BALNC TOTAL CURR-BALNC TOTAL DEPOSIT- (r)
:BREAK-ON BILL-RATE BY BILL-RATE (r)

PAGE 1                                          16:33  25 OCT 1973

ACCOUNT... BILL-RATE CURR-BALNC DEPOSIT.

11015          0.30       8.60    10.00
11030          0.30      20.50    10.00
11075          0.30      13.10    10.00
11115          0.30       9.20    10.00
23030          0.30      11.80    10.00

               ****      63.20    50.00

11070          0.35      22.60    10.00
23025          0.35      18.70    10.00
23090          0.35      20.45    10.00
35095          0.35      19.25    10.00

               ****      81.00    40.00

11100          0.40      17.50    10.00
35075          0.40       7.70    10.00

               ****      25.20    20.00

****                     169.40   110.00
```

PAGE. Halts list at end of each page when output is to terminal. Listing is resumed when (r) is entered.

DBL-SPC. Sets double spacing for output list.

DICT. Modifies the file-name so that the ENGLISH sentence references the file dictionary instead of the file.

EVERY, EACH. Modifies a selection-criterion so that every value for the attribute must meet the specified conditional for the selection criterion to be true. This modifier follows the modifier WITH.

HDR-SUPP, SUPP. Suppresses the automatic time and date heading at the top of each new page. Also suppresses the end-of-list message.

```
:LIST ACCOUNT '11015''11020''11074' HDR-SUPP NAME ADDRESS (r)

ACCOUNT... NAME..............  ADDRESS.............

11015      L K HARMAN          118 ANCHOR PL
11020      F E CABRON          101 BEGONIA
11075      T F LINDSEY         113 BAY STREET
```

COL-HDR-SUPP. Suppresses column headings, as well as the time and date headings and the "END OF LIST" message.

ID-SUPP. Suppresses the file-name and the item-id on LIST or SORT statements.

LPTR, (P). Routes the output to the line-printer attached to the system.

ONLY. Used preceding the file-name, inhibits the appending of the special synonyms when a null output-specification is encountered. Used in the same sense as EACH, except that ONLY may not follow WITH.

BY. Designates the following attribute-name as a sort-key. The item-id is always a sort-key and additional sort-keys may be specified using this modifier. The left most specified sort-key is the most significant. Sequencing is in ascending order comparing the ASCII value of the characters left to right. Sort keys are generated in such fashion that numeric portions of the key are logically padded with leading zeros so that numeric fields sort correctly.

BY-DSND. Same as BY, but sequencing is done in descending order.

```
:SORT ACCOUNT CURR-BALNC AVG-USAGE BILL-RATE SEWER-ASMT O(c) (r)
:BY CURR-BALNC BY AVG-USAGE BY BILL-RATE O(c) (r)
:SEWER-ASMT ITEM-ID (r)

PAGE 1                                          16:52  25 OCT 1973

ACCOUNT... CURR-BALNC AVG-USAGE BILL-RATE SEWER-ASMT

35000                 13        0.35          150
35040                 16        0.35
35005                 17        0.35          150
23000                 19        0.35
23055                 19        0.35
11020                 20        0.30
11050                 20        0.30
23080                 20        0.35          150
11060                 21        0.30
9999                  21        0.30
35090                 21        0.35
11040                 22        0.30
35070                 22        0.35
23060                 23        0.35          150
23065                 23        0.35          150
23115                 24        0.35
35100                 24        0.35
35025                 24
11085                 25
23040
23050
```

TOTAL. Causes totals for the following attribute to be accumulated. Up to four levels of totals will be stored. These totals are printed and rolled forward in conjunction with a control break (see BREAK-ON) at the associated level.

WITH, IF. Designates a selection-criterion. The following attribute-name and associated value-list constitute a conditional limitation which must be satisfied for the selection-criterion to be met.

RELATIONAL OPERATORS

Relational Operators; are used to conditionally limit the attribute-values and item-ids which they immediately precede.

=. EQ, null limits to an equal relationship (e.g. DESC = "ABC" implies a value of "ABC" must be found for the attribute DESC for the specified relation to be satisfied). The absence of any relational operator implies an equality.

>, GT, AFTER the retrieved datum must be greater than the specified datum.

<, LT, BEFORE the retrieved datum must be less than the specified datum.

>=, GE The retrieved datum must be greater than or equal to the specified datum.

<=, LE The retrieved datum must be less than or equal to the specified datum.

#, NE, NOT, NO (e.g. LIST ACCOUNT WITH CURR-BALNC NOT "10.00") Or may be used to select items with a null attribute value. (e.g. LIST ACCOUNT WITH NO SEWER-ASMT). The retrieved datum must be not equal to the specified datum.

CONNECTIVES

AND. Logical connective specifying that both the connected parts must be true. This connective may bind selection-criterion into AND clauses, and logically AND item and value-lists. In all cases where AND is not specified, OR is assumed.

A, AN, ARE, ANY, FILE, FOR, IN, ITEMS, OF, OR, THE. These connectives are ignored by the ENGLISH preprocessor (throwaway). They are included to provide a degree of naturalness.

HEADING. A heading can be specified in the ENGLISH LIST or SORT statement, which will be printed at the top of every page. The normal page number, time and date, and column heading will be suppressed, as will the END OF LIST message. The heading is entered, enclosed in double-quote signs, immediately following the 'HEADING' connective (D/CODE=CL); it may appear anywhere in the LIST or SORT statement.

All characters enclosed in the double quote signs are literal, except the following:

^ or ↑	: specifies insertion of current page number
]	: specifies start of a new line (carriage-return/line feed insertion)
\	: specifies insertion of system time and date in the standard format.

The HEADING connective and its associated heading data are recognized in the ENGLISH pre-processor (MD3), and the formatted heading stored in the history string; the text is preceded by an "H"; the heading text follows as it is, except for the following conversions: to AM, to VM; to SVM. The text is terminated by a SM and a "Z"; HSEND points to the SM. If the heading text enclosed in double-quotes does not follow, error message 7 (HEADING TEXT MUST FOLLOW THE "HEADING" CONNECTIVE) will be returned.

Section XI

CONVERSION

INTRODUCTION

Conversions may be defined for attributes which will apply special conversion processing on the associated attribute-value just prior to outputting. The same conversions are also applied to values on input. The purpose of the conversion facility is to provide a means whereby attribute-values may be stored in some un-converted form on the file but be referenced and displayed in a converted form. Conversions are stored as values for the V/CONV attribute (AMC = 7) in the file-dictionary. Conversion processing is invoked automatically by referencing an attribute-name with defined conversion specifications. Multiple conversions may be defined for the same attribute by storing multiple-values for V/CONV. Values in selection-criterion will be converted by the pre-processor prior to comparison, therefore it is important to remember that all comparisons are made with converted values.

D Conversion (Date)

The date conversion allows a date to be input in a variety of formats and will convert the date to a compact internal format suitable for arithmetic processing. The formats for the date conversion are:

D

Dn

Dn*m

- D - Date conversion code
- n - Number of digits to print in year for output; if null, 4 is assumed (e.g., 1974)
- * - Concatenated segment specification
- m - Number of concatenated segments to skip prior to performing conversion (for both input and output).

The date conversion processor will accept a variety of input formats including the following:

6/26/72	MM/DD/YY
6/26/1972	MM/DD/YYYY
26 June 72	DD MM.. YY

These dates will be converted to any input format which is a signed integer equal to the number of days plus or minus from December 31, 1967. Using this technique the date 6/26/72 would be stored internally as the integer 1639. The following table shows a list of converted dates and their associated internal formats.

DATE.....	RAW-DATE....
14 AUG 1940	-10000
05 APR 1965	-1000
22 SEP 1967	-100
21 DEC 1967	-10
30 DEC 1967	-1
31 DEC 1967	0
01 JAN 1968	1
10 JAN 1968	10
09 APR 1968	100
26 SEP 1970	1000
18 MAY 1995	10000

On output the date is always converted to the form "DD MM YYYY" as shown in the preceding list. The year may be reduced in size or eliminated altogether through the use of the "n" option following the D-code.

MD Conversion (Mask Decimal)

The MD conversion provides a facility for converting and scaling numbers with decimal points, commas and dollar signs to or from an internal format of strictly a signed integer. The format for the MD conversion is:

MDnm,$

MD - Mask decimal conversion code

n - Single numeric digit defining the number of digits to print following the decimal point. If n = 0, the decimal point will not be ouput following the value.

m - Single numeric digit defining the number of implied decimal digits for the integer on the file. If this parameter is omitted, m = n is assumed.

, Optional parameter for output which causes commas to be inserted appropriately.

$ Optional parameter for output which causes a dollar sign to be appended preceding the converted output value.

Examples:

V/CONV	File-value	Output-value
MD2	1234	12.34
MD23	1234	1.23
MD32	1234	12.340
MD2,	-123456	-1,234.56
MD23,$	1234567	$1,234.57

MT Conversion (Time)

The MT conversion provides a facility for converting a time shown as hour:minute to or from internal format. The internal format is the number of seconds from midnight (24:00). The external format uses the 24 hour military format. The format for the MT conversion is:

MT

MX Conversion (Hexadecimal)

The MX conversion will convert any string of characters stored on the file to or from its corresponding hexadecimal equivalent. One byte on the file will convert to two hexadecimal digits. The format for the MX conversion is:

MX

T Conversion (Translate)

The T conversion provides a facility for converting a value by translating through a file. The value to be translated is used as an item-id for retrieving an item from the defined translation file. The input value is then converted by replacing it with a defined attribute-value from the translation item. The format for the T conversion is:

T file name; c; input-amc; output-amc

T - translate conversion code

file name - The file-name through which the translation takes place. It may be preceded by the single character "*" to indicate a dictionary.

; - separator

c - translate Sub-code, must be one of the following:

V - verify; conversion item must exist on file, and specified attribute must have value for conversion.

C - convert; if conversion item does not exist, or if specified attribute has no value use original value; otherwise perform conversion.

I - input verify only; functions like a V for input and a C for output.

O - output verify only; functions like a V for output and a C for input

X - convert; if conversion item does not exist, or if specified attribute has no value, use null value; otherwise perform conversion.

input-amc - attribute mark count for input translation. After locating the translation item using the input value as the item-id the attribute-value for the defined amc, if any, will replace (convert) the original value. If this parameter is null to input translation takes place.

output-amc - attribute mark count for output translation. Functions similarly to input-amc but is invoked for output translation. If this parameter is null no output translation takes place.

Example:

Conversion Code: T sample-file; C;1;1

Sample-File items: Programmer
100

100
Programmer

Unconverted Value	Converted Value
Programmer Engineer 100	100 Engineer Programmer

U Conversion (User)

User conversion permits a user defined special purpose subroutine to be evoked for special converson. The format is:

Unxxx

U - user conversion code

nxxx - Mode-ID (refer to Section XVI Operand Conventions)

At the point where conversion normally occurs for both input and output the user program is entered with the value to be converted in a work area. For the exact nature of the programming interface please consult the programming documentation.

Section XII

CORRELATIVES

INTRODUCTION

Correlatives may be used to define special processing interrelationships (correlations) to be applied to attribute values. Correlatives are stored as values for the V/CORR attribute (AMC=8) in the file-dictionary items. Correlative processing is invoked automatically when referencing an attribute-name with a defined correlative. Correlative processing for output occurs as the values are retrieved from the file, and prior to being output or used in a selection-criterion. Multiple correlatives may be defined for the same attribute, with each defined correlative being a multi-value.

D Correlative (Associative)

The D correlative is used to identify primary and secondary associative attributes within the same item. There are two correlatives involved; D1 and D2. The form of the correlatives are:

D1;amc1;amc2;amc3;...

- D1 — Correlative code identifying a primary associative attribute
- ; — Separator
- amc — numeric attribute mark count of each of the defined secondary association attributes in the file. Each amc in the correlative must be numerically greater than the primary D1 amc.

D2;amc

- D2 — Correlative code identifying a secondary association attribute
- ; — separator
- amc — numeric attribute mark count of the defined primary associative attribute in the file.

The purpose of the D correlative is to provide a facility whereby a set of attributes, the secondary D2's, can be logically grouped with a single master attribute, the primary D1. This type of relationship is useful in describing, for example, a list of purchase order numbers in a part-file

where the purchase order number is the D1 and the set of related attributes-values, like quantity-on-order, quantity-received, etc. are D2's, and each relates back to and is grouped with, the primary D1 value.

The D1 attribute may have multi-values each separated by a value-mark. Each D2 attribute should have a corresponding number of multi-values, however each of these multi-values may be multi-valued themselves; each sub multi-value is separated by a secondary-value-mark (X'FC'). In the following example both a columnar and non-columnar output is shown for a D1 attribute and three associated D2 attributes. The attribute defining dictionary items are also listed.

```
:LIST ACCOUNT '11080' DATE CODE UNITS DOLLARS (r)

PAGE 1                                          15:24  29 OCT 1973

ACCOUNT... DATE          CODE          UNITS          DOLLARS
                         *             *              *

11080      07 APR 1972   P                                  9.50
           18 MAR 1972   B                                  9.50
           17 MAR 1972   T                                  2.00
           13 MAR 1972   R             2721                 7.50
           05 FEB 1972   P                                  9.20
           15 JAN 1972   B                                  9.20
           14 JAN 1972   T                                  2.00
           10 JAN 1972   R             2696                 7.20

END OF LIST

:LIST ACCOUNT '11080' DATE CODE UNITS DOLLARS (r)

PAGE 1                                          15:24  29 OCT 1973

ACCOUNT : 11080
DATE        07 APR 1972   18 MAR 1972   17 MAR 1972
            13 MAR 1972   05 FEB 1972   15 JAN 1972
            14 JAN 1972   10 JAN 1972
  CODE                P             B             T
                      R             P             B
                      T             R
  UNITS
                   2721
                                 2696
  DOLLARS          9.50          9.50          2.00
                   7.50          9.20          9.20
                   2.00          7.20

END OF LIST
```

:LIST DICT ACCOUNT 'DATE' 'CODE' 'UNITS' 'DOLLARS' ⓡ

PAGE 1 15:27 29 OCT 1973

ACCOUNT...	D/CODE.	A/AMC	S/NAME...	S/AMC	V/CONV...	V/CORR.....	V/TYP	V/MAX
DATE	S	20	DATE	20	D	D1;21;22;23	R	R11
CODE	S	21	CODE	21		D2;20	R	R11
UNITS	S	22	UNITS	22		D2;20	R	R11
DOLLARS	S	23	DOLLARS	23	MD2	D2;20	R	R11

END OF LIST

Function correlatives defined for attributes which also have D1, D2 correlatives ignore the D1, D2 correlative. A print-limiter on the D1 attribute causes all corresponding D2 values to be suppressed.

F Correlative (Function)

The F correlative is used to compute a value as a mathematical function on a defined set of attributes within one item. All arithmetic operations operate on the last two entries in a push-down stack. This push-down stack has a maximum capacity of three entries. A function correlative is comprised of any number of operands or operators separated by semi-colons. When the function processor encounters an operand specification (i.e., a numeric attribute-mark-count or constant) it "pushes" the corresponding value into the stack. When the function processor encounters an arithmetic operator it performs the corresponding operation on the last two entries in the stack. The general form of the F correlative is as follows:

F;e1;e2;e3;...

F correlative code

; separator

e element codes

a numeric amc specifying the value from the attribute specified.

If a single valued attribute is to be repetitively added, subtracted, etc., with a multivalued attribute the single letter R should follow the amc entry in the F correlative.

Cn where n is a numeric constant to be used in the computation.

D uses today's date in computation.

T uses current time in computation.

an operator as below:

*	multiplication of the last two entries in the stack.
/	division of the last entry in the stack by the second-last entry.
+	addition of the last two entries in the stack.
-	subtraction of the second-last entry in the stack from the last entry.
S	specifies that a Sum is needed of all previous computations; therefore only one value is returned if Sum is specified. The S operator can only occur as the last entry in the F-correlative.
=	equal
<	less than
>	greater than
#	not equal
[	equal to or greater than
]	equal to or less than

Attribute operands may be multi-valued. When arithmetic operations are performed on two multi-values lists (vectors) the answer will also be multi-valued and will have as many values as the longer of the two lists. Zeros will be substituted for the null values in the shorter list. In the following example this concept is demonstrated.

Stack 1	5	10	15	(no value)
+				
Stack 2	20	30	40	50
=				
Stack 1	25	40	55	50

Stack 2 is compared to Stack 1; Stack 1 contains either a "1" or a "0" depending on the result. A "1" indicates a positive or yes result; a "0" indicates a negative or no result.

EXAMPLE --

F;C3;C3; = Stack 1 would contain a "1"

By following with a data and a multiply operator, the attribute can be conditionally set.

EXAMPLE --

F;C3;C3;=;C5,* Stack 1 now contains a "5"

FUNCTION PROCESSING

Element	Description	Action
amc	attribute	push corresponding attribute values into push-down stack maximum three levels.
		STACK 2 ⟶ STACK 3
		STACK 1 ⟶ STACK 2
		attribute values ⟶ STACK 1
Cn	constant	push numeric constant "n" into stack
		STACK 2 ⟶ STACK 3
		STACK 1 ⟶ STACK 2
		n ⟶ STACK 1
+	ADD	STACK 1 + STACK 2 ⟶ STACK 1
		STACK 3 ⟶ STACK 2
-	SUBTRACT	STACK 1 - STACK 2 ⟶ STACK 1
		STACK 3 ⟶ STACK 2
*	MULTIPLY	STACK 1 * STACK 2 ⟶ STACK 1
		STACK 3 ⟶ STACK 2
/	DIVIDE	STACK 1/STACK 2 ⟶ STACK 1
		STACK 3 ⟶ STACK 2
S	SUM	Σ_1 STACK 1 ⟶ STACK 1
		prior to this operation STACK 1 may be multi-valued, this operator sums all those multi-values into a single value.

Element	Description		Action
D	DATE		push numeric value representing current system date into stack.
			STACK 2 ⟶ STACK 3
			STACK 1 ⟶ STACK 2
			date ⟶ STACK 1
T	TIME		push numeric value representing current system time into stack.
			STACK 2 ⟶ STACK 3
			STACK 1 ⟶ STACK 2
			time ⟶ STACK 1
=	EQUAL	1)	if STACK 1 = STACK 2, 1 ⟶ STACK 1
			STACK 3 ⟶ STACK 2
		2)	if STACK 1 ≠ STACK 2, 0 ⟶ STACK 1
			STACK 3 ⟶ STACK 2
≠	NOT EQUAL	1)	if STACK 1 ≠ STACK 2, 1 ⟶ STACK 1
			STACK 3 ⟶ STACK 2
		2)	if STACK 1 = STACK 2, 0 ⟶ STACK 1
			STACK 3 ⟶ STACK 2
<	LESS THAN	1)	if STACK 1 < STACK 2, 1 ⟶ STACK 1
			STACK 3 ⟶ STACK 2
		2)	if STACK 1 NOT < STACK 2,0 ⟶ STACK 1
			STACK 3 ⟶ STACK 2
[	EQUAL TO OR GREATER THAN	1)	if STACK 1 [STACK 2, 1 ⟶ STACK 1
			STACK 3 ⟶ STACK 2
		2)	if STACK 1 NOT [STACK 2,0 ⟶ STACK 1
			STACK 3 ⟶ STACK 2

Element	Description	Action
>	GREATER THAN	1) if STACK 1 > STACK 2, 1 → STACK 1 STACK 3 → STACK 2 2) if STACK 1 NOT > STACK 2, 0 → STACK 1 STACK 3 → STACK 2
]	EQUAL TO OR LESS THAN	1) if STACK 1] STACK 2, 1 → STACK 1 STACK 3 → STACK 2 2) if STACK 1 NOT] STACK 2, 0 → STACK 1 STACK 3 → STACK 2

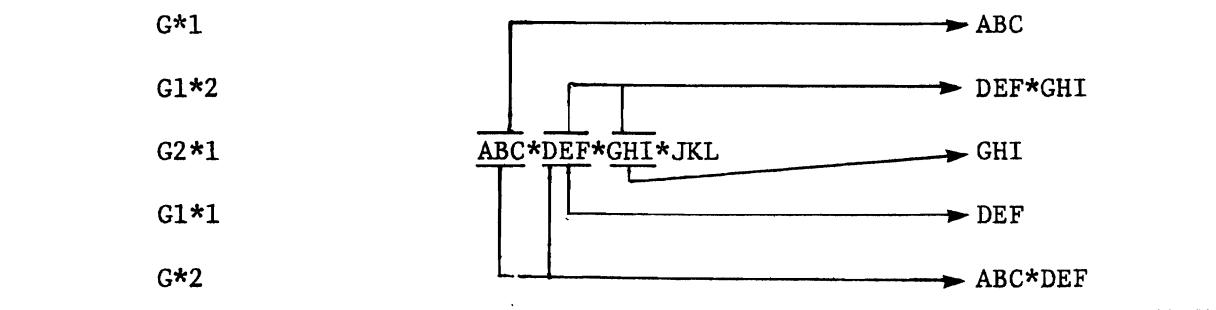
G Correlative (Group Retrieved)

The G correlative is used to select one or more contiguous segments of a concatenated attribute value from an item for retrieval. An attribute-value in an item may consist of any number of concatenated segments with each segment separated by an asterisk "*". The form of the correlative is:

G m * n

- G — correlative code
- m — number of the first selected segment. If omitted, one is assumed and retrieval begins with the first concatenated value segment.
- * — concatenated segment separator
- n — number of contiguous concatenated segments to be selected.

The following diagram shows an example of the use of the G correlative.



T Correlative (Text)

The T correlative is used to extract a fixed number of characters from a stored value. The form of the correlative is:

Tn

where n is the number of characters to extract. If the stored value is less than "n" characters long, only the actual number of characters stored will be retrieved. Extraction takes place beginning with the first character from left to right or right to left depending on whether type L or R is specified for V/TYPE.

Section XIII

SECURITY

INTRODUCTION

Security codes may be optionally stored in the L/RET and L/UPD attributes of a dictionary item to restrict access. Access may be denied at both the file and attribute level, additionally separate codes may be assigned for both retrieval and update. At LOGON, each user is assigned the set of security codes which are in his logon-item. During the session whenever an L/RET or L/UPD code is encountered, a search is made of the user-assigned codes for a match; if no match is found, the user is denied access. A security code may consist of any combination of legal ASCII characters.

L/RET and L/UPD

Both file-definition ("D" code) and attribute-definition ("A" "S" code) items have L/RET (retrieval lock) and L/UPD (update lock) defined as attributes. When these attributes have values stored, they are known as security codes. Although there is no prohibition against multiple values for these attributes, only the first attribute-value is used for matching against the user-assigned codes. Since each file and/or attribute may be individually locked for both update and retrieval, a user must be assigned multiple codes to that set of data he is allowed to access. Using this feature, a complex "mask" can be constructed for each user, giving each user a different sub-set of files and attributes which he may access.

Security at the file level is invoked at the processor level. The following processors are assumed to be updating processors and therefore require a match on the L/UPD attribute in the file-definition item:

AS

BATCH (files referenced by the BATCH-string)

COPY

EDIT

All of the other processors are assumed to be retrieval processors and require a match on the L/RET attribute in the file-definition item. Failure to match one of the user security codes with either the L/RET or L/UPD attribute value will generate the message:

```
210   FILE file-name IS ACCESS PROTECTED
```

and control will return to TCL.

Security at the attribute level is available only through ENGLISH. Requesting of an attribute without the matching security code causes an error message, reference to the attribute is deleted from the input sentence and processing continues on the remainder of the sentence.

User Assigned Codes

The logon-item (refer to LOGON) contains the list of security codes assigned for that particular user. These codes are values for the dictionary attributes L/RET and L/UPD in the particular logon-item. There is a one to one correspondence between the L/RET and L/UPD attribute values in the logon-item and in the particular attribute entry for which the codes are being verified. In other words an L/RET code in an attribute definition item must be verified against one of the values stored in the L/RET attribute in the logon-item.

Security codes may be assigned initially when an account is created using the CREATE-ACCOUNT Proc. Security codes may be added or deleted by updating the appropriate logon-item using the EDITOR (assuming one has the appropriate security codes); however updates to the logon-item should only be performed when no one else is logged onto the system.

Security Code Comparison

Security codes are verified comparing the value in the file dictionary vs. the corresponding string of values in the logon-item. An equal or verified compare occurs when the value in the file dictionary is exhausted and all characters match up to that point.

EXAMPLE--

file dictionary code	logon-item code	result
123	123	match
12	123	match
123	12	no match

When referencing a file using a Q synonym (refer to DICTIONARIES) a security code match is made at all levels (i.e., SYSTEM, M/DICT, file dictionary) and therefore a correspondence must be maintained at all levels in order to process the Q synonym files. Since the logon-item for the account containing the primary file is verified for security codes, the user referencing the Q synonym must have a code defined in this logon-item which will verify with the first code in the equated accounts logon-item. Therefore in a logon-item only the first code is used to protect the account from Q synonym accesses while all the codes in logon-item are assigned to the user when he logs on.

Section XIV

BATCH PROCESSOR

INTRODUCTION

The BATCH processor (BATCH) provides a facility for inputting, updating and deleting items or data within items. BATCH operates using a predefined "BATCH-string", and an input line, to update one or more items in multiple files simultaneously. The BATCH-string is stored as an item in a file and provides the dictionary function for the subsequent update. In other words, the BATCH process ignores the attribute defining items defined for the designated files and instead relies on the BATCH-string to define the updating algorithm.

Evoking BATCH

BATCH is evoked using the format:

```
B/ADD   file-name   item-id
  or
B/DEL
```

the file-name and item-id define the specified BATCH-string. B/ADD in general defines an updating function and may be used to delete items. B/DEL provides a reversing update function in that using B/DEL on the identical BATCH-string and input line will in general negate or reverse the effects of a previous B/ADD operation. After BATCH gains control it will prompt the terminal for input using the character ">" as the prompt. Each input line entered will be processed separately by BATCH and will generate a file update. BATCH will continue to prompt for more inputs until the user exits by entering a null line (a carriage-return immediately) following the BATCH prompt character).

Many users store BATCH strings in the Master Dictionary (M/DICT) but this is not required; in fact the recommended procedure is to define a separate file or files which are used exclusively to store BATCH-strings. Additionally, these files should be single-level (i.e., Dictionaries) to save an additional file access in retrieving the BATCH-string. Also, most common usage of the BATCH processor is from a PROC where the input lines to BATCH have been stored in the PROC's secondary output buffer (stack). For this reason one must examine both the BATCH-string and its associated PROC to fully comprehend the resulting processing.

The B/DEL verb allows the user to delete specific values from attributes in an item. In general, it should be used to delete specified values from one or more single or multi-valued attributes, or to delete implied values (not specified) in single-valued attributes. In addition, such implied values may also be accessed by the "secondary file" section(s) of the BATCH-string. Thus it is possible to delete a value from an attribute

in the primary file without knowing what the value on file is, and to use this same value as implied inputs to attribute updating elements in secondary files.

As a specific example, suppose that attribute five of file ACCOUNT contains a value that is to be zeroed at the end of a month, and that value is to be added in to attribute six of the file HISTORY. A representative BATCH-string would be as follows:

```
ACCOUNT,I
4N
A,Y31          (a)
Z
HISTORY,I1
5N
A2,J(5),Y32    (b)
```

If the value to be deleted in known, the input to this string would be:

```
>account-id   value (r)
```

which would subtract the value, using element (a), in the ACCOUNT file, and add in the same value, using element (b), in the HISTORY file. (Note reversal of the Y31/Y32 actions). In this case the "J(5)" sub-element in (b) is not used.

If, however, the value to be deleted is not known, or if the user does not wish to enter the value, and the input is:

```
>account-id (r)        (or)        >account-id\ (r)
```

element (a) causes the deletion of the value in attribute 5 of ACCOUNT, as before; and the "J(5)" sub-element is used to "link" back to the primary file (ACCOUNT) and to add in the value to the HISTORY file.

Note that the same effect could have been achieved, at significantly greater processing cost, by a suitable BATCH-string employing Translate conversions to pick up the values to be deleted. However, if the primary attribute is multi-valued and one of the values is deleted, the "J" sub-element must be used to delete the corresponding value in a secondary file.

If the primary item itself is deleted (using the ",X" option in the file-defining element), the "J" sub-elements can be used to add or delete values into secondary file attributes.

BATCH-string Format

The BATCH-string is an item in a file and is the definition of the updating algorithm to be performed. The BATCH-string consists of a set of elements, one element per attribute or line. In general, for each file-item accessed

there will be: a file-defining element, one attribute-defining element for each attribute in the defined file and an end-of-file element. This sequence is repeated for each file-item to be updated by the BATCH-string. Each element in the string begins with a mnemonic code identifying the element type (the sole exception is the file defining element which has no code and appears as the first item in the string and immediately following each Z-code, end-of-file, element) followed by modifier and sub-elements delimited by commas. The entire update resulting from a BATCH-string is accumulated and processed in a single step. Any errors encountered during the processing of the string aborts the process and none of the updates occur. Additionally multiple updates to the same file-items will not work from the same BATCH-string; the last file-item update will override any previously generated updates by the string to the same file-item.

There is a one-to-one correspondence between attribute-defining elements in the BATCH-string and the attributes themselves. In other words, the attribute-mark-count (AMC) is not explicitly specified in the element, but is implied by the sequence of attribute-defining elements. The following example illustrates this principle by showing the effect of BATCH-string and input-line updating on items.

PREVIOUS-ITEM	1234 ^ ABC ^ DEF ^ GHI
BATCH-STRING	file-name, I ^ N ^ N ^ A,Y21^
INPUT-LINE	1234 JKLM (r)
UPDATED-ITEM	1234 ^ ABC ^ DEF ^ JKLM ^

The 1234 in the input-line "feeds" the BATCH-string and provides the item-id to go with the file defining element (file-name, I). The two N elements in the string direct BATCH to ignore (nop) the next two attributes (whose previous values were ABC and DEF respectively). The JKLM in the input-line "feed" the attribute-defining element (A,Y21) and directs BATCH to replace the previous value (GHI) in that attribute with the new value (JKLM) from the input-line.

Input Data Conventions

BATCH-string elements reference fields in the input-line that the BATCH processor requests. When BATCH is evoked, data is requested from the terminal. Each input-line from the terminal is processed against the BATCH-string; a null input-line terminates BATCH processing, and causes a return to TCL. The first input-line from the terminal may be in one of the special formats below, specifying to BATCH that may be in one of the special formats below, specifying to BATCH that the actual data input is to be obtained from a disk-file item, or from the attached mag-tape unit.

(file-name item-id)	instructs BATCH to use the
(DICT file-name item-id)	specified items as the input

	data stream. Each attribute value (line) in the item is treated as one input-line, lines may be up to 511 bytes in length. The optional DICT specifies a reference to the dictionary instead of the data-file.
(TAPE) (TAPE C) (TAPE A)	instructs BATCH to read data from the tape unit. Each fixed length (max 511 chars.) unblocked tape record is treated as one input-line. The optional C parameter causes EBCDIC conversion; A causes masking to 7 bit ASCII.

Values from the input-line may be used by BATCH in either a free field or fixed field format. BATCH uses an input-line pointer to reference data from the input-line. Each file-defining or attribute defining BATCH-string element uses one value from the input-line; at the conclusion of this usage the input-line pointer points to the character immediately following the previously used value. Special BATCH-string elements are available which permit movement of this pointer so that the same value from the input-line may be used repetitively. A free form input-line value starts with the first non-blank character from the current position of the input pointer and continues up to but not including the next blank. Values with imbedded blanks may be created by surrounding the entire value with a single quotes or by replacing the imbedded blank with a backslash" ". A backslash surrounded by blanks represents a null missing value. Fixed field input-line values may be defined by specifying a starting column number and a field width. Fixed field values always contain the number of characters specified and may contain embedded blanks. All leading and trailing blanks are deleted. Two or more contiguous blanks will be condensed to only one blank.

BATCH-string File-defining Element

The first element in a BATCH string is a file-defining element. This element generates an item-id in a file which will be updated. A BATCH-string may contain additional file-defining elements immediately following a "Z" element; the first file specified is referred to as the "primary file"; other files are referred to as "secondary files."

file-name , d [(m,n)] [,C(m,n)] [, conversion spec]

file name :	if the file containing the item to be updated. If prefixed by the word DICT, the dictionary will be updated.

d :	I	update existing item.
	N	overwrite existing item.
	V	verify that item exists
	X	delete existing item.
	A	ensure item does not exist.
[(m,n)] [n] [(m)] [(,n)]		represents the location in the input-line of the value to be used as the item-id. m points to the column of the field to be processed. n is the number of characters to be processed. Either m or n or the entire parenthetical specification may be missing. If m is missing, free field format is assumed with the field beginning at the next non-blank input-line character.
[,C(mn,)] [,Cp] [,C(m)] [,C(,n)]		is optional, specifying concatenation with another value. If the m and n are missing, value will concatenate with next free field value in input-line. There may be multiple C specifications.
[,conversion spec]		is optional, and is used to specify input data conversions. All conversions as defined in the chapter CONVERSIONS may be used for the conversion specification.

In addition, for "secondary files" only, the following sub-element is defined:

```
...,BC(n),...
   BV(n),
```

which specifies that item-ID's are to be created (BC) or verified (BV) from the primary item AMC "n" (in new item image). B subelements may be repeated to specify concatenated ID's, or may be combined with I, J, K, or C subelements as needed.

Notes: If the attribute is multi-valued, it can only be used as the first element of a concatenated specification. D2 attributes should not be used with a bridge sub-element, i.e., BC(n) or BV(n).

BATCH-string Attribute-defining Elements

Following the file-defining element there must be one and only one for each attribute to be updated. The BATCH-string elements and the file attribute have a one-to-one positional relationship. Attributes not updated beyond the last updated attribute need not be represented in the BATCH-string. The mnemonics for the attribute-defining elements are: A, D_n, N, T and X.

A [(m,n)] [, C(m,n)] [, conversion spec] , Yn

Update attribute. The A statement in a BATCH-string describes what is to be done to the consecutive attributes in the item.

A(m,n) — same as file-defining element.

,C(m,n) — same as file-defining element.

,conversion spec — same as file-defining element.

Yn — must be part of the A command and is used to control how the input data is stored.

n — must be

- 11 to store non-redundantly (unique multiple values)
- 12 to store redundantly (non-unique multiple values)
- 21 to replace a single value
- 22 to reject a single value if value already present (prints error message)
- 23 to ignore a single value if value already present
- 31 to add to the existent value
- 32 to subtract from the existent value
- 33 to add one to the existent value
- 41 store multiple values in ascending sequence; sequencing is done on the ASCII collating sequence only. Only unique values are stored. May not be used for D1 attributes
- 42 as above; redundant values are also stored.

Additionally a 4 may be suffixed to the arithmetic value to place a "negative balance not permitted" restriction on the result. An example: Y324.

If the attribute to be modified is a co-related attribute (see D1, D2 CORRELATIVES) the A command takes the form:

D1;x [(m,n)] [,C(m,n)] [, conversion spec] , Yn

D2;x [(m,n)] [,C(m,n)] [, conversion spec] , Yn

where D1;x or D2;x replaces A. Three sets of co-related attributes may be processed by BATCH. The primary co-related attribute is identified with D1. Subsequent secondary attributes related to this primary attribute use the letter D2. x is 1, 2 or 3 providing for the three sets.

If co-related attributes (D1,D2) are to be deleted, using B/DEL, the A command takes the form:

```
D1;x,Y1X
D2;x,Y1X
```

The above attribute-defining element used with B/ADD will have the same effect as:

```
D1;x,Y11
D2;x,Y11
```

The following attribute-defining elements may also be used:

N -	ignore the corresponding attribute
nN -	ignore n attributes
T -	add one to the corresponding attribute
X -	delete the corresponding attribute.

Additional BATCH-string Elements

In addition to file-defining and attribute-defining elements the following elements may also be used:

F	Move input pointer forward to the next field
B	Move input pointer back to the prior field
S(n)	Set input pointer to the nth column in the input
Z	Terminate a file-item update section. Must be immediately followed by another file-item update section.

EXAMPLE--

In the example on the following page, an item '99-A' in the file PO-NUMBER is updated using a BATCH-string. Additionally an item in a secondary file, TRANS is created.

Additional Sub-elements

The following sub-elements may be used (unless otherwise noted) in both file-defining and attribute defining BATCH-string elements.

[, F op (m,n)] is used to specify arithmetic operations on the input data.

op must be

* multiplication. Following * must be a positive scaling factor. If specified, the product will be divided by 10 to the power of the scaling factor.

/ division

+ addition

- subtraction

The (m,n) maintains its meaning. It isolates an input field from which numeric information can be extracted. Several F options can be appended to an A command.

[,Um] is used to exit to a user defined program.

[,SD] store system data

[,SH] store system time

Note: no input data field is necessary if these two immediately follow the A, D1 or D2.

J(n) This sub-element should only occur in secondary file attributes, and is ignored if (a) the update is B/ADD or (b) the update is B/DEL with some specific data input to the field used by this element. If no data is input (meaning delete the entire value in the attribute), the nth primary file attribute is referenced, and the value that was deleted there is also deleted from the second file attribute. Only the first value used in the delete processing is used. Consequently, multi-valued primary attributes will in general be processed incorrectly. The 'J' sub-element must be specified if the primary item itself is to be deleted.

ITEM BEFORE BATCH-UPDATE

```
PO-NUMBER : 99-A
AL.   ABC
A2.   10
A3.   11
A4.   22
A5.   29 OCT 1974
A6.   $ 0.33
A7.   -44
A8.   1
A9.   9
A10   33:44
A11   44
A12   31 OCT 1974
```

ITEM AFTER BATCH UPDATE

```
PO-NUMBER : 99-A
A1.   ABC
A2.   11
A3.   11   12
A4.   34   22
A5.   02 NOV 1974
A6.   $ 0.89
A7.   -122
A8.   2
A9.   10
A10   56:78
A11   78
A12   31 OCT 1974
```

EXECUTION OF BATCH STRING

```
:B/ADD BS UPD-PO (r)
>99-A 12 34 11/2/74 56 78 90 (r)
'99-A' UPDATED
```

```
UPD-PO
001 PO-NUMBER,I
002 N
003 T
004 A,Y11
005 A,Y12
006 A,D,Y21
007 A,Y31
008 A,Y32
009 A,Y33
010 A1,TPO-NUMBER;V;2;2,Y21
011 F
012 F
013 F
014 A,C,Y21
015 B
016 A,Y21
017 A,SD,Y21
018 Z
019→TRANS,I2
020→A2,F+,Y21
```

12

46

Section XV

MICRODATA REALITY REFERENCE MANUAL

INTRODUCTION

This section is a reference manual for the Microdata Reality CPU. It provides a description of the system structure; of the arithmetic, logical, branching, skipping, and input/output operations; and of the interrupt and storage management system. Input/output devices are discussed in a separate document.

SYSTEM STRUCTURE

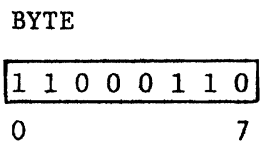
The Reality system consists of a core storage unit, a disk storage device used as a virtual storage unit, a central processing unit (CPU) and from one to 64 input/output terminals. There is a one-to-one correspondence between a terminal attached to the system and a process. Additionally, input/output devices such as magnetic tape units, disk units, card readers and printers may be attached to the system. Input/output devices, other than the process terminal, may be accessed by any process. It should be noted that the disk unit containing the virtual store cannot be accessed as an input/output unit, except by the monitor.

Information Formats

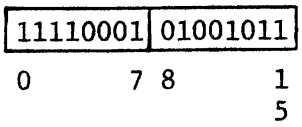
The system transmits information between the CPU and core storage, and between core storage and virtual storage in units of 8 bits or in multiples of 8 bits at a time. Each 8 bit unit is called a byte.

Information may be a single byte, or may be grouped together in fields. Fields of two, four, and six bytes are called words, double words and triple words respectively. A field made up of an arbitrary number of bytes is called a string. The location of any field is specified by the address of the left most byte of the field. Addresses increase from left to right.

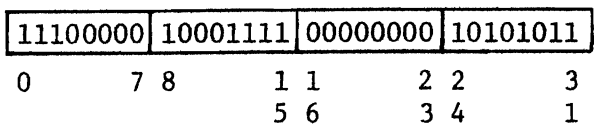
Within any information format, the bits making up the format are numbered from left to right starting with 0. The figure below shows the information formats.



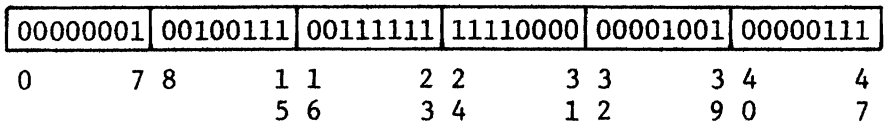
WORD



DOUBLE WORD



TRIPLE WORD



Addressing

Byte locations in main storage are consecutively numbered starting with zero. Each number is the address of a byte. A group of bytes is addressed by the leftmost byte of the group. The number of bytes in a group is either implied or explicitly defined by the operation. The addressing mechanism uses a 16 bit binary address giving a maximum of 65,536 addressable bytes. Main storage is available from 8,192 bytes to 65,536 bytes in 8,192 byte increments. Main storage is partitioned into blocks of 512 bytes each. A main storage block is called a buffer.

Virtual storage is also partitioned into blocks of 512 bytes each. A block of virtual storage is called a frame. Frames are numbered consecutively starting with zero. Each number is the address of a frame. A frame address is also called a frame identification (FID). FID's are 24 bit binary numbers giving an addressing capacity of 16,777,216 frames or 8,589,934,592 bytes. Virtual storage is available in 9,744 frame (4,988,928 byte) increments.

All program references to information are references to virtual storage. Fields in virtual storage are referenced via a frame number and a displacement. If the field being referenced is a single byte or a string, the displacement is the number of bytes relative to the first data byte of the frame. If the reference is to a word, double word or triple word, the displacement is the number of words relative to the first data byte of the frame. References to instructions are via a 12-bit frame number. Therefore, programs must be located in the first 4,096 frames.

VIRTUAL MEMORY MANAGEMENT

The CPU directly accesses information from buffers in main storage. These buffers contain the contents of virtual storage frames. The virtual frames are moved between the disk and main storage as required by processes in progress. Two of the main storage buffers, 0 and Z, contain the monitor program that performs the actual operation of swapping frames in and out of main storage. Main storage buffers 1 and 3 contain information about each of the main storage buffers and a map of the frames currently contained in each main storage buffer.

Buffer Status

Main storage locations X'200' through X'27F' contain the status of each of the main storage buffers. One byte is used for the status of each buffer. Location X'200' contains the status of buffer 0, location X'201' contains the status of buffer 1 and so forth. The information contained in the buffer status byte is given below.

Buffer Status Byte

Bit	
0	I/O BUSY/
1	CORELOCK1/
2	CORELOCK
3	WRTREQD/
4	
5	
6	
7	

PSYM Name	Bit	Description
	0	Zeroed whenever an I/O (disc or peripheral) is in progress for this buffer; set when I/O completes. Firmware prevents "attachment" by a virtual process to a buffer with this bit zero.
	1	This bit is zeroed during cold-start tape generation, along with bit 2, if a buffer is to remain core-locked.

PSYM Name	Bit	Description
CORELOCK/	2	A zero indicates that this buffer may not be selected for disc input.
WRTREQD/	3	A zero indicates that data in this buffer has changed since it has been read from disc, and must therefore be written back to disc.
	4	Unused
	5-7	These bits are used by the 'FAR' instruction which changes the buffer status.

Buffer Map

Main storage locations X'280' through X'2FF' and locations X'700' through X'7FF' contain the addresses of the frames currently in the main storage buffers. The map is divided into two sections. Locations X'280' through X'2FF' contain the least significant byte of each of the frame addresses. Locations X'700' through X'7FF' contain the most significant two bytes of each of the frame addresses. For example, the virtual storage address of the frame in buffer 4 is found by concatenating the contents of main storage bytes X'708', X'709', X'204'.

Buffer Queue

A buffer queue is maintained by the firmware in main storage locations X'300' through X'3FF'. The buffer queue consists of a doubly linked list of buffer numbers ordered according to their time of attachment by the firmware. Each time a register is attached to a buffer, the firmware moves the attached buffer to the head of the buffer queue.

When the contents of a buffer must be replaced because of a frame fault, the buffer queue is used to identify the least recently attached buffer for replacement. The buffer used is then moved to the head of the buffer queue.

Each entry in the buffer queue consists of two bytes. The word (two bytes) displacement of the entry from main storage location X'300' corresponds to the buffer number. The two bytes forming each entry in the buffer queue are the two pointers forming the doubly linked list.

The first byte of each entry points to the next more recently attached buffer entry in the queue. The second byte of each entry points to the next less recently attached buffer entry. The first byte of the most recently attached entry (i.e., head of the queue) contains X'FF'. The second byte of the least recently attached entry (i.e., tail of the queue) contains zero.

Bytes at locations X'300' and X'301' contain pointers to the head and tail of the buffer queue respectively. The head of the queue identifies the most recently attached buffer number. The tail of the queue identifies the least recently attached buffer number.

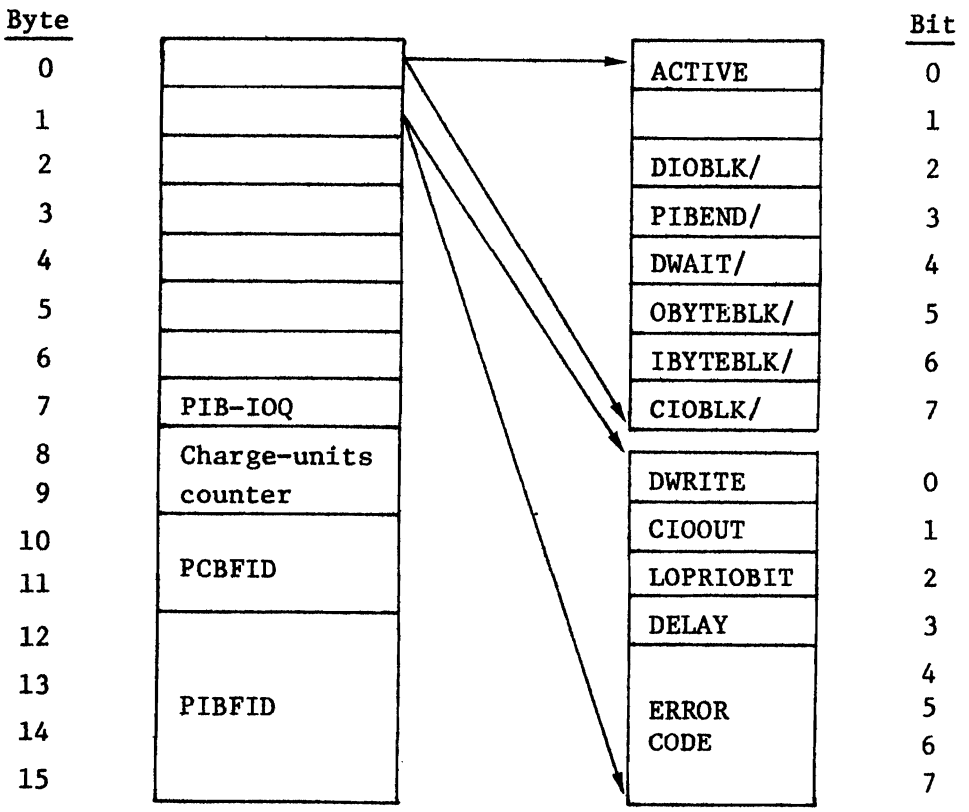
PROCESS

The Reality CPU is designed as an interactive system capable of communicating with several users simultaneously. A user communicates with the system via a communication terminal such as a Teletype or CRT terminal. Associated with each terminal is a process. A process is not an element of the system but rather a continuing operation on a set of functional elements. Refer to table at end of Chapter XV for peripheral I/O details.

Process Identification Block

For each process attached to the system, there is a Process Identification BLOCK (PIB). Each PIB is 32 bytes long. The PIB for terminal zero is in main storage locations X'800' through X'81F'; locations X'820' through X'83F' contain the PIB for terminal one, and so forth. The PIB contains information about the status of the process with which it is associated. The following is a description of the PIB contents. Bytes 0 through 6 are determined by firmware.

PIB Status Bytes



PIB Status Bytes

Name	Byte	Bits	Meaning
ACTIVE	0	0	One indicates that process may be activated (candidate for Select Next User process).
--	0	1	Unused
DIOBLK/	0	2	Zero (zeroed by firmware on a frame fault) indicates that process is roadblocked waiting for referenced frame to be input from virtual storage. Set to one when monitor accepts request by moving FID onto IOQ.
PIBEND/	0	3	Zero indicates the end of the PIBs.
DWAIT/	0	4	Zeroed by monitor when frame fault request is accepted (FID moved to IOQ). Set to one when disc transfer is complete.
OBYTEBLK/	0	5	Zeroed by firmware when process is roadblocked waiting for terminal to complete output. Set to one when output is complete.
IBYTEBLK/	0	6	Zeroed by firmware when process is roadblocked waiting for terminal to complete input. Set to one when input is complete.
CIOBLK/	0	7	Zeroed by monitor when process is roadblocked waiting for concurrent I/O block transfer to complete. Set to one when block transfer is complete.
DWRITE	1	0	One indicates read request is roadblocked waiting for buffer to be written out to disc. Zeroed when output is complete and read request has been replaced in IOQ.
CIOOUT	1	1	One indicates process is roadblocked waiting for concurrent output to complete. Zeroed when output is complete.
LOPRIOBIT	1	2	Set to one on a Release Quantum entry to monitor, if the process does not have either a byte input or byte output roadblock. Also set to one during concurrent block output.
DELAY	1	3	One indicates a one-cycle delay to the Select Next User process, on a Release Quantum entry to monitor.

Name	Byte	Bits	Meaning
ERROR CODE	1	4-7	Software generated error trap codes: 08 - illegal FID 09 - disc error 0C - register zero detached 0E - charge-units counter overflow
--	2	0-7	Last byte address of PIB I/O buffer (bytes 16-31 of PIB).
--	3	0-7	Number of bytes in PIB I/O buffer less one (X'FF' = no bytes).
--	4	0-7	Mask byte used by Communications controller.
--	5	0-7	"Unusual status" of Communications Controller line associated with this PIB.
--	6	0-7	Data byte received from Communications controller line associated with this PIB.
PIB--IOQ	7	0-7	Pointer connecting PIB to IOQ entry.
Charge-units counter	8-9	0-15	Number of charge-units associated with this process.
PCBFID	10-11	0-15	Frame-id of the Primary Control Block (PCB) for this process.
PIBFID	12-15	0-31	When a process is roadblocked because of a frame fault, the frame-id is placed in these bytes. When the monitor is entered as a result of a call operation, these bytes contain parameters:

			Frame I/O Request	Monitor Call
--	12	0-7	Buffer number which contains PCB.	High-order address of PCB frame
--	13	0-7	High-order byte of absent FID.	Mask byte from Call instruction.
--	14	0-7	Middle byte of absent FID.	High-order byte of address of register referenced in Call.
--	15	0-7	Low-order byte of absent FID.	Low-order byte of address of register referenced in Call.

--	16-31		Input/output buffer for the terminal associated with this process.

Primary Control Block

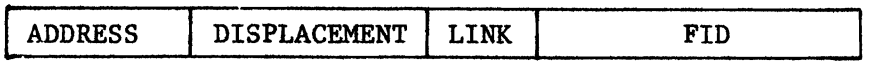
For each process there is a frame called the Primary Control Block (PCB). The PCB contains the accumulator, address registers, subroutine return stack and string scan control characters associated with the process. The location of the PCB is contained in the PIB of the process. The following paragraphs describe the contents of the PCB. The bytes that are not described are not accessed by the Firmware. However, the remaining bytes of the PCB contain information used by the operating system.

Bytes	Description
0	This byte is reserved for a lock code used for storage protection.
1	This byte contains the condition code resulting from a previous arithmetic instruction execution.
3-5	These bytes are used for controlling the Move and Scan through Delimiter instructions.
6-7	These bytes are used for controlling the debug trace mode of operation.
8-X'0B'	These bytes contain the double word accumulator extension. The accumulator extension contains the most significant portion of a product after a multiply operation. It contains the remainder after a divide operation.
X'0C'-X'0F'	These bytes contain the double word accumulator.
X'100'-X'17F'	These bytes contain the 16 address registers. See the description of the address registers below.
X'180'-X'181'	These bytes contain the address (relative to byte zero of the PCB) of the limit of the subroutine stack.
X'182'-X'183'	These bytes contain the pointer to the current top of the subroutine stack.
X'184' and above	The bytes contain the subroutine return stack. The number of bytes allocated for the stack is determined by the contents of bytes X'180' and X'181'.

Address Registers

All references to data, except immediate data, are made indirectly through an address register. There are 16 address registers in each PCB. Each address register contains 8 bytes. The following paragraphs describe the address register format.

Address Register Format



Bytes	Description
0-1	These bytes contain the 16 bit main storage address of the referenced data. If the address is less than X'800', the frame containing the data may be absent from main storage.
2-3	These bytes contain the displacement of the referenced data relative to the first data byte of the frame. The displacement is a 16-bit signed number. Negative values are represented in two's complement form. These bytes are meaningful only when the register is detached. (See Register Attachment below.)
4	Zero in bit zero of this byte indicates that the register references data in the linked format. If bit zero is a one, the register references the data in the unlinked format. One in bit one indicates that frame attachment is in progress. Bit one can only be set during the execution of instructions that increment addresses with data movement.
5-7	These bytes contain the virtual storage frame number of the byte being referenced.

Address Register Attachment

When a program loads ("restores") an address register, the first two bytes of the register are set to zero. Bytes 2 through 7 of the address contain a virtual frame number and displacement. A register in this format is said to be detached. When a subsequent instruction uses the detached register for a data reference, an attempt is made to convert the address register to the attached format. The attaching attempt is automatic and performed as follows. The buffer map is scanned to determine if the referenced frame is located in main storage. If the frame is in main storage, the location of the required byte is computed by adding the buffer address from the map to the displacement from the address register. The address is then

placed into bytes 0 and 1 of the address register, thus forming the attached format. Once the register is attached, instruction execution takes place.

If the referenced frame is not in main storage, the frame number is placed into bytes 12 through 15 of the PIB. Byte 0, bit 2 of the PIB is set to 0, thus roadblocking the process. Next all of the address registers in the PCB are converted to detached format and a fault interrupt to the monitor is taken.

Address Register Zero

Register zero is used in a special way. This register always contains the FID of the PCB. Register zero is attached when the process is activated. The displacement field of this register is always assumed to be zero.

Address Register One

When a process is not active, address register one contains the FID and displacement (minus one) for the next instruction to be executed. When the process is activated, the buffer address of the program frame (as determined from the buffer map) is added to the displacement from register one. This value is placed into a hardware instruction counter. The register is then converted to the attached form with the buffer address set to the base address (byte zero) of the program frame. When the process is deactivated, the main storage location from the instruction counter is converted to the corresponding FID and displacement and the register is detached with these values placed into it.

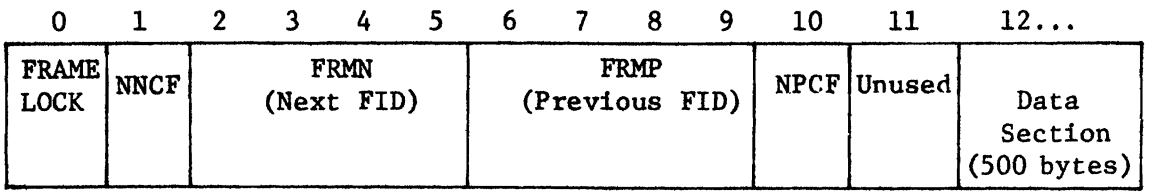
Frame Formats

The Reality system recognizes two types of frame formats; linked and unlinked. In both formats byte zero of the frame is reserved for a frame lock.

Unlinked frames contain 511 data bytes. For unlinked frames the displacement portion of an address is relative to byte 0 of the frame, i.e., a displacement of 1 is a reference to the first data byte. Displacements outside the range 0 through 511 are not valid for frames in the unlinked format.

Linked frames contain 500 data bytes. For linked frames, the displacement field in the address is relative to byte 11 of a frame. However, a displacement of zero is a reference to byte 511 of the frame to the left of the current frame. Displacements for linked

frames may be positive or negative so long as the displacement references a logically linked item of data. The following paragraphs describe the linked format.



Linked Frame Format

Bytes	Description
0	This byte is reserved for a frame lock.
1	This byte contains a count of the number of next contiguous frames to the right of this frame (NNCF). A zero in this byte indicates that this frame is the rightmost frame in a contiguously linked set of frames.
2-5	This field contains the frame number of the frame that is logically to the right of this frame. If byte 1 contains other than zero, the frame to the right is the next higher numbered frame. If byte 1 contains a zero the frame to the right may be any frame number. A zero in this field indicates that this is the rightmost frame of a linked set.
6-9	This field is similar to bytes 2 through 5 except that it contains the number of the frame to the left of this frame.
10	This byte is similar to byte 1 except that it contains a count of the number of previous contiguous frames to the left of this frame (NPCF).
11	Unused.
12-511	Data section.

MONITOR

The monitor is a program that is an integral part of the Reality system. The monitor process is the only one not associated with a PIB. The PCB for the monitor is defined as buffer 0 of main storage.

The function of the monitor is to initiate the transmission of information between main storage buffers and virtual storage and to schedule each of the processes.

When the system is operating in monitor mode, address registers are not checked for attachment. Instead all data references are assumed by the firmware to be references to absolute core addresses. The system is in monitor mode whenever the location of the PCB is at core-address zero.

The monitor gives control to another process by executing either a Resume Virtual Process or a Start Virtual Process instruction.

The multi-disc monitor may be used with one or two drives per controller, and with one through four disc controllers. In any multi-disc configuration, the capacity of every drive in the system must be the same; i.e., either 5 megabyte or 10 megabytes/drive; also all controllers must have the same number of drives attached to them.

Monitor PCB

The PCB associated with the monitor is at absolute core-address 0 through X'1FF'. Beside the functional elements that are described in the section "Primary Control Block," the following locations are used:

Bytes	Description
2	Contains the Interrupt Address code on an External interrupt fault trap to the monitor.
3	Contains monitor status flags (bits).
6	Contains the hardware clock counter; a fault is generated when this is incremented (every one millisecond) to zero.
7	Extension of clock counter used by the monitor.
X'10'-X'1F'	Exclusively a hardware save area.
X'20'-X'FF'	Contains the bootstrap software executable code.
X'1A0'-X'1A1'	Contains the system date (days since 31 Dec 1967).
X'1A4'-X'1A7'	Contains the system time (seconds since midnight).
X'1C0'-X'1DF'	Contains PIB pointers for peripheral devices 0 through 15.
X'1E0'-X'1FF'	Contains address pointers for peripheral devices 0 through 15.

Initial Condition of Monitor PCB Registers

LOC								
100	R0	00	00	03	05	BITS		
108	R1	04	00	01	C0	01	E0	02	7F
110	R2	06	00	14	80	10	F0	0C	84
118	R3	0C	00	FF	FF	20	A0	28	3F
		PIBWA		PIBSTART		PIBSIZE		PFID	
120	R4	--	--	07	E0	00	20	--	--
128	R5	01	--	F6	--	FE	64	06	3F
130	R6	01	60	--	--	00	01	51	81
			IOQWAL	IOQSTART	IOQSIZE	IOQMAX	IOMAX	IPQ#	NUMCONT
138	R7	06	--	*	10	*	*	00	*
			DCTWAL			FIDMAX			
140	R8	06	--	--	04	*			
					=H24				
148	R9	00	0B	0C	18	02	00		
150	R10	--	--	--	--	--	--	--	--
158	R11	0E	--	--	--	--	--	--	--
160	R12	--	--	--	--	--	--	--	--
168	R13	--	--	--	--	--	--	--	--
170	R14	--	--	--	--	--	--	--	--
178	R15	--	--	--	--	--	--	--	--
180		--	A0	--	84	FF	FF	FF	FF

-- : Unused or scratch

* : Preset by MSETUP Program.

Monitor Register Assignment

Register No.	PSYM Name	Address	Description of Usage
0	None	X'0000'	Addresses Monitor PCB.
1	None	X'0400'	Addresses MMONITOR.
2	None	X'0600'	Addresses MMONITORX.
3	None	X'0C00'	Addresses MMONITORY/Nx.
4	PIB	Variable	Current PIB pointer.
5	None	X'0100'	Addresses Monitor PCB, lower half.
6	None	X'0160'	Addresses WA of R12.
7	IOQ	X'06xx'	Current IOQ entry pointer.
8	DCT	X'06xx'	Current DCT entry pointer.
9	None	X'000B'	Addresses H4 in Monitor accumulator.
10	None	--	Scratch
11	None	X'0E00'	Addresses MMONITORZ.
12-15	None	--	Scratch

Interrupts and Monitor Calls

Once a virtual process gains control, it remains in control until the occurrence of an interrupt or until the process executes a Monitor Call instruction. The occurrence of a Monitor Call instruction will cause all registers in the current PCB to be converted to the detached form.

There are three types of interrupts to the monitor; external, internal and fault.

An external interrupt is generated when a device (including the virtual storage device) completes an operation. When an external interrupt (excluding the virtual storage device) occurs, the status of the active process is saved in the hardware. The process that was interrupted must be resumed by executing a Resume Virtual Process instruction. After the occurrence of an external interrupt, further external interrupts will be inhibited until a Resume Virtual Process instruction has been executed by the monitor. Refer to Disc Interrupt Handling in this chapter for an explanation of external interrupts from the Virtual Storage Device. Device addresses 0-X'F' are assumed to be non-virtual storage devices and X'10' - X'17' are assumed to be virtual storage devices.

When an internal interrupt occurs with a dependent process active, all registers in the PCB are converted to detached form and control passes to the monitor. An internal interrupt can be recognized with the monitor active at any time except in the case of a real time clock runout which can only be recognized after the execution of a Test Interrupt instruction (X'01').

A fault interrupt can occur only when a virtual process is active. A fault interrupt causes all the registers in the PCB to be converted to detached form.

The monitor must execute a Start Virtual Process instruction to start a process that was interrupted by an internal or fault interrupt.

Interrupts cause entry to the monitor at predefined locations. The table below shows the monitor entry point for each interrupt condition.

Entry Address	Interrupt Condition
1	Reference to absent frame (fault)
3	Input/output operation complete (external)
5	Power fail console interrupt, or clock runout (internal)
7	Terminal input/output with device not ready, or attempt to attach a buffer with input or output active in the buffer (fault)
9	Attempt to attach register 0 when not in the monitor (fault)
11	Power restored entry point (internal)
13	Hardware abnormal condition while in Monitor mode
15	Not used
17 thru 31	Monitor Call instruction entry points

Traps

Certain operations can cause a trap condition to be signaled. The occurrence of a trap causes a Branch and Stack location instruction

to be executed to a predefined location in virtual storage frame one. The table below shows the entry point and the cause for each trap.

Entry Address	Cause
1	Illegal operation code encountered.
3	The return stack is empty. This occurs when a Return instruction is executed with the current stack position pointing to the beginning of the return stack.
5	The return stack is full. This occurs when a stack location type of operation is executed and the current stack position is equal to the end of stack location. The current stack location is reset.
7	Attempt to reference frame 0 when not in monitor mode. The number of the address register that contained the reference is placed into the condition code byte of the PCB.
9	Attempt to cross a frame boundary for an unlinked frame or a word, double word, or triple word not entirely in one frame. The register number containing the reference is placed into the condition code byte of the PCB.
11	Attempt to link across a frame with a forward link of zero. The register number containing the reference is placed into the condition code byte of the PCB.
13	Attempt to link across a frame with a backward link of zero. The register number containing the reference is placed into the condition code of the PCB.
15	Attempt to execute a privileged opcode when not in monitor mode.
17	Attempt to reference a non-existent frame.
19	Disk error.
21	Break key activated on the terminal.
23	Return stack format error. There are two conditions that cause this error. Either the end of stack location is less than the current stack position, or the stack size is defined for less than 7 entries.
31	Debug trace mode. This is not an error condition. Trace mode is controlled by bytes 6 and 7 of the PCB.

Trace Mode

The Reality CPU can operate in a special mode called trace mode. When trace mode is in effect the hardware monitors the instruction execution and traps to location 31 of frame 1 on the occurrence of certain conditions. When a trap occurs, bits 0 through 3 of byte 6 of the PCB are set to zero, inhibiting further traps. The conditions that can cause the trap are defined in bytes 6 and 7 of the PCB. The following table shows the conditions.

Byte 6, Bit 1 - This bit indicates that a trace interrupt is to occur on every BSL, ENT, BSL*, or ENT* instruction (modal trace).

Byte 6, Bit 3 - This bit indicates that trace traps can occur on every instruction.

Byte 6, Bit 5 - This bit is set by firmware when an instruction trace trap has occurred.

Byte 6, Bit 6 - This bit is set by firmware when a RTN trace trap has occurred.

Byte 6, Bit 7 - This bit is set by firmware when a BSL, ENT, ENT* or BSL* trace trap has occurred.

Byte 7 - When the system is in the instruction trace mode, this byte is incremented for every instruction executed. A trace trap will not occur until this byte has been incremented to zero.

Monitor Disc Scheduling Tables

There are two tables that control the disc I/O scheduling, the I/O Queue (IOQ) and the Device Control Table (DCT). The IOQ table may be considered a subset of the Process Identification Block set (PIB); a process that requires disc input must first be allotted a spot in the IOQ before its request can be honored. Since the IOQ can be set to any size between two and eight entries, the IOQ acts as a funnel between the disc input requests from the processes, and the actual disc I/O. By preventing the honoring of requests for too many processes in rotation, the IOQ serves to control "thrashing".

The IOQ Table

Moving on and off the IOQ is controlled by the monitor in the following manner: When a process requests disc input, and it is not on the IOQ, it is moved on to an available IOQ entry if such an entry exists at the time; if not, the process goes into a wait state until an entry

becomes available. A process moves off the IOQ ("deactivates") under the following conditions:

1) The process executes a "Release Quantum", either explicitly (via the RQM instruction) or implicitly due to:

   a) Real Time Clock Interrupt

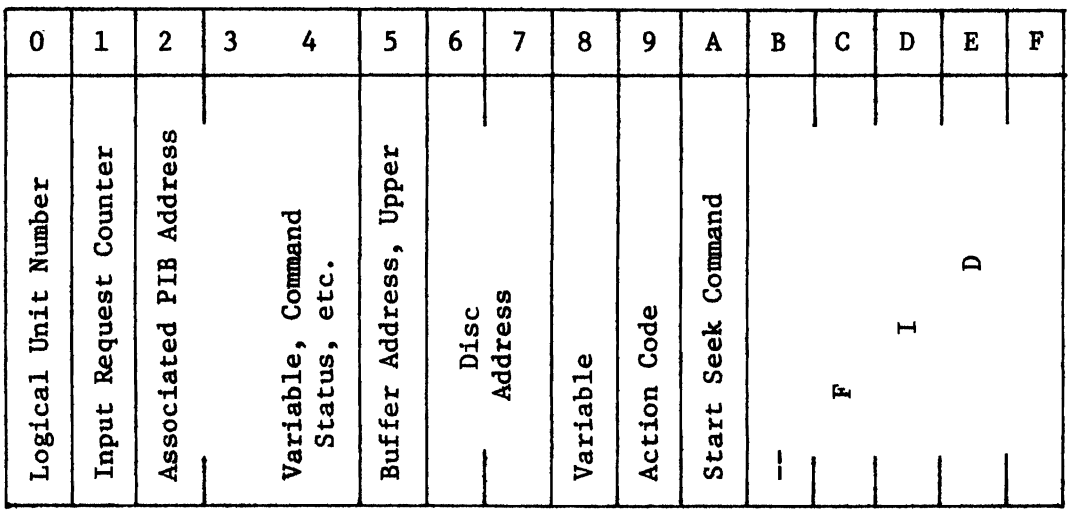
   b) Terminal input or output roadblock

   c) Concurrent I/O block transfer roadblock

2) The process executes the maximum number of disc input requests, as specified by the monitor parameter IOMAX.

In any of the above cases, that process is taken off the IOQ; the entire PIB table is then searched for other processes that are roadblocked due to a disc input request, and one of them selected to be moved to the IOQ, at which time the disc input request counter (IOCTR) is set to the initial value specified in IOMAX.

IOQ TABLE FORMAT



Byte	PSYM Name	Description
0	None	Logical unit number; consists of the controller device address, with the high-order bit specifying the drive number:

Controller Address -		14	15	16	17
Drive Number	0	14	15	16	17
	1	94	95	96	97

Byte	PSYM Name	Description
0 (Continued)	None	This byte is set up when the disc address is computed from the FID; is set to zero when the disc request has been processed.
1	IOCTR	Disc input request counter; is set to value specified in IOMAX when a process is moved to the IOQ; is decremented on every input request processed. If zero specifies an available IOQ entry.
2-3	IOQ--PIB	Link from IOQ to PIB; is set up when a process is moved to the IOQ; is never zeroed.
4	IOQCOM1	Scratch location used to communicate with the disc controller may store a RETURN command, a SELECT and QUEUE SEEK command, or the major status.
5	IOQBUF	Buffer address, (upper) to which disc I/O is being done; the low-order bit is first zeroed to output the "buffer start, upper" command, to the disc controller, then set to output the "buffer end, upper" command.
6-7	IOQDA	Disc address computed from the FID.
8	None	Used as a scratch location to output the "buffer start, lower" and "buffer end, lower" bytes (always X'00' and X'FF' respectively).
9	IOQACT	Controller action code: X'00' - Read X'01' - Verify X'02' - Write
A	None	Controller command--always X'90' (start queued seek(s), arm interrupt).
B	-	Unused
C	IOQFIDO	FID, uppermost byte is always X'80'.
D-F	IOQFID	

Selection of a Process to be placed on the IOQ

The following rules are used to select a process to be placed on the IOQ (if more than one process is roadblocked due to a disc input request):

1) If the process had been taken off the IOQ due to a terminal I/O roadblock, it is selected to be placed on the IOQ immediately. This is governed by bit LOPRIOBIT being zero in the PIB.

2) If none of the processes are so roadblocked, the first process with LOPRIOBIT set (therefore lower priority) is selected to be placed on the IOQ.

The effect of this selection criterion is that processes that had moved off the IOQ due to terminal I/O will have a higher priority than processes that do not do any I/O. In order that the latter type of processes do not get into a state where they may never be selected due to the existence of other processes that are in a heavy terminal I/O state, LOPRIOBIT is zeroed during the search described above.

Note that in the event that only one process requires disc input, a full search of the PIB table is completed, before the deactivated process is moved back to the IOQ.

IOQ Setup

When a process is placed on the IOQ due to a disc input request, the following sequence of events occurs:

1) The disc input roadblock (DIOBLK/) in the PIB status is set, thereby indicating that this process is on the IOQ, and is no longer a candidate for IOQ selection.

2) The "waiting for disc" flag (DWAIT/) is zeroed, thereby preventing the Select Next User routine from selecting that process for execution.

3) The requested FID is moved from the PIB to the IOQ, and the disc address is computed.

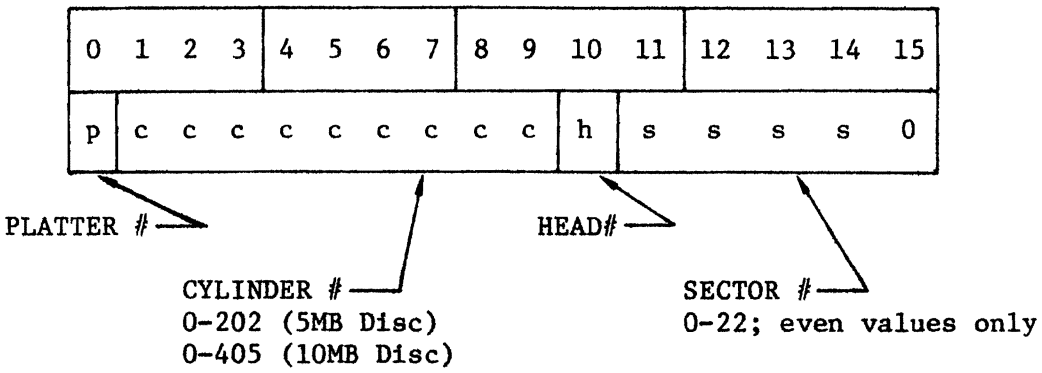
4) a) If the addressed disc is busy, nothing further can be done.

   b) If not, the 'SETUP' routine in the disc interrupt handler is entered.

Disc Address Computation

Subroutine SETIOQ takes the FID specified in the IOQ entry; checks against a maximum FID as specified in the literal FIDMAX; converts the FID to a 16-bit disc address and stores the latter at IOQDA.

Disc Address Format

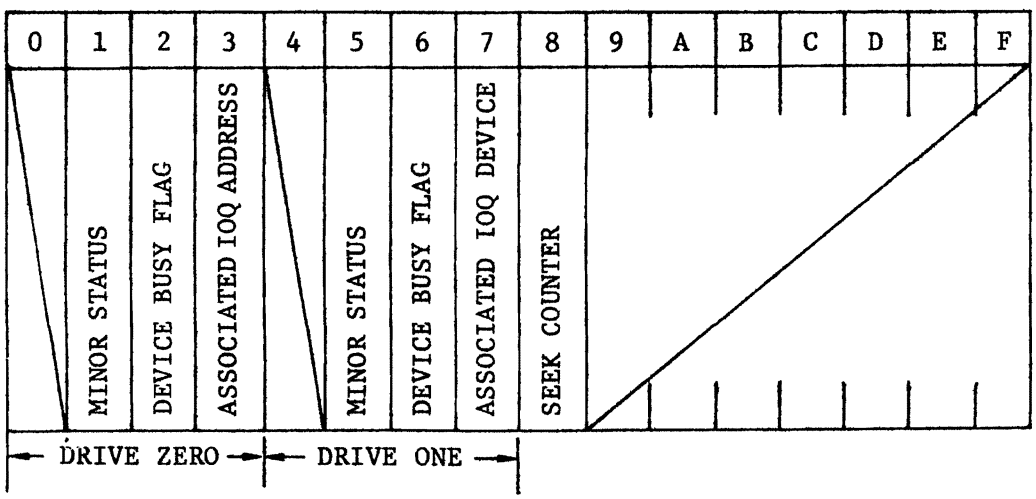


Device Control Table

The DCT entry is uniquely associated with a disc drive; it contains a flag indicating whether the device is busy or not, a link to the IOQ associated with the drive, and in the case of two drives per controller, a seek counter which keeps track of the number of seeks that have been started on the controller.

The DCT location is mapped directly from the device address of the controller, and the disk drive number; its main function is to provide an easy linkage from the interrupt address supplied by the CPU at the time of an interrupt, to the IOQ.

DCT TABLE ENTRY (per controller)



Byte	PSYM Name	Description
1	None	Minor status from drive if an error has occurred; is not reset if no errors.
2	DCTBSY	High-order bit, if set, indicates drive is busy. Reset on I/O completion.
3	DCT--IOQ	Address link from DCT to associated IOQ Entry; is setup when seek is started; is not reset.

Disc Interrupt Handling

On receiving an end-of-transfer interrupt, the firmware deactivates the currently executing virtual process (if in virtual mode), and traps to location X'403' in the monitor. The interrupt address of interrupting device (device address times two) is stored at X'0F' - the low-order byte of the monitor accumulator. A virtual process cannot resume execution after completion of disc interrupt handling, since its buffers may be replaced and attached registers may no longer be valid.

The interrupt address is mapped into the DCT address, which leads to the associated IOQ address, which in turn leads to the associated buffer address, FID, and PIB address. The status of the drive is obtained, and, if there are no errors, the controller is re-armed (if another seek is pending it is also re-started). If the completed operation was a write, a verify is now started, and interrupt handling terminates.

Selection and setup of next I/O

Since the drive is now ready, the IOQ table is searched for a matching device/drive number, or logical unit number. If a match is found, the setup phase is entered (also entered from frame fault).

1) The FID is picked up from the IOQ, and a FAR instruction executed; this is the only monitor-level instruction that causes the firmware to attach an A/R. If the requested FID is core-resident, the disc read roadblock is removed from the associated PIB, and the IOQ table searched for the next I/O.

2) If the FID is not core-resident, the execution of the FAR instruction has automatically caused an attachment to the "oldest" buffer in core. If this is core-locked, the FAR is repeated. If not, and if the buffer has no write-required flag on its status, an available spot has been found for the disc read, and the read parameters are set up.

3) If a write-required flag exists on the buffer status, that data must be written out before the read request can be processed; therefore, the IOQ entry is overwritten with the FID to be written out, and the write parameters are set up in the IOQ. Also, the flag DWRITE in the PIB status is set, indicating that the read request from the process is yet to be processed. Note that in this case a drive other than the one that just completed a transfer may be started.

Starting I/O

In the case of two drives per controller, a RETURN instruction is issued to the controller if another seek is pending completion (DSCSEEKS non-zero); if the controller status indicates that it did not return, the start I/O sequence is aborted, and the associated process arbitrarily reactivated by clearing its PIB roadblocks. This is because the controller either

a) is transferring data at this time, in which case it cannot be interrupted to queue another seek, or

b) it has completed a transfer, in which case an interrupt is pending recognition and another seek cannot be queued.

When reading a frame, the buffer FID is set to zero till the read completes; when writing a frame to the disc, the FID of the buffer remains unchanged. In either case, the buffer status is set core-locked and I/O busy for the duration of the I/O.

Disc Errors

If a disc error is detected at the completion of the transfer, the minor status from the drive is stored in the DCT; the buffer is set non-core-locked, not-I/O-busy, the associated process PIB roadblock is cleared just as if the I/O had completed, and error #9 is set in the PIB error byte. This causes a virtual software trap to the DEBUG State, and an eventual re-stacking of the request. Note that the buffer FID is maintained if the transfer was a write, and is left as zero if the transfer was a read.

If an illegal FID is requested (as determined by comparing against FIDMAX in the monitor PCB), as above, except that error #8 is flagged. The DEBUGGER will abort the process in this case.

Select Next User Routine

The Select Next User routine (SNU) is entered whenever the monitor has completed setting up a disc transfer; has completed processing of a disc interrupt; or is in a "wait" state due to all virtual

processes being quiescent. While the monitor is in the SNU routine, it cycles through the PIB's to see if any of the processes require activation. At this time also, the monitor executes the Test Interrupt instruction, which is the only monitor-executed instruction that allows recognition of external interrupts.

A process may be selected under the following conditions:

1) All roadblocks clear; that is, if DIOBLK/, DWAIT/, OBYTEBLK/, IBYTEBLK/, CIOBLK/ are set, and DELAY and DWRITE are zero.

2) External activation and disc roadblocks clear; that is, if ACTIVE, CIOBLK/ and DWAIT/ are set. This would happen if the process received a BREAK-key interrupt, or if another process sent this one a message via the MESSAGE processor.

In either case, the monitor sets up to activate the process by loading the PCB-FID from the PIB into R4FID (of the monitor), and executing a Start Virtual Process instruction.

If the process had been roadblocked due to a disc input request, and the monitor had overwritten this request with a disc write, all status bits are as in (1) above, except that DWRITE is set. In this case, the monitor will cause the read request to be re-stacked, by re-entering the Frame Fault entry point.

Programming Notes

The current IOQ entry is addressed by the address register "IOQ" (R7); the current DCT entry by the A/R "DCT" (R8); in the case of two discs per controller, R12 addresses byte zero of the DCT block for the current controller.

Registers DCT and IOQ always work in the same 256-byte block (X'600 - X'6FF') therefore, the upper bytes of their address words are preset to X'06' (as an initial condition when cold-starting the system), and only their low order address bytes are altered. Also, the IOQ table starting address pointer (IOQSTART) is a half-tally.

The DCT location is from X'640' through X'67F' (see formats later); sixteen bytes are allocated to each of four possible controllers, (with device addresses X'14' through X'17' respectively), with one or two drives per controller.

The IOQ table has a fixed ending address--the last entry is X'6F0' through X'6FF'; the starting address is variable, depending on the

number of entries allowable. The table beginning pointer (IOQSTART) is 16 bytes before the first IOQ entry:

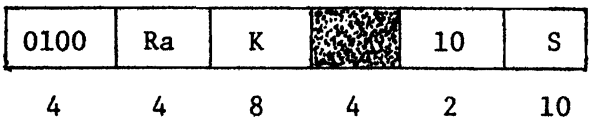
IOQSTART=X'700' - 16* (IOQMAX+1)

where IOQMAX is in the range 2 through 8 inclusive.

INSTRUCTION DESCRIPTIONS

This section lists all computer instructions and describes their execution. A diagram representing the format is given with each instruction description. Preceding the diagram is the name of the instruction. Enclosed in parentheses is the assembler code for the instruction. It should be noted that the assembler codes are not unique. That is several of the instructions have the same code. The assembler uses both the code and the operand attributes to determine a particular operation. Below is an example of an instruction description:

Branch Byte Equal to Immediate (BCE)



The numerical operation code bits are shown as binary numbers. Note that the operation code need not occupy consecutive bit positions of an instruction. In the example above the operation code occupies the first and fifth fields of the instruction. The numbers appearing beneath the diagram indicate the number of bit positions occupied by the particular field. The symbols appearing in the diagram indicate the type of information in the field. Shading indicates that the field is not used in the instruction. The table below defines the symbols used in the instruction diagrams.

Symbol	Meaning
R	The field contains an address register number.
D	The field contains a displacement relative to the contents of an address register or relative to the beginning of a frame.
FID	The field contains a frame identification number. If the field is less than 24 bits wide, high order zeroes are assumed.
S	The field contains a signed magnitude skip distance (in bytes) for conditional skipping. A skip distance of zero means no bytes are to be skipped.

Symbol	Meaning
L	The field contains an operand length L = 0 is a 1 byte operand. L = 1 is a 2 byte operand. L = 2 is a 4 byte operand. L = 3 is a 6 byte operand.
K	The contents of the field itself is an operand.

Definitions of Terms Used in the Descriptions

Ra or Rb means the contents of the addressing register named by the Ra or Rb field of an instruction.

C(Ra) or C(Rb) means the contents of the location referenced by the address contained in the named addressing register.

C(Ra,Da) or C(Rb,Db) means the contents of the storage frame location referenced by adding the D field of the instruction to the contents of the named storage register.

When a register or part of a register is cleared, the cleared part contains zero bits.

When the word 'load' is used in a description it means the contents of some frame location replaces the contents of a special register (address register or accumulator).

When the word 'store' is used in a description it means the contents of a special register or the contents of an instruction field replaces the contents of some frame location.

When the word 'move' is used in a description it means that the contents of a frame location replaces the contents of another frame location, or the contents of a register replaces another register.

Effective Address Computation

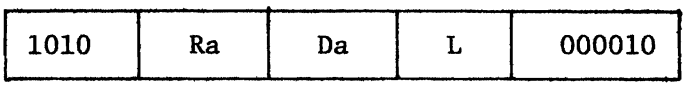
Storage operands are always referenced through one of the 16 addressing registers. An addressing register contains the byte address of the operand. For instructions with a D field, a displacement is added to form an effective address. When the operand is a single byte (L field = 0), the D field of the instruction is the displacement. When the operand is a word, double word or triple word (L = 1, 2 or 3) the D field is doubled to form the displacement.

ARITHMETIC OPERATIONS

The following operations perform arithmetic on binary integers. Negative values are represented in two's complement form. The 'L' field of the instruction specifies the length of the operand in storage. For storage to accumulator operations, triple word operands are not allowed (L field of 3); byte and word operands are sign extended to form a double word value before the operation is performed. The accumulator operand is always a double word. Storage operands must lie entirely in a single frame. The condition codes resulting from an arithmetic operation are placed in byte 1 of the PCB. The condition codes are defined below.

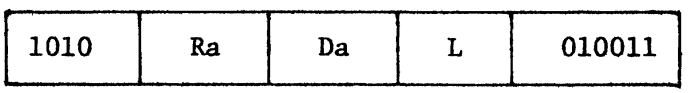
Symbolic Name	Bit Position	Condition Indicated
ZROBIT	5	zero result
NEGBIT	6	negative result
OVFBIT	7	arithmetic overflow

Test and Set Arithmetic Condition Flags (TST)



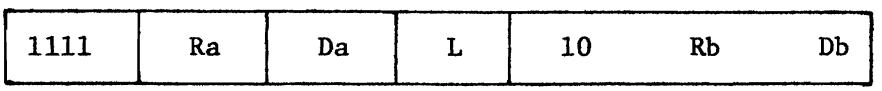
The contents of (ra,Da) is tested and the arithmetic condition flags (i.e., ZROBIT and NEGBIT) are updated appropriately. The instruction may be used with a half tally, a tally, or a double tally. L is '00', '01', or '10' respectively depending upon the type of operand.

Add to Accumulator (ADD)



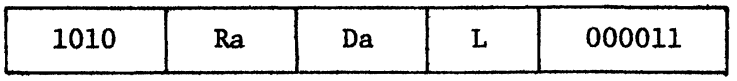
The C(Ra,Da) are added algebraically to the accumulator. The sum is placed in the accumulator. The C(Ra,Da) are unchanged.

Add to Storage (INC)



The C(rb,Db) are added algebraically to the C(Ra,Da). The sum is placed in the C(Ra,Da). The C(Rb,Db) are unchanged.

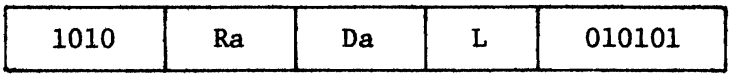
Add a One to Storage (INC)



1010	Ra	Da	L	000011
4	4	8	2	6

The C(Ra,Da) are algebraically increased by 1.

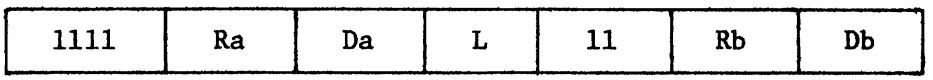
Subtract from Accumulator (SUB)



1010	Ra	Da	L	010101
4	4	8	2	6

The C(Ra,Da) are algebraically subtracted from the accumulator. The difference is placed into the accumulator. The C(Ra,Da) are not changed.

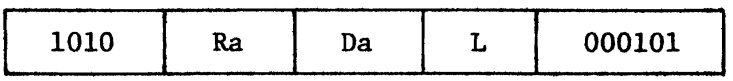
Subtract from Storage (DEC)



1111	Ra	Da	L	11	Rb	Db
4	4	8	2	2	4	8

The C(Rb,Db) are algebraically subtracted from the C(Ra,Da). The difference replaces the C(Ra,Da). The C(Rb,Db) are not changed.

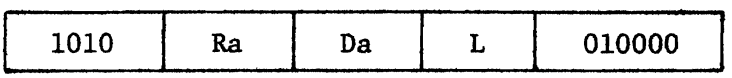
Subtract One from Storage (DEC)



1010	Ra	Da	L	000101
4	4	8	2	6

The C(Ra,Da) are algebraically decreased by 1.

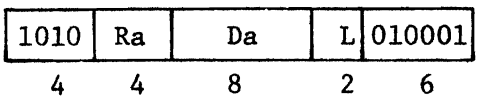
Multiply (MUL)



1010	Ra	Da	L	010000
4	4	8	2	6

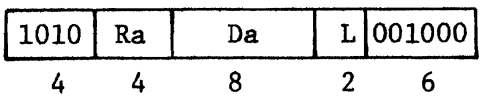
The contents of the accumulator are multiplied by the C(Ra,Da). A 64 bit product replaces the contents of the accumulator and accumulator extension. The sign of the product is determined by the rules of algebra. The C(Ra,Da) are not changed.

Divide (DIV)



The sign of the accumulator is replicated into the accumulator extension to form a 64 bit dividend. The C(Ra,Da) are divided into the dividend to form a 32 bit quotient and a 32 bit remainder. The quotient replaces the contents of the accumulator and the remainder replaces the contents of the accumulator extension. The sign of the quotient is determined by the rules of algebra. The sign of the remainder is the sign of the dividend. The C(Ra,Da) are not changed.

Negate (NEG)



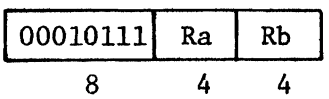
The sign of the C(Ra,Da) is changed.

DATA TRANSMISSION OPERATIONS

The following operations are concerned with the transmission of data between storage locations, between registers, and between registers and storage locations.

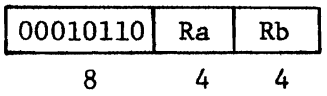
For instructions in this group, operands in storage must be within a single frame.

Exchange Address Registers (XRR)



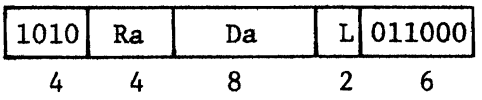
Ra and Rb are exchanged.

Move Address Register to Address Register (MOV)



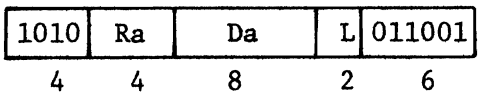
Ra replaces Rb. Ra is not changed.

Load Accumulator (LOAD)



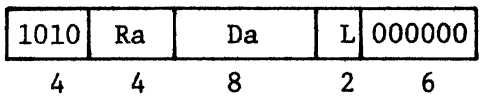
The C(Ra,Da) replace the contents of the accumulator. The C(Ra,Da) are not changed. The accumulator is sign extended to form a double word value.

Store Accumulator (STORE)



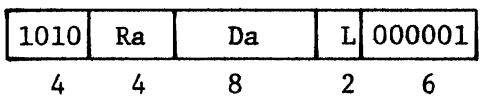
The contents of the accumulator replaces the C(Ra,Da). The contents of the accumulator is not changed.

Store a Zero (ZERO)



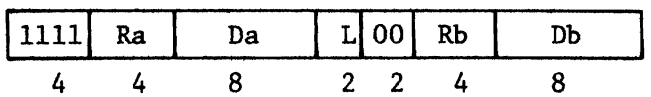
The C(Ra,Da) are replaced with zeros.

Store a One (ONE)



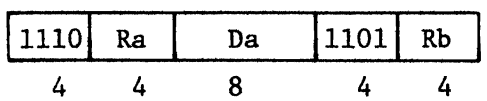
The C(Ra,Da) are replaced with a 1.

Move (MOV)



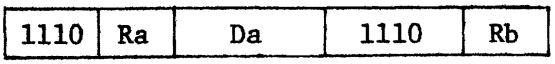
The C(Rb,Db) replace the C(Ra,Da). The C(Rb,Db) are not changed.

Store Address Register (MOV)



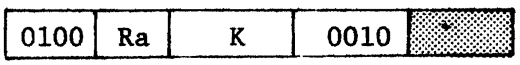
The detached form of Rb is stored into the C(Ra,Da). A triple word is stored. Rb is not changed. Da is doubled to form the effective address.

Load Address Register (MOV)



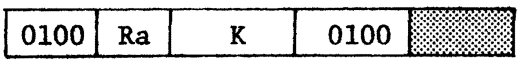
The C(Ra,Da) replace the 6 low order bytes (displacement and FID) of Rb. The high order byte of Rb (buffer location) is set to zero. The C(Ra,Da) are not changed. Da is doubled to form the effective address.

Move Immediate Character (MCC)



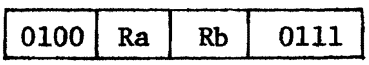
The byte, K, replaces the C(Ra).

Increment and Move Immediate Character (MCI)



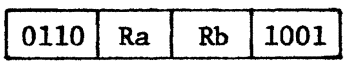
Ra is incremented by 1 and then the byte, K, replaces the C(Ra).

Exchange Characters (XCC)



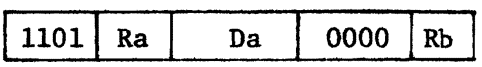
The single byte in C(Ra) is exchanged with the single byte in C(Rb).

Move Character (MCC)



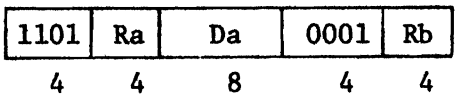
The single byte in C(Rb) replaces the byte in C(Ra). The C(Rb) are not changed.

Move Character to Relative Character (MCC)



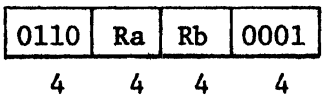
The single byte at the C(Rb) replaces the byte at the C(Ra,Da). The C(Rb) are not changed.

Move Relative Character to Character (MCC)



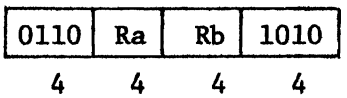
The single byte at the C(Ra,Da) replaces the byte at the C(Rb). The C(Ra,Da) are not changed.

Increment Source Register and Move Character (MIC)



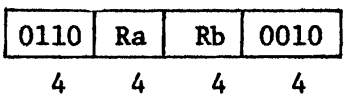
Ra is incremented by 1 and then the single byte in C(Ra) replaces the C(Rb). The C(Ra) are not changed.

Increment Destination Register and Move Character (MCI)



Ra is incremented by 1 and then the single byte at C(Rb) replaces the C(Ra). The C(Rb) are not changed.

Increment Both Registers and Move Character (MII)

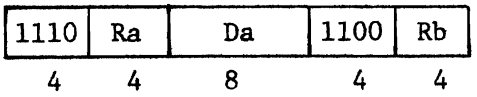


Ra and Rb are each incremented by 1 and then the single byte at C(Ra) replaces the C(Rb). The C(Ra) are not changed.

ADDRESS MODIFICATION OPERATIONS

The following group of instructions are used to modify the displacement portion of an addressing register.

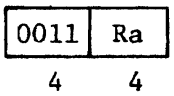
Load Absolute Address Difference (LAD)



This operation treats the triple word in the C(Ra,Da) as a storage address. The absolute value of the difference between this address and the address in Rb is computed. The result is a two byte integer. The result replaces the contents of the low-order accumulator.

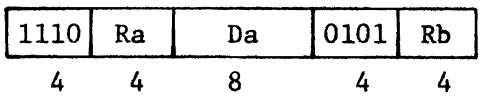
NOTE: This instruction is valid for unlinked frames only if the frame number in C(Ra,Da) is the same as the frame number in Rb. The instruction is valid for unequal frame numbers only if both frames are in the same group of contiguously linked frames and the difference between the frame numbers is less than 32.

Increment Address Register (INC)



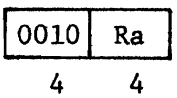
The displacement portion of Ra is incremented by one.

Add to Address Register (INC)



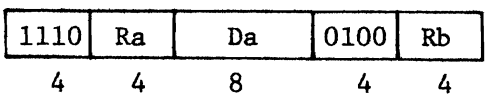
The two byte integer at the C(Ra,Da) is added to the displacement portion of Rb. The C(Ra,Da) are not changed. The Da field is doubled to form the effective address.

Decrement Address Register (DEC)



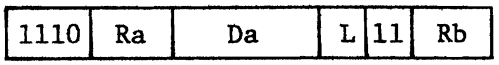
The displacement in Ra is decremented by 1.

Subtract from Address Register (DEC)



The 16 bit integer at the C(Ra,Da) is subtracted from the address portion of Rb. The C(Ra,Da) are not changed. The Da field is doubled to form the effective address.

Load Effective Address (SRA)

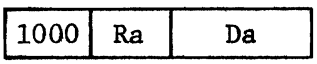


An effective address is computed using the contents of Ra and the Da field. The Da field is doubled if the L field is not zero. The resulting effective address replaces Rb. Ra is not changed.

BIT MANIPULATING INSTRUCTIONS

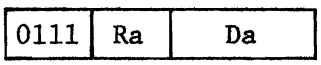
The following two instructions are used to manipulate individual bits.

Set Bit On (SB)



Da is a bit displacement relative to the byte address in Ra. The most significant bit of the byte has a bit number of zero. The least significant bit of the addressed byte has a displacement of seven. The bit in the C(Ra,Da) is set to 1.

Set Bit Off (ZB)



Da is a bit displacement relative to the byte address in Ra. The most significant bit of the byte has a bit number of zero. The least significant bit of the addressed byte has a displacement of seven. The bit in the C(Ra,Da) is set to 0.

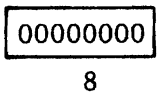
CONTROL INSTRUCTIONS

Instructions that govern the flow of a program, and in particular cause an alteration of the process of taking instructions from sequential locations, are called control instructions.

Branch instructions specify the frame and word displacement relative to the start of the frame from which the computer is to take the next instruction.

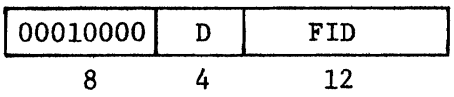
Skip instructions specify the number of bytes to be skipped in order to reach the next instruction. The skip amount is relative to the first byte of the instruction following the skip instruction. The skip amount is a 10 bit field represented in sign magnitude form. For skip instructions, a skip out of the current frame causes a fault trap to location 9 of frame 16.

No Operation (NOP)



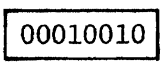
This instruction causes the computer to take the next instruction in sequence.

External Branch (ENT)



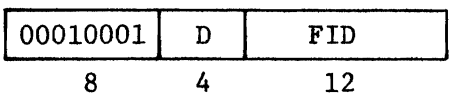
This instruction causes the computer to take its next instruction from the location specified by the FID and D fields. The D field is doubled to determine the branch location relative to byte 1 of the frame. Only the first 16 words of a frame can be specified as branch locations. The first 16 words of a procedure frame normally contain entry vectors.

External Branch Indirect (ENTI)



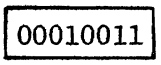
This instruction is similar to the Branch instruction except that the D and FID fields are contained in the two low order bytes of the accumulator.

Branch and Stack Location (BSL)



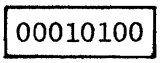
The location following this instruction is stacked in the return stack. The next instruction is taken from the location specified by the FID and D fields. The D field is doubled to determine the location relative to byte 1 of the frame. The location is stored as a two byte FID and a two byte displacement.

Branch and Stack Location Indirect (BSLI)



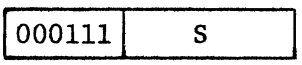
This instruction is similar to Branch and Stack Location except that the D and FID fields are contained in the two low order bytes of the accumulator. The location is stored as a two byte FID and a two byte displacement.

Return (RTN)



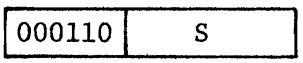
The address stored in the top of the return stack replaces the instruction counter and the return stack is popped. This causes the next instruction executed to be the one following the most recently executed Branch and Stack instruction.

Branch (B)



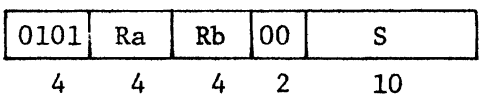
The number of bytes specified by S are skipped.

Branch and Stack Location (BSL)



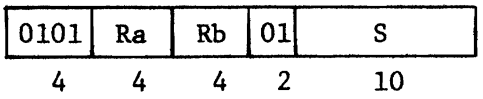
The location following this instruction is stacked in the return stack. Then the number of bytes specified by S are skipped. The location is stored as a two byte zero field and a two byte displacement.

Branch Character Not Equal (BCU)



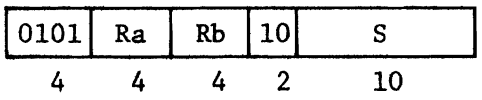
If the byte at the C(Ra) is not equal to the byte at the C(Rb), the number of bytes specified by S are skipped.

Branch Character Less Than (BCL)



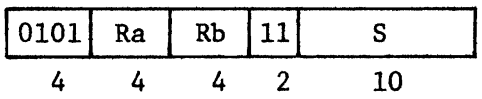
If the byte at the C(Ra) is less than the byte at the C(Rb), the number of bytes specified by S are skipped.

Branch Character Equal (BCE)



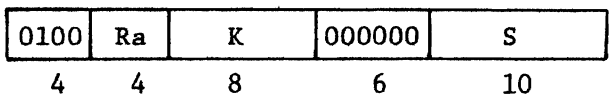
If the byte at the C(Ra) is equal to the byte at the C(Rb), the number of bytes specified by S are skipped.

Branch Character Less Than or Equal (BCLE)



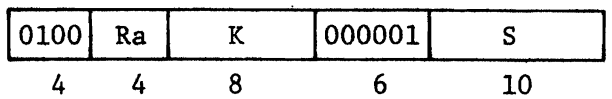
If the byte at the C(Ra) is less than or equal to the byte at the C(Rb), the number of bytes specified by S are skipped.

Branch Character Not Equal to Immediate (BCU)



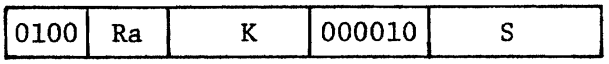
If the byte, K, is not equal to the byte at the C(Ra), the number of bytes specified by S are skipped.

Branch Character Less Than Immediate (BCL)



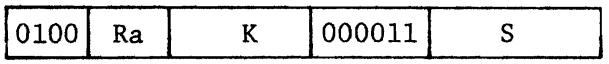
If the byte, K, is less than or equal to the byte at the C(Ra), the number of bytes specified by S are skipped.

Branch Character Equal to Immediate (BCE)



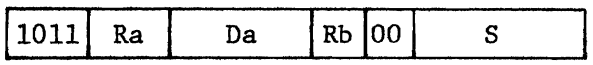
If the byte, K, is equal to the byte at the C(Ra), the number of bytes specified by S are skipped.

Branch Character Less Than or Equal to Immediate (BCLE)



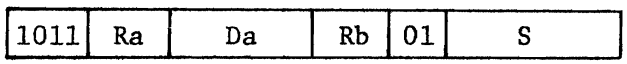
If the byte, K, is less than or equal to the byte at the C(Ra), the number of bytes specified by S are skipped.

Branch Relative Character Not Equal (BCU)



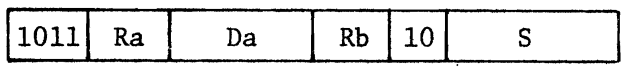
If the byte at the C(Ra,Da) is not equal to the byte at the C(Rb), the number of bytes specified by S are skipped.

Branch Relative Character Less Than (BCL)



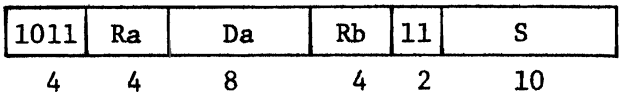
If the byte at the C(Ra,Da) is less than the byte at the C(Rb), the number of bytes specified by S are skipped.

Branch Relative Character Equal (BCE)



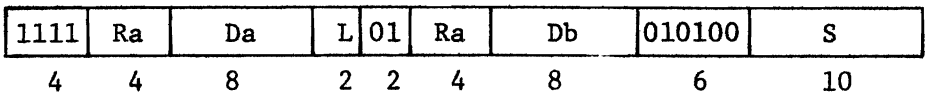
If the byte at the C(Ra,Da) is equal to the byte at the C(Rb), the number of bytes specified by S are skipped.

Branch Relative Character Less Than or Equal (BCLE)



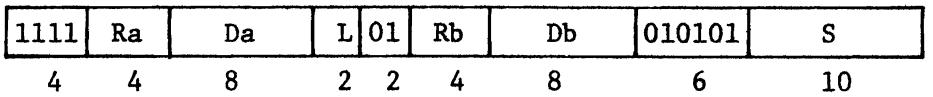
If the byte at the C(Ra,Da) is less than or equal to the byte at the C(Rb), the number of bytes specified by S are skipped.

Compare and Branch Not Equal (BU)



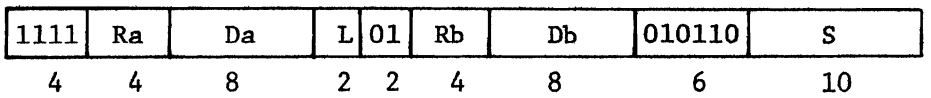
If the C(Ra,Da) are not equal to the C(Rb,Db), the number of bytes specified by S are skipped.

Compare and Branch Less Than (BL)



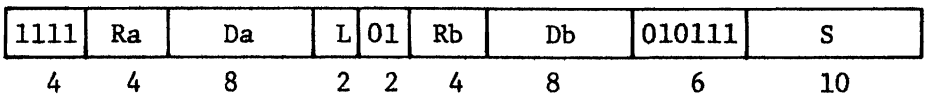
If the C(Ra,Da) are less than the C(Rb,Db), the number of bytes specified by S are skipped.

Compare and Branch Equal (BE)



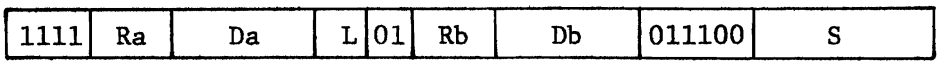
If the C(Ra,Da) are equal to the C(Rb,Db), the number of bytes specified by S are skipped.

Compare and Branch Less Than or Equal (BLE)



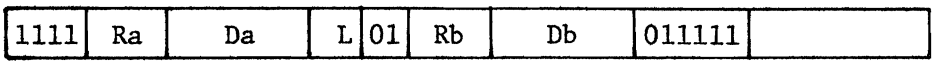
If the C(Ra,Da) are less than or equal to the C(Rb,Db), the number of bytes specified by S are skipped.

Subtract and Branch Not Equal (BDNZ)



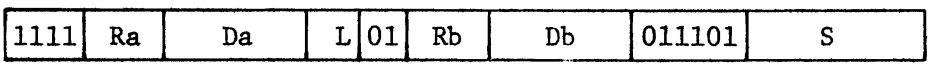
The C(Rb,Db) are subtracted from the C(Ra,Da). The difference replaces the C(Ra,Da). If the original C(Ra,Da) are not equal to the C(Rb,Db), the number of bytes specified by S are skipped. The condition codes are set.

Subtract and Branch Less Than or Equal (BDLEZ)



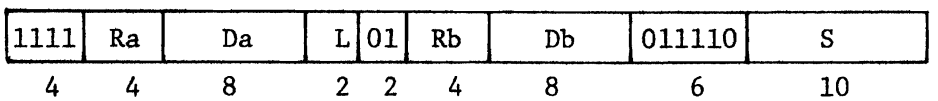
The C(Rb,Db) are subtracted from the C(Ra,Da). The difference replaces the C(Ra,Da). If the original C(Ra,Da) are less than or equal to the C(Ra,Da) the number of bytes specified by S are skipped. The condition codes are set.

Subtract and Branch Less Than (BDLZ)



The C(Rb,Db) are subtracted from the C(Ra,Da). The difference replaces the C(Ra,Da). If the original C(Ra,Da) are less than the C(Rb,Db), the number of bytes specified by S are skipped. The condition codes are set.

Subtract and Branch Equal (BDZ)



The C(Rb,Db) are subtracted from the C(Ra,Da). The difference replaces the C(Ra,Da). If the original C(Ra,Da) are equal to the C(Rb,Db), the number of bytes specified by S are skipped. The condition codes are set.

Branch Address Equal (BE)

The 6 byte (C(Ra,Da) are compared with the address of Rb. If the values are equal, the number of bytes specified by S are skipped. It is possible for two addresses to compare not equal even though they represent the same storage location. This can occur if the FID in the C(Ra,Da) is not the same as the FID in Rb. See Note under LAD instruction.

Branch Address Not Equal (BU)

The 6 byte C(Ra,Da) are compared with the address of Rb. If the values are not equal, the number of bytes specified by S are skipped. It is possible for two addresses to compare not equal even though they represent the same storage location. This can occur if the FID in the C(Ra,Da) is not the same as the FID in Rb. See Note under LAD instruction.

Branch Bit Set (BBS)

Da is a bit displacement relative to the byte address in Ra. The most significant bit of the addressed byte has a displacement of zero. The least significant bit of the addressed byte has a displacement of seven. If the bit in the C(Ra,Da) is on (bit = 1), the number of bytes specified by S are skipped.

Branch Bit Zero (BBZ)

Da is a bit displacement relative to the byte address in Ra. The most significant bit of the addressed byte has a displacement of zero. The least significant bit of the addressed byte has a displacement of seven. If the bit in the C(Ra,Da) is off (bit = 0), the number of bytes specified by S are skipped.

LOGICAL OPERATIONS

The following group of instructions perform logical operations between a byte in storage and an immediate operand. The logical operations are AND, OR (sometimes called "inclusive or") and XOR ("exclusive or").

When two bytes are combined by an AND, they are matched bit for bit. If the same bit position in each byte contains a 1, the result is a 1. If a position of either byte (or both bytes) contains a 0, the result is 0.

The following is an example of a logical AND operation:

```
10110001
00110110
00110000   resulting AND
```

When two bytes are combined by an OR they are matched bit for bit. If the same bit position in each byte contains a 0, the result is a 0. If a position of either byte (or both bytes) contain a 1, the result is a 1. The following is an example of logical OR:

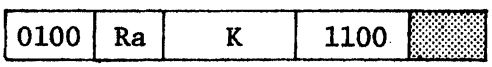
```
11011000
00010001
11011001   resulting OR
```

When two bytes are combined by an XOR they are matched bit for bit. If the corresponding bit positions in each byte are the same (both 0 or both 1) the result is 0. If the same bit position in each byte is not the same (either contains a 1 while the other contains a 0) the result is 1.

The following is an example of a logical XOR operation.

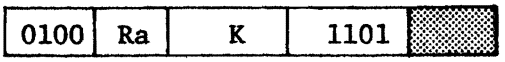
```
10011100
00011011
10000111   resulting XOR
```

AND Character (AND)



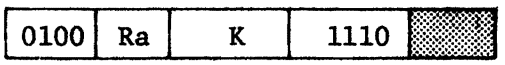
The byte in the C(Ra) and K are logically AND'ed. The result replaces the C(Ra).

OR Character (OR)



The byte in the C(Ra) and K are logically OR'ed. The result replaces the C(Ra).

Exclusive OR Character (XOR)

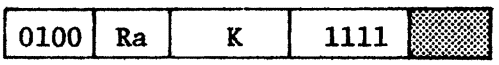


The byte in the C(Ra) and K are logically exclusive OR'ed. The result replaces the C(Ra).

SHIFT OPERATION

The following instruction allows a byte to be shifted one bit to the right.

Shift Character Right (SHIFT)

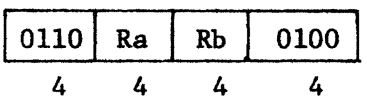


K is shifted right one bit position. A zero bit is inserted on the left and the bit shifted off the right is lost. The result replaces the C(Ra). The value K in the instruction does not change.

STRING OPERATIONS

The following instructions operate on strings. A string is a logically contiguous group of bytes. Strings may extend across frame boundaries provided that the frames are linked.

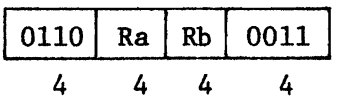
Increment and Move String Under Count Control (MIIT)



The contents of the lower half of the accumulator (T0) is read into internal hardware registers. Ra and Rb are each incremented by one and then the byte at c(Ra) replaces the byte at c(Rb). Next the internal hardware registers are decremented. If the resulting value is not zero the increment and move is repeated. If, during the

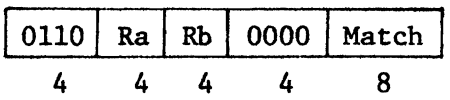
execution of the instruction, an internal interrupt occurs or if the move crosses the frame boundary of a linked frame, the current contents of the internal hardware registers are stored in TO. For this reason the contents of TO at the conclusion of the move are indeterminable. If TO is initially zero no operation is performed.

Increment and Move String Under Address Control (MIIR)



Ra and Rb are each incremented by one and then the byte at C(Ra) replaces the byte at C(Rb). Next Ra is compared with the contents of address register 15. If the values are not equal the operation is repeated. If Ra is initially equal to the contents of address register no operation is performed.

Increment and Move String Under Match Control (MIID)



Ra and Rb are each incremented by one and then the byte at C(Ra) replaces the byte at C(Rb). The byte that moved is then tested for a match with one of 7 possible values as defined by the match field. If the match is not successful, the operation is repeated.

The matching is performed as follows. For each of the bit positions one through seven that is a 1, a match test is performed. If bit position zero is a 1, the move stops on any equal match. If bit position zero is a 0, the move stops if none of the bytes tested match. The table below shows the test performed for each bit in the mask.

Bit in Match Field	Test Performed
0	1 = Stop on equal 0 = Stop if unequal
1	Compare with Hexadecimal "FF"
2	Compare with Hexadecimal "FE"
3	Compare with Hexadecimal "FD"
4	Compare with Hexadecimal "FC"
5	Compare with Byte at 003 in PCB
6	Compare with Byte at 004 in PCB
7	Compare with Byte at 005 in PCB

Note: byte 003 of the PCB may not contain a hexadecimal 00 or 01.

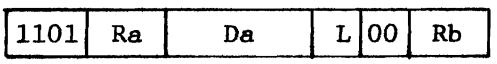
Increment and Scan String Under Match Control (SCD)

Ra is incremented by one and then the byte at C(Ra) is tested for a match as defined by the match field. If the match is not successful, the operation is repeated. See the description of the Increment and Move String Under Match Control instruction for the matching rules.

CONVERSION OPERATIONS

Conversion operations are provided to convert decimal integers represented by ASCII characters into binary values, and to convert hexadecimal integers into binary values, and binary values to hexadecimal.

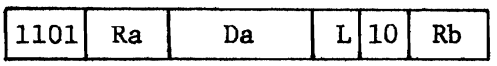
Decimal to Binary (MDB)



The C(Ra,Da) are multiplied by ten. The binary value of the ASCII digit in the C(Rb) is added to the product, and the result replaces the C(Ra,Da). This instruction is not defined for a single byte at C(Ra,Da). (A value of L = 0 represents a different operation.)

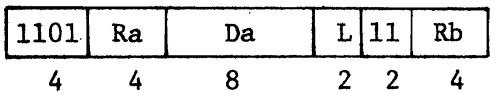
If the C(Ra,Da) are initially zero, repeated use of this instruction (with incrementing of Rb) will convert an ASCII string representing a decimal value into a binary integer.

Hexadecimal to Binary (MXB)



The C(Ra,Da) are multiplied by sixteen. The binary value of the ASCII hexadecimal digit in the C(Rb) is added to the product and the result replaces the C(Ra,Da). If the C(Ra,Da) are initially set equal to zero, repeated use of this instruction will convert an ASCII string representing a hexadecimal value into a binary integer.

Binary to Hexadecimal (MBX)

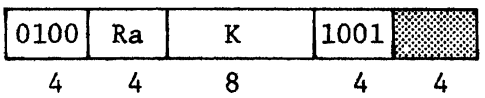


The binary integer at the C(Ra,Da) is converted into an ASCII string starting at the C(Rb) +1. Bits 28 through 31 of the accumulator contain a count of the maximum number of ASCII bytes to be generated. If bit 24 of the accumulator is a zero, the leading zeros of the hexadecimal string are suppressed and the C(Rb) +1 will contain the most significant non-zero hexadecimal digit. If bit 24 of the accumulator is a 1, zero suppression will not take place. The contents of the accumulator is unpredictable after this instruction is executed.

INPUT OUTPUT OPERATIONS

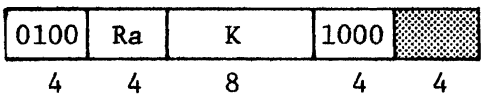
The input output operations provide for communication with the terminal associated with a process and for input and output with peripheral devices.

Input a Byte (IB)



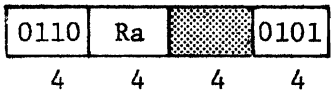
The K field specifies a 3 bit function code and a 5 bit device address. The byte from the selected device replaces the C(Ra). This instruction can be executed only in monitor mode.

Output a Byte (OB)



The K field specifies a 3 bit function code and a 5 bit device address. The single byte in the C(Ra) is transmitted to the selected device. This instruction can be executed only in monitor mode.

Read Input Queue (READ)



The next character from the terminal input queue replaces the C(Ra). If the input queue is empty the process is suspended until a character is received from the terminal. Characters transmitted by the terminal are automatically queued in the PIB for the terminal.

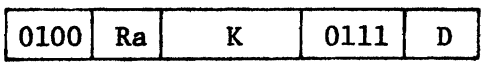
Write to Output Queue (WRITE)

The byte in the C(Ra) is placed into the terminal output queue. If the queue is full, the process is suspended until the terminal has printed all but four characters from the queue. If there are any characters in the input queue before this instruction is executed, they are lost.

MONITOR OPERATIONS

The following operations are used to communicate with the monitor.

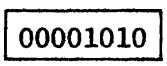
Monitor Call (MCAL)



This operation generates an interrupt into the monitor. The four bits of the D field are doubled to determine the location relative to byte 1 of the monitor frame for transfer of control.

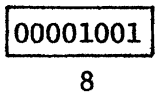
The address contained in Ra, the address of the PCB of the current process, and the K field from this instruction are all placed into the PIB for the current process, and then control is passed to the monitor.

Resume Virtual Process (RVP)



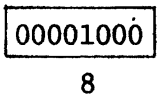
This operation returns control to a process that has previously been interrupted. The status of the interrupted process is restored and execution of the process resumes from the point of the interrupt. This instruction can be executed only in monitor mode.

Start Virtual Process (SVP)



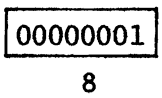
The FID portion of register 4 (of the monitor) is treated as the location of the PCB for a dependent process. The buffer map is searched for the PCB frame. If the PCB is present, registers 0 and 1 of the PCB are attached and execution of the process begins. If the frame is not present, the referenced FID is placed into the PIB and the monitor is reentered at the absent frame entry point (location 1). This instruction can be executed only in monitor mode.

HALT



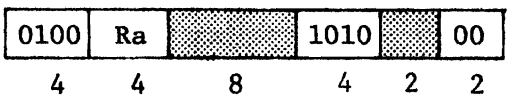
The CPU is halted; this instruction can be executed only in monitor mode.

Test Interrupts



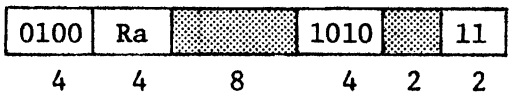
This instruction has meaning only in the monitor mode; it is a NOP in the virtual mode. Internal and External Interrupts are tested for, and, if any are pending, a fault trap to the appropriate monitor location is taken.

Halt and Display (HLD)



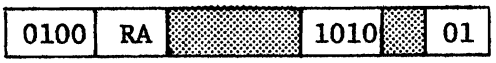
Halts the CPU and gates the eight-bit literal addressed by register RA to the A bus where it can be displayed in the eight least significant indicator lamps of the system panel by depressing the Data select switch. This instruction is restricted to Monitor level code.

Enter Console Command Switches (ECS)



The status of the eight low-order console command switches is placed in the eight bit byte addressed by register RA. If the switch is on, the corresponding bit in the byte addressed by register RA is set to

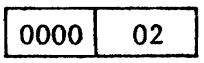
one. This instruction is restricted to Monitor level code. If a switch is not set, the corresponding bit will be set to zero.



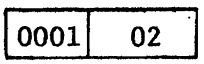
The status of the four console sense switches is placed in the four most significant bits of the eight bit byte addressed by register RA. The status of a switch is one when the switch is set. The four low order bits are set to one. This instruction is restricted to Monitor level code.

INSTRUCTION SUMMARY

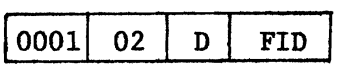
The following diagrams show the formats for each of the instructions. The diagrams are listed in order of increasing primary operation code (first four bits). The operation code is shown as a binary value. The second and third portions of the operation code (if they appear) are labeled 02 and 03 respectively.



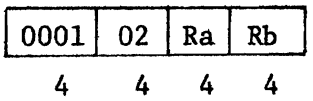
02	INSTRUCTION
0000	No Operation
0001	Test Interrupts
1000	Halt the CPU
1010	Resume Virtual Process
1011	Start Virtual Process
1100	Branch to Absolute Address
1101	Branch and Stack Location, to Absolute Address



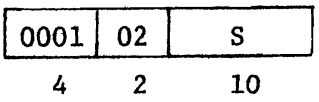
02	INSTRUCTION
0010	Branch Indirect (External)
0011	Branch and Stack Location Indirect (External)
0100	Return
0101	Return without Trace



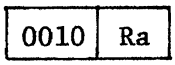
02	INSTRUCTION
0000	Branch (External)
0001	Branch and Stack Location (External)



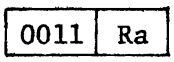
02	INSTRUCTION
0110	Move Address Register to Address Register
0111	Exchange Address Registers



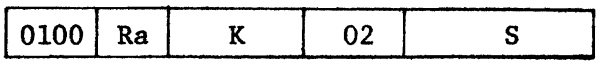
02	INSTRUCTION
10	Branch and Stack Location (Internal)
11	Branch (Internal)



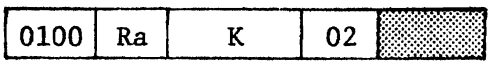
INSTRUCTION
Decrement Address Register



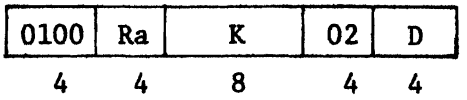
INSTRUCTION
Increment Address Register



02	INSTRUCTION
000000	Branch Character Not Equal to Immediate
000001	Branch Character Less Than Immediate
000010	Branch Character Equal to Immediate
000011	Branch Character Less Than or Equal to Immediate

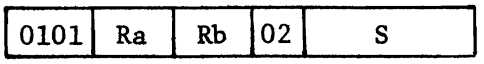


02	INSTRUCTION
0010	Move Immediate Character
0100	Increment and Store Immediate Character
0110	Flag the Address Register
0000	Output Byte
1001	Input Byte
1100	Or
1101	Exclusive Or
1110	And
1111	Shift

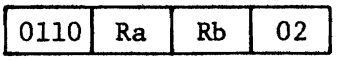


02	INSTRUCTION
0111	Monitor Call

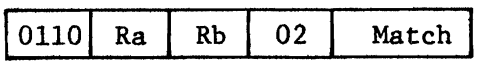
THIS PAGE INTENTIONALLY LEFT BLANK



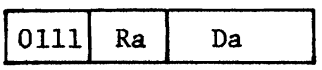
02	INSTRUCTION
00	Branch Character Not Equal
01	Branch Character Less Than
10	Branch Character Equal
11	Branch Character Less Than or Equal



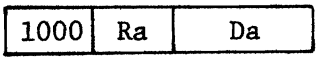
02	INSTRUCTION
0001	Increment Source Register and Move Character
0010	Increment Both Registers and Move Character
0011	Increment and Move String Under Address Control
0100	Increment and Move String Under Count Control
0101	Read Terminal Queue
0111	Exchange Bytes
1001	Move Byte
1010	Increment Destination Register and Move Character
1101	Write Terminal Queue



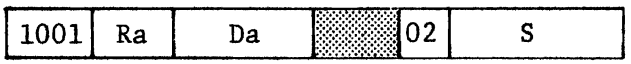
02	INSTRUCTION
0000	Increment and Move String Under Match Control
1000	Increment and Scan String Under Match Control



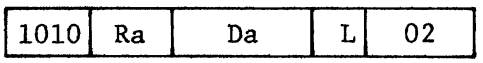
INSTRUCTION
Set Bit Off



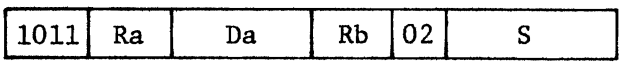
INSTRUCTION
Set Bit On



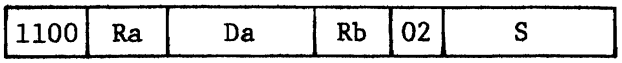
02	INSTRUCTION
00	Branch Bit On
01	Branch Bit Off



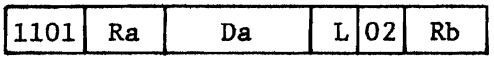
02	INSTRUCTION
000000	Store a Zero
000001	Store a One
000011	Add One to Storage
000101	Subtract One from Storage
001000	Negate Storage
010000	Multiply
010001	Divide
010011	Add to Accumulator
010101	Subtract from Accumulator
011000	Load Accumulator
011001	Store Accumulator



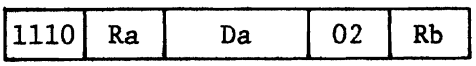
02	INSTRUCTION
00	Branch Relative Character Not Equal
01	Branch Relative Character Less Than
10	Branch Relative Character Equal
11	Branch Relative Character Less Than or Equal



02	INSTRUCTION
00	Branch Addresses Equal
10	Branch Addresses Not Equal



02	INSTRUCTION
00	Move Byte to Relative Byte (L equal to zero)
00	Decimal to Binary (L not equal to zero)
01	Move Offset Byte to Byte
10	Hexadecimal to Binary
11	Binary to Hexadecimal



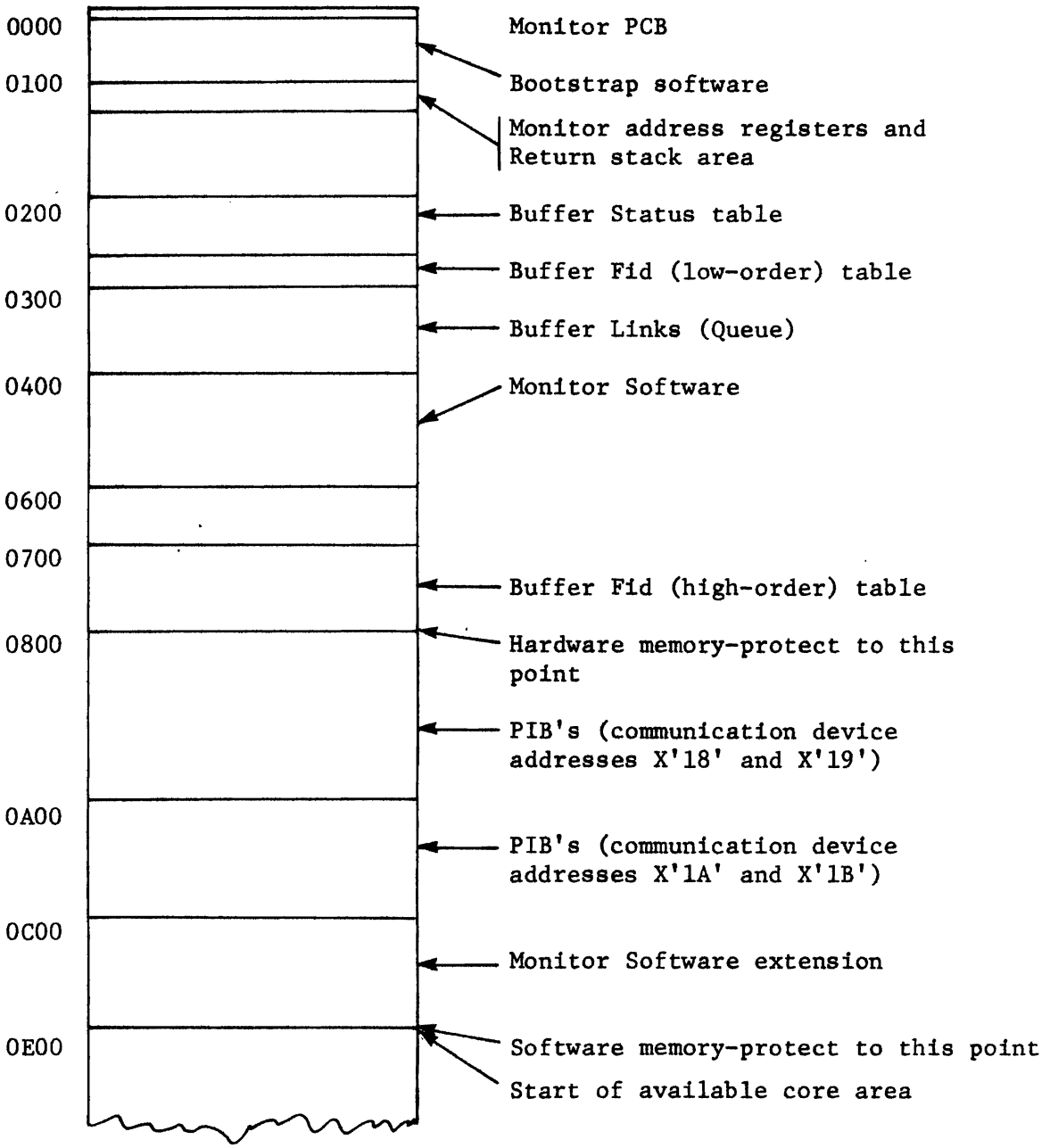
02	INSTRUCTION
0011	Load Effective Address (half word)
0100	Subtract from Address Register
0101	Add to Address Register
0111	Load Effective Address (full word)
1011	Load Effective Address (double word)
1100	Load Absolute Address Difference
1101	Store Address Register
1110	Load Address Register
1111	Load Effective Address (triple word)

02	INSTRUCTION
00	Move Storage to Storage
10	Add Storage to Storage
11	Subtract Storage from Storage

1111	Ra	Da	L	02	Rb	Da	03	S
4	4	8	2	2	4	8	6	10

02	03	INSTRUCTION
01	010100	Compare and Branch Not Equal
01	010101	Compare and Branch Less Than
01	010110	Compare and Branch Equal
01	010111	Compare and Branch Less Than or Equal
01	011100	Subtract and Branch Not Equal
01	011111	Subtract and Branch Less Than Equal or Equal
01	011101	Subtract and Branch Less Than Equal
01	011110	Subtract and Branch Equal

CORE MAP


Core Address
0000
0100
0200
0300
0400
0600
0700
0800
0A00
0C00
0E00
Monitor PCB
Bootstrap software
Monitor address registers and Return stack area
Buffer Status table
Buffer Fid (low-order) table
Buffer Links (Queue)
Monitor Software
Buffer Fid (high-order) table
Hardware memory-protect to this point
PIB's (communication device addresses X'18' and X'19')
PIB's (communication device addresses X'1A' and X'1B')
Monitor Software extension
Software memory-protect to this point
Start of available core area

Peripheral I/O: Device Orders

Order Number	Operation	Description
0	Data Transfer	A data byte will be transferred between the addressed device and the processor. Direction of the transfer will depend on whether the instruction is an input or an output.
1	Status/Function	A status byte will be input from the addressed device or a function byte will be output to the addressed device, depending on whether the instruction is an input or an output.
2	Block Input/INT	The addressed device will start a concurrent block input to memory and will generate an external interrupt at the conclusion of the transfer unless the interrupt has been subsequently disarmed. This order should be sent by an output instruction.
3	Arm Interrupt	Permits the addressed device to make an external interrupt request upon the satisfaction of an interrupt condition. This order should be sent by an output instruction.
4	Disconnect	The block transfer in progress by the addressed device is stop ed and end of block interrupt will occur unless the interrupt has been disarmed. This order should be sent by an output instruction.
5	Disarm Interrupt	Inhibits the addressed device from marking an external interrupt request under any condition. This order should be sent by an output instruction.
6	Block Output/INT	The addressed device will start a concurrent block output from memory and will generate an external interrupt at the conclusion of the transfer unless the interrupt has been subsequently disarmed. This order should be sent by an output instruction.
7	Unassigned	This order, if assigned, may perform any required function as interpreted by the individual interface. If a byte transfer is desired the order may be sent by an input or an output instruction.

Section XVI

REALITY ASSEMBLY LANGUAGE (REAL)

INTRODUCTION

The Reality Assembler (REAL) translates source statements into Reality CPU machine language equivalent. The source file, or "mode" is an item in any file defined on the database. The mode is assembled in place; that is, at the conclusion of the assembly process, the item contains both the original source statements, as well as the generated object code. The same mode can then be used to generate a formatted listing (using the MLIST verb) or can be loaded for execution (using the MLOAD verb).

Source Language

The source language accepted by the REAL assembler is a sequence of symbolic statements, one statement per source-item line. Each statement consists of a label field, an operation (or op-code) field, an operand field, and a comment field.

Label Field

The label field begins in column one of the source statement, and is terminated by the first blank or comma; there is no limit on its length. If the character "*" appears in the first column, the entire statement is treated as a comment, and is ignored by the assembler. The reserved characters * + - ' = are the only ones that may not appear in the label field. An entry in this field is optional for all except a few opcodes. A label may not begin with a numeric character.

Operation Field

The operation field begins following the label field and consists of a legal REAL op-code. Op-codes are pre-defined in the permanent op-code symbol file OSYM and consist of one or more alpha characters. Op-codes may be mnemonics for; Reality machine language instruction (eg. B for BRANCH) macros, which may assemble into several Reality machine language instructions (eg. MBD for MOVE BINARY to DECIMAL), or assembler pseudo-ops (eg. ORG for ORIGIN). Additionally, users may define new mnemonics or "macros" which expand into several Reality machine instructions. This may be done by creating new entries in the OSYM file.

Operand Field

Operand field entries are optional, and vary in number according to the needs of the associated REAL op-code. Entries are separated by commas and cannot contain embedded blanks (except for character string literals enclosed by single quotes). The operand field is terminated by the first blank encountered. The characters + - ' * have special meaning in this field.

Operand Field Expressions

Entries in the operand field may be a symbol, or a constant. A symbol is a string of characters that is either defined by a single label-field entry in the mode, or is an entry in the pre-defined permanent symbol file (PSYM). A constant may be one of the following forms:

*	-	Defines current value of the assembler location counter.
n	-	(n decimal) - A decimal constant.
X'h'	-	(h hexadecimal) - A hexadecimal constant.
C'text'	-	Character string; any characters, including blanks and commas, may appear as part of "text"; a sequence of two single quotes ('') is used to represent one single quote in the text.

Arithmetic operators (+,-) may be used to combine two or more constants.

Comment Field

Any commentary information preceded by a blank may follow the operand field entries.

"Argument" Field

For the purposes of the remainder of this documentation, the label field entry, op-code field entry, and operand field entries will be referred to as "argument field" (AF) 0, 1, 2, etc.

Calling the Assembler

The assembler is called by the statement:

```
AS file-name item-name (Q)
```

which will assemble the item in the file specified. The optional specification "(Q)" specifies that error lines are not to be listed at the end of the assembly.

As the assembler processes, it will output an asterisk (*) as every ten source statements are assembled. At the end of pass-1 a new line is started and an asterisk is printed for each ten statements reassembled.

Listing Output

The listing processor may be called by the statement:

```
MLIST file-name item-name (options)
```

Options are separated by commas:

- P — Routes output to the line-printer.
- M — Prints macro-expansions of source statements.
- E — Prints error lines only; also suppresses the pagination, and enters EDIT at the end of the listing.
- Z — Inhibits EDIT entry when E option is specified.
- n-m — Restricts listing to line numbers n through m inclusive.

The listing is output with a statement number, location counter, object code and source code with the label, op-code, operand and comment fields aligned. A page heading is output at the top of each new page.

Errors, if any, appear in the location counter/object code area; macro expansions appear as source code if not suppressed, with the operation codes prefixed by a plus sign (+).

Loading

The assembled mode may be loaded into the frame specified by the FRAME op-code by using the statement:

```
MLOAD file-name item-name
```

If the load is successful, the message:

```
[216] 'item-name' LOADED ON FRAME # n
size = m (DEC), h (HEX)
```

is returned.

```
:MLIST SYSGEN-FILE ABSL (1-25,M)  r)

MODE: ABSL                                  PAGE 1       17:26  26 OCT 1973

001                          FRAME 047
    001 7FF0002F             +FRM: 047
                             +ORG  1
002                *SYSTEM MODE
003                          B     !ABSD              0
    001 1C04                 +B:   !ABSD
004                          B     !ABSL              1
    003 1C3C                 +B:   !ABSL
005                          B     !SEGMNT            2
    005 1D80                 +B:   !SEGMNT
006                DETACH    DEFM  14,TAPEIO
007                GETWS     DEFM  3,WSPACES
008                *
009 007 100031     !ABSD     ENT   ABSD
010 00A 0000002F   ABSLFID   DTLY  47                 FID OF THIS PROGRAMME
011 00E 00000022   OF1FID    DTLY  34                 FID OF OF1 PROGRAMME
012                XUSER     DEFT  15,USER            TALLY USER RELATIVE TO REGISTER 15
013 012 0000       PIBADDR   ADDR  0,X'80FFFFFC'
    014 80FFFFC
014                *
015                * ERROR ENTRY POINT *
016 018 E062E4     TPERR     MOV   JSBAG,IS
017 01B 111006               BSL   CRLFPRINT
018 01E 07                   TEXT  X'07',C'TAPE FORMAT ERROR'.X'FDFF'
    01F 54415045
    023 20464F52
    027 4D415420
    02B 4552524F
    02F 52
    030 FDFF
019                          MII   IS,OB,OBSIZE       COPY TAPE DATA & PRINT
    032 A05B58               +LOADT OBSIZE
    035 64B4                 +MIITRR IS,OB
020 037 112006               BSL   WRTLIN
021 03A 08                   TEXT  X'08'              BLOW-UP HERE
022                          DEC   RSCWA,4            BACKUP STACK
    03B F0C171E7             +DEC  RSCWA,=T4
023                          B     NXTAB              CONTINUE TO NEXT TAPE SEGMENT
    03F 1CAC                 +B:   NXTAB
024                *
025                **********************************************
EOF:
```

The mode will not load correctly if its size exceeds 512 bytes, or if a FRAME statement is not the first statement assembled in the mode. In either case, a message will be returned indicating the error.

Note: MLDAD can not load itself. It must be loaded by the XLOAD verb by the following message:

XLOAD SYSTEM-MODES LOADER

TCL-II Cross Reference Capability

Cross-Index Verb

The TCL-II CROSS-INDEX Verb first clears the CSYM file then updates it by item with the external references of that item. The CROSS-INDEX Verb requires the following format:

CROSS-INDEX FILE-NAME ITEM-LIST (OPTIONS)

EXAMPLE--

CROSS-INDEX MODES *

Would cross index all items of the modes file. An example of what a portion of the CSYM file might look like after using the CROSS-INDEX Verb follows. Notice that the item called DLOAD has one external reference to LISTFLAG, two external references to RMBIT, etc.

```
    DLOAD
ØØ1 LISTFLAG Ø1 RMBIT Ø2
ØØ2 CH8 Ø1
ØØ3 NNCF Ø2
ØØ4 CTR1 Ø2 CTR2 Ø1 MODULO Ø7 OBSIZE Ø1 RSCWA Ø1 SEPAR 1Ø TØ Ø1 T4 Ø3
ØØ5 BASE Ø8 DØ Ø1 OVRFLW Ø1 R15FID Ø1 RECORD Ø5
ØØ6 BMSBEG Ø1 CSBEG Ø1 ISBEG Ø2 OBBEG Ø1 S2 Ø2
ØØ7 CS Ø6 IS 21 OB Ø5 R14 Ø3 R15 Ø6 TS Ø1
ØØ8 ABSL Ø2 CRLFPRINT Ø1 CVDR15 Ø3 CVTNIS Ø2 GETBLK Ø1 LINK Ø1 MBDNSUB Ø3
    UPDITM Ø1 WRTLIN Ø2
ØØ9 AM Ø2
Ø1Ø

    WRAPUP-III
ØØ1 INHIBIT Ø4 RMBIT Ø4 SB6Ø Ø3 SB61 Ø6
ØØ2
ØØ3
ØØ4 CTRØ 13 CTR1 Ø3 EMOD Ø1 MMOD Ø1 MODULO Ø1 OBSIZE Ø2 TØ Ø2 T6 Ø3 USER Ø4
ØØ5 BASE Ø3 EBASE Ø3 MBASE Ø1
ØØ6 BMSBEG Ø3 OBBEG Ø2 OBEND Ø6 SR4 Ø2 SYSR1 Ø2
ØØ7 AF Ø8 BMS Ø4 IR 25 OB 3Ø R14 Ø4 R15 Ø3 TS 14
ØØ8 DATE Ø1 GBMS Ø1 RETIX Ø2 TIME Ø1 WRITOB Ø1 WRTLIN Ø8
ØØ9 AM Ø5 MMOD Ø1 SM Ø4
Ø1Ø
```

```
    DUMP-I
ØØ1 GBIT Ø6
ØØ2
ØØ3 NNCF Ø1 NPCF Ø1
ØØ4 C1 Ø8 C2 Ø2 C6 Ø7 C7 Ø6 REJCTR Ø1 TØ Ø1 T4 Ø1
ØØ5 D3 Ø7 D4 Ø5 FRMP Ø1 LINQUE Ø7 RECORD Ø7
ØØ6 IREND Ø6 S1 Ø3 S3 Ø8
ØØ7 IR 18 IS 15 OB 17 R15 14 IS Ø5
ØØ8 CVTNIS Ø4 DUMP-II Ø1 MBDNSUB Ø1 MBDSUB Ø7 MD99 Ø1 MD999 Ø1 RDREC Ø4
    WRTLIN Ø5
ØØ9 LF Ø2
Ø1Ø

    PROC-I
ØØ1 PQFLG Ø4 SB1 Ø2 SB2 Ø1 SBIT Ø7 STKFLG Ø1
ØØ2 PRMPC Ø1
ØØ3
ØØ4 C1 Ø1 TØ Ø2
ØØ5
ØØ6 BMSBEG Ø1 IBBEG Ø1 IBEND Ø1 ISBEG Ø1 OSBEG Ø1 PBUFBEG Ø4 PQBEG Ø2
    PQCUR Ø2 PQEND Ø3 SR35 Ø1 SR4 Ø1 Ø2
ØØ7 CS Ø2 TB Ø7 IR 81 IS 11 OB Ø5 OS Ø1 R14 12 TS Ø4 UPD 12
ØØ8 CVTHIR Ø1 GETIBX Ø1 MD18 Ø1 MD995 Ø1 MD999 Ø1 PROC-II Ø4 PROC-III Ø4
    WRAPUP-I Ø1 WRITOB Ø1 WRTLIN
ØØ9 AM 12 CR Ø1 SM Ø2 VM Ø1
Ø1Ø
```

XREF Verb

The TCL-II XREF Verb uses the CSYM file as updated by the Cross-Index Verb for input. XREF then updates the XSYM file in the opposite order of the CSYM file. The XREF Verb requires the following format:

```
XREF FILE-NAME ITEM-LIST (OPTIONS)
```

EXAMPLE--

```
XREF CSYM *
```

Would cross reference all items of the CSYM file. An example of what a portion of the XSYM file might look like after using the XREF Verb follows. Notice that the item called T4 was externally referenced by DLOAD, DUMP-I, and SYSTEM-SUBS-II.

```
    MBIT
ØØ1 ASTAT

    GEN
ØØ1 ASTAT

    CVDR15
ØØ1 DLOAD
```

```
    LFDLY
ØØ1 SYSTEM-SUBS-II

    PAGSIZE
ØØ1 SYSTEM-SUBS-II

    T4
ØØ1 DLOAD DUMP-I SYSTEM-SUBS-II

    D5
ØØ1 SYSTEM-SUBS-II

    CTR2
ØØ1 DLOAD AID1

    UPDITM
ØØ1 XREF ILOAD T-LOAD DLOAD EDIT-IV/P&A

    C1
ØØ1 XREF ILOAD ASTAT T-LOAD DUMP-I PROC-I EDIT-IV/P&A EDIT-I

    SR13
ØØ1 EDIT-I

    SR7
ØØ1 EDIT-I

    MBASE
ØØ1 XREF WRAPUP-III TI WII

    DB1
ØØ1 WII

    B15
ØØ1 MONITOR/295Ø,1ØMB

    R7WA
ØØ1 DISK-DIAGNOSTIC/2314

    T5
```

The sort verb may be used after performing X-REF to produce a sorted output.

EXAMPLE--

```
    SORT XSYM REFERENCES NONCOL (P)
```

Would produce an alphabetical non-columnar listing on the line printer. References and noncol are attribute definitions in the XSYM dictionary.

The following is an example of a partial listing.

```
XSYM : ABIT
REFERENCES   EDIT-I

XSYM : ABSL
REFERENCES   DLOAD

XSYM : ACF
REFERENCES   WII

XSYM : ADDLAB
REFERENCES   ASTAT

XSYM : AF
REFERENCES   ASTAT          WRAP-III          EDIT-I

XSYM : AFBEG
REFERENCES   ASTAT          EDIT-I

XSYM : AFEND
REFERENCES   ASTAT

XSYM : ALIGN
REFERENCES   ASTAT

XSYM : AM
REFERENCES   XREF           ASTAT             T-LOAD
             DLOAD          WRAPUP-III        PROC-I
             AID1           TI                WII
```

XREF Proc

The XREF Proc will perform the following functions:

1. Clear the XSYM file.

2. Use the XREF verb to update the XSYM file.

3. Alphabetically sort the XSYM file and output the results to either the user's terminal or to the system line printer.

The XREF Proc requires the following format:

```
XREF FILE-NAME ITEM-LIST (OPTIONS)
```

EXAMPLE--

```
XREF CSYM * (P)
```

would cross reference all items of the CSYM file and would list the results in alphabetical order on the line printer.

The following is an example of a partial listing.

```
                                                    11:2Ø  2Ø DEC 1973

XSYM......REFERENCES........

ABIT        EDIT-I

ABSL        DLOAD

ACF         WII

ADDLAB      ASTAT

AF          ASTAT
            WRAPUP-III
            EDIT-I

AFBEG       ASTAT
            EDIT-I

AFEND       ASTAT

ALIGN       ASTAT

AM          XREF
            ASTAT
            T-LOAD
            DLOAD
            WRAPUP-III
            PROC-I
            AID1
            TI
            WII

ATTOVF      WII

B15         MONITOR/295Ø,1ØMB

B4          MONITOR/295Ø,1ØMB

BASE        XREF
            ILOAD
            ASTAT
            T-LOAD
            DLOAD
            WRAPUP-III
            TI
            WII
```

Operand Conventions

In defining the REAL op-codes the following set of symbolic operands are used.

a	ABS	An absolute core-address reference
b	BIT	A bit addressed relatively via a base address register and a bit displacement.
c	CHARACTER	A byte addressed relatively via a base address register and an 8-bit byte displacement.
d	DOUBLE-WORD	A 4-byte field addressed relatively via a base register and a 16-bit word displacement.
h	HALF-WORD	A 1-byte field addressed relatively via a base register and an 8-bit byte displacement.
l	LABEL	A label definition local to the current program frame.
m	MODE ID	A 16-bit modal identification, comprised of a 4-bit entry point and a 12-bit frame number. The implied location is in the frame defined by the low-order 12-bits of "m", offset from the frame-beginning by twice the entry-point value.
n	LITERAL	A literal or immediate value. The size of the assembled literal or value is dependent on the instruction in which the "n" is used.
r	ADDRESS-REGISTER	One of the sixteen Reality address registers (A/R's).
s	STORAGE-REGISTER	A 6-byte field (usually a storage-register, or S/R) relatively addressed via a base register and a 16-bit word displacement.
t	WORD	A 2-byte field relatively addressed via a base register and a 16-bit word displacement.

Note: The parenthesized footnotes in the following section are defined at the end of the section.

Character Instructions (Moves)

MCC	n,c	(1)	Move Character to Character; the byte (character) defined or addressed by operand-1 is moved to the location addressed by operand-2.
	n,r		
	n,s	(1)	
	c,c		
	c,r		
	c,s	(1)	
	r,c		
	r,r		
	r,s	(1)	
	s,c	(2)	
	s,r	(3)	
	s,s	(2)	
MCI	n,r		Move Character to Incrementing character; the byte (character) pointer operand-2 is incremented by one and the byte defined or addressed by operand-1 is moved to the location then addressed by operand-2.
	n,s	(1)	
	c,r		
	c,s	(1)	
	r,r		
	r,s		
	s,r	(3)	
	s,s	(2)	
MCI	n,r,n	(4)	Move Character Incrementing; the byte (character) pointer operand-2 is incremented by one and the byte defined by operand-1 is moved to the location then addressed by operand-2. This process continues until the number of bytes specified by operand-3 have been moved. At least one byte is always moved and if initially operand-3 = 0, 65,536 bytes will be moved.
	n,r,h	(4)	
	n,r,t	(4)	
	n,r,d	(4)	
MIC	r,c		Move Incrementing character to Character; the byte (character) pointer operand-1 is incremented by one and the byte then addressed by operand-1 is moved to the location addressed by operand-2.
	r,r		
	r,s	(1)	
	s,c	(2)	
	s,r	(3)	
	s,s	(2)	

MII	r,r		Move Incrementing character to Incrementing character; both byte pointers are incremented by one and the byte then addressed by operand-1 is moved to the location addressed by operand-2.
	r,s	(1)	
	s,r	(3)	
	s,s	(2)	
MII	r,r,n	(5)	Move Incrementing character to Incrementing character; both byte pointers are incremented by one and the byte addressed by operand-1 is moved to the location addressed by operand-2. This process is repeated until the number of bytes specified by n,h,t or d have been moved. h,t or d are not destroyed and if initially zero, no bytes are moved.
	r,r,h	(5)	
	r,r,t	(5)	
	r,r,d	(5)	
MII	r,r,r	(3)	Move Incrementing character to Incrementing character; both addressing-registers operand-1 and operand-2 are incremented by one and the byte then addressed by operand-1 is moved to the location addressed by operand-2. This process is repeated until the first addressing-register operand-1 matches the byte-pointer operand-3. If operand-1 = operand-3 on entry no movement takes place.
	r,r,s	(3)	
MIID	r,r,n		Both addressing-registers are incremented by one, and the byte addressed by addressing-register-1 is moved to the location addressed by addressing-register-2. The byte moved is then tested under the following masking condition where "n" is an 8-bit mask field:

Bit	Meaning	
0	True/False	
1	Match on:	X'FF'
2		X'FE'
3		X'FD'
4		X'FC'
5		SC0
6		SC1
7		SC2

Bit 0 is a true/false flag; is set, the move stops on a "match" condition (as defined by bits 1 through 7); if zero, the move stops on a "non-match". Bits 1 through 7 represent one character each; if any bit is set, the byte moved is compared to the

		character represented by the bit for a match. Bits 1 through 4 represent the special system delimiters SM (X'FF), AM (X'FE'), VM (X'FD'), and SVM (X'FC') respectively. Bits 5, 6, and 7 represent the contents of the scan character-registers SC0, SC1, and SC2 respectively. (Thus only three of the delimiters are variable. NOTE: Character-register SC0 may not contain the hex patterns X'00' or X'01'.
SCD	r,n	Scan characters to delimiter(s). The addressing-register is incremented till a "match" condition (see MIID instruction) as defined by the 8-bit mask field "n" is found.
MIIT	r,r	This instruction assumes that the lower half of the accumulator (T0) has an absolute byte count (up to 65535) defining the number of bytes to be moved (see MII op-code). If T0 is zero when the instruction is executed, no operation is performed. Otherwise, the addressing-registers are incremented by one, and the byte addressed by addressing-register-1 is moved to the location addressed by addressing-register-2, and T0 is decremented by one. This sequence is repeated till T0 reaches zero.
MIIR	r,r	This instruction assumes that address register R15 is setup to a location equal to or greater than that of addressing-register-1. (See MII op-code). If the addresses of addressing-register-1 and register R15 are equal, no operation is performed. Otherwise, the addressing-registers are incremented by one, and the byte addressed by addressing-register-1 is moved to the location addressed by addressing-register-2. This sequence is repeated till the addresses of addressing-register-1 and register R15 are equal.

XCC	c,c	(2)	Exchange Character with Character; the byte
	c,r	(3)	(addressed) by operand-1 is interchanged
	c,s	(2)	with the byte defined by operand-2.
	r,c	(1)	
	r,r		
	r,s	(1)	
	s,c	(2)	
	s,r	(3)	
	s,s	(2)	
OR	c,n	(3)	OR character; the byte (character) addressed
	r,n		by operand-1 is logically or'd with the
	s,n	(3)	8-bit immediate operand-2.
XOR	c,n	(3)	Exclusive OR character; the byte (character)
	r,n		addressed by operand-1 is exclusively or'd
	s,n	(3)	with the 8-bit immediate operand-2.
AND	c,n	(3)	AND character; the byte (character)
	r,n		addressed by operand-1 is logically and'd
	s,n	(3)	with the 8-bit immediate operand-2.

Character Instruction (Tests)

BCE	n,c,1	(1)	Branch Character Equal; the byte (character)
	n,r,1		defined or addressed by operand-1 is compared
	c,n,1	(3)	to the byte defined or addressed by operand-2.
	c,c,1		If the two bytes are equal, instruction
	c,r,1		execution branches to the location as defined
	r,n,1		by operand-3. Neither operand-1 nor
	r,c,1		operand-2 are altered. The arithmetic
	r,r,1		condition flag (ACF) is set on c,c,1 only.
BCU	(see BCE)		Branch Character Unequal; branch if characters are not equal.
BCL	(see BCE)		Branch Character Low; branch if operand-1 is less than operand-2.
BCLE	(see BCE)		Branch Character Less than or Equal; branch if operand-1 is less than or equal to operand-2.
BCH	(refer to BCE)		Branch Character High; branch if operand-1 is greater than operand-2.
BCHE	(refer to BCE)		Branch Character High or Equal; branch if operand-1 is greater than or equal to operand-2.

BCN	r,1	Branch if Character is Numeric; branch if the character addressed by the first operand is in the range 0-9, inclusive.
BCX	r,1	Branch if Character is hexadecimal; branch if the character addressed by the first operand is in the range 0-9 or A-F, inclusive.
BCA	r,1	Branch if Character is Alphabetic; branch if the character addressed by the first operand is in the range A-Z, inclusive.

Bit Instructions

SB	b	Set Bit; the bit addressed by the operand is set to an on condition (one).
ZB	b	Zero Bit; the bit addressed by the operand is set to an off condition (zero).
BBS	b,1	Branch Bit Set; the bit addressed by operand-1 is tested and if set (one) instruction execution branches to the location defined by operand-2.
BBZ	b,1	Branch Bit Zero; the bit addressed by operand-1 is tested and if not set (zero) instruction execution branches to the location defined by operand-2.

Data Movement and Arithmetic Instructions

All arithmetic is done on two's complement binary integers. All instructions in this section except the MOV set the arithmetic condition flag (ACF).

MOV	n,h	(6)	MOVe word to word; integer defined or addressed by integer-1 is moved to the location addressed by operand-2.
	n,t		
	n,d		
	h,h		
	h,t	(6)	
	h,d	(6)	
	t,h	(6)	
	t,t		
	t,d	(6)	
	d,h	(6)	
	d,t	(6)	
	d,d		
	b,b		

TST	h t d		Test the contents of the operand and set the arithmetic condition flags.
INC	h t d		INCrement by one; the integer defined by the operand is incremented by one
	h,n	(6)	INCrement word by word; the integer defined or addressed by operand-2 is added to the integer stored in the location addressed by operand-1 and the result is stored in the latter location.
	h,h		
	h,t	(6)	
	h,d	(6)	
	t,n		
	t,h	(6)	
	t,t		
	t,d	(6)	
	d,n		
	d,h	(6)	
	d,t	(6)	
	d,d		
DEC	h t d		DECrement by one; the integer defined by the operand is decremented by one.
	h,n	(6)	DECrement word by word; the integer defined or addressed by operand-2 is subtracted from the integer stored in the location addressed by operand-1 and the result is stored in the latter location.
	h,h		
	h,t	(6)	
	h,d	(6)	
	t,n		
	t,h	(6)	
	t,t		
	t,d	(6)	
	d,n		
	d,h	(6)	
	d,t	(6)	
	d,d		
ZERO	h t d		ZERO word; a zero is moved to the operand location defined by operand-1.
ONE	h t d		Set word ONE; an integer value of one is moved to the operand location defined by operand-1.
NEG	h t d		NEGate word; the integer defined by operand-1 is negated (two's complement).
LOAD	n h t d		LOAD to accumulator; the integer addressed by operand-1 is loaded into the 32-bit accumulator (D0). For half-word and word operands, the sign bit is extended.

STORE	h t d	STORE from accumulator; the contents of the 32-bit accumulator (D0) are stored into the location defined by operand-1. For half-word and word operands, the high order bits are lost.

THIS PAGE INTENTIONALLY LEFT BLANK

ADD	n h t d		ADD to accumulator; the integer addressed by operand-1 is added to the 32-bit accumulator (D0) with sign extension.
SUB	n h t d		SUB from accumulator; the integer addressed by operand-1 is subtracted from the 32-bit accumulator (D0) with sign extension.
MUL	n h t d		MULtiply to accumulator; the integer addressed by operand-1 is multiplied by the contents of the 32-bit accumulator (D0). The resulting product is stored in the 64-bit accumulator extension (D1,D0), as a 63 bit number and a duplicated sign bit.
DIV	n h t d		DIVide into the accumulator; the integer addressed by operand-1 is divided into the 32-bit accumulator (D0). The answer is stored in D0 and the integer remainder is stored into the accumulator extension (D1).

Register Instructions

MOV	r,r r,s s,r s,s		MOVe register to register; the address or storage register operand-1 is moved into the address or storage register operand-2.
XRR	r,r r,s s,r s,s	 (1) (2) (1)	eXchange Register with Register; the address or storage register operand-1 is exchanged with the address or storage register operand-2.
INC	r s		INCrement register; the address or storage register operand-1 is incremented by one.
INC	r,n r,h r,t r,d s,n s,h s,t s,d		INCrement register by count; the address or storage register operand-1 is incremented by the integer stored at the location addressed by operand-2.

DEC	r s		DECrement register; the address or storage register operand-1 is decremented by one.
DEC	r,n r,h r,t r,d s,n s,h s,t s,d		DECrement register by count; the address or storage register operand-1 is decremented by the integer stored at the location addressed by operand-2.
LAD	r,r r,s s,r s,s	(7) (1)	Load Absolute Difference; the absolute difference in bytes (characters) between the byte pointer operand-1 and the byte pointer operand-2 is computed and stored into the lower half of the accumulator (T0). Please see special note following Branch Register Equal/Unequal instructions.
SRA	r,c r,h r,t r,d r,s r,l		Set Register to Address; the byte pointer operand-1 is set pointing to the first byte of the functional element at the location addressed by operand-2.
FAR	r,n		Flag and Attach Register; the address-register operand-1 is attached and secondary processing as defined by the 8-bit literal n is performed:

0	0 - I/O busy for buffer
1	not used
2	0 - buffer core-locked
3	0 - write-required
4	not used
5	Set-up R15 to 1st byte unlinked; old buffer status in R15DSP
6	Change buffer FID
7	0 - OR n with status, 1 - AND n with status

In normal execution only n2 is effective; the remainder of the functions can only be evoked in the "monitor mode."

BE BU	r,r,1 r,s,1 s,r,1	(7)	Branch Register Equal/Unequal; the address of the byte pointer operand-1 is compared to the address of the byte pointer operand-2. The branch is taken appropriately. If the FID's of the registers are unequal, it is assumed that the affected frames are contiguously linked and the address computation is made on that basis; further, the difference in the FID's is assumed to be less than or equal to thirty-two (32); therefore the instruction execution may prove incorrect if 1) The FID's are unequal and not contiguously linked or 2) One of the registers is in an unlinked format, and the other is not.
BE BU	s,s,1		Branch Register Equal/Unequal; the 6-byte storage register operand-1 is arithmetically compared to the storage register operand-2 and the branch is made accordingly.

Data Comparison Instructions

BE	n,h,1 n,t,1 n,d,1 h,n,1 h,h,1 h,t,1 h,d,1 t,n,1 t,h,1 t,t,1 t,d,1 d,n,1 d,h,1 d,t,1 d,d,1	(6) (6) (6) (6) (6) (6) (6) (6) 	Branch word Equal; the integer stored in the word addressed by operand-1 is compared arithmetically (2's complement) to the integer stored in the word addressed by operand-2. If an equal comparison is made, instruction branches to the location defined by operand-3.
BU	(see BE)		Branch word Unequal; branch if words are unequal.
BL	(see BE)		Branch word Low; branch if operand-1 is less than operand-2.
BLE	(see BE)		Branch word Low or Equal; branch if operand-1 is less than or equal to operand-2.

BH	(see BE)	Branch word High; branch if operand-1 is greater than operand-2.
BHE	(see BE)	Branch word High Equal; branch if operand-1 is greater than or equal to operand-2.
BDZ	h,h,1 t,n,1 t,t,1 d,n,1 d,d,1	Branch on Decrementing word Zero; the word at the location addressed by operand-1 is decremented by the integer at the location addressed by operand-2. If the result is zero, instruction branches to the location defined by operand-3.
BDNZ	(see BDZ)	Branch on Decrementing word not Zero; same as BDZ but branch on result not zero.
BDLZ	(see BDZ)	Branch on Decrementing word Less than Zero; same as BDZ but branch on result less than zero.
BDLEZ	(see BDZ)	Branch on Decrementing word Less than or Equal to Zero; same as BDZ but branch on result less than or equal to zero.
BDZ	t,1 d,1	Branch on Decrementing word Zero; same as BDZ above but decrement by one.
BDNZ	t,1 d,1	Branch on Decrementing word not Zero; same as BDNZ above but decrement by one.
BDLZ	t,1 d,1	Branch on Decrementing word Less than Zero; same as BDLZ above but decrement by one.
BDLEZ	t,1 d,1	Branch on Decrementing word Less than or Equal to Zero; same as BDLEZ above but decrement by one.

All of the above data comparison instructions set the arithmetic condition flags.

Translate Instructions

MBD MBDN	h,r t,r d,r n,h,r n,t,r n,d,r	Move Binary word to Decimal characters; This macro generates a call to the subroutine MBDSUB (MBD) or MBDNSUB (MBDN), which converts from a binary integer at the location addressed by operand-1 to a string of decimal ASCII characters, stored beginning from the location addressed by the byte-pointer operand-2 plus one.

			The following elements are used by the subroutine and macro D0; D1; D2; T4; T5; R14; R15. A minus sign will precede the converted value if it was negative; at the conclusion of the instruction, the byte pointer operand-2 addresses the last converted byte. MBD deletes leading zeros, but converts at least one character; MBDN converts at least "n" characters, padded with leading zeros if necessary.
MDB	r,t r,d		Move Decimal character to Binary word; ASCII decimal to binary conversion. The word at the location addressed by operand-2 is multiplied by 10, and a value (as defined above for the MXT instruction) from the byte addressed by the addressing register is added to it. The arithmetic condition flags are <u>not</u> reset, and arithmetic overflow cannot be detected.
MBX	h,r t,r d,r		Move Binary word to heXadecimal characters; Binary to ASCII hex conversion. This instruction assumes that the least significant byte of the accumulator (H0) has a parameter (see MBX/MBXN macro). Bits 3-0 contain a digit count, specifying the maximum number of ASCII digits to be converted. As each digit is converted, the addressing register is incremented by one, and the converted ASCII character is stored in the location addressed by the addressing register. The format of H0 at the conclusion of this instruction is unpredictable. If the digit count in H0 exceeds the field defined by operand-1, no operation is performed.
MBX MBXN	n,h,r n,t,r n,d,r	(9)	Move Binary word to heXadecimal characters; This macro expands as a LOAD of the first operand (MBX) or the first operand +X'80' (MBXN), and an primitive. The MBX macro, therefore, causes conversion from binary to ASCII hex, with only significant digits (to a maximum of "n") converted. The MBXN macro causes conversion as above, but always converts "n" digits, with leading zeros if necessary. The addressing register defined by the third operand is incremented before each byte converted.

MXB	r,h r,t r,d	Move heXadecimal characters to Binary word; ASCII hex to binary conversion The field defined by operand-2 is shifted left 4 bits, and the value defined below, from the byte addressed by the addressing register, is added to the field: The 4-bit value from bits 3-0 of the byte (bits numbered right to left), plus nine times bit 5. The arithmetic condition flags are <u>not</u> reset by this instruction, and arithmetic overflow cannot be detected.

Execution Transfer Instructions

B	l a	Branch; branch to location defined, in the current frame, defined by label "l", or the absolute core address "a". The branch to an absolute core-address is a privileged instruction executable only at the monitor-level.
BSL	l m a	Branch and Stack Location; Subroutine call to mode defined by mode-ID "m", local label "l", or absolute core-address "a". (The BSL to an absolute core-address is a privileged instruction executable at the monitor-level only). The location-1 of the instruction following the BSL is saved in the return stack, and the next instruction executed is that defined by the operand. The return stack level is increased by one; if the call causes the return stack level to exceed its maximum value, the stack pointers are reset to the beginning and a trap to the DEBUG mode is executed.
BSLI		Branch and Stack Location Indirect; Subroutine call indirect; this instruction assumes that the lower half of the accumulator, T0 contains a mode-ID (see BSL* macro). The 16-bit mode-ID contained in T0 defines the location of the next instruction that is to be executed, after the location-1 of the instruction following the TCI is saved in the return stack.

RTN			ReTurN; Return to subroutine called. The last entry in the return stack defines the location of the next instruction to be executed; the return stack level is decremented by one. If the return stack is empty, a trap to the DEBUG mode is executed. A return instruction from a subroutine called via a local call, or an absolute core-address call, will return within the current 512-byte frame only.
ENT	m		ExterNal Transfer; Branch to location defined by mode-ID"m".
ENTI			ExterNal Transfer Indirect; Enter mode indirect: this instruction assumes that T0 contains a 16-bit mode-ID (see ENT* macro), which defines the next instruction to be executed.
BSL*	h t d	(8)	Branch and Stack Location indirect; subroutine call to mode defined by the mode-ID contained in the word addressed by operand-1. The 16 bit mode-ID is loaded into the accumulator, and a BSLI instruction is executed.
ENT*	h t d	(8)	ExterNal Transfer indirect; branch to external location defined by the mode-ID contained in the word addressed by operand-1. The 16 bit mode-ID is loaded into the accumulator, and an ENTI instruction is executed.

I/O and Control Instruction

IOI	r,n_1,n_2		I/O Instruction Input; this instruction is used to control input from peripheral devices whose device addresses are in the range 0 through X'F' (15). This instruction causes an MCAL instruction to entry point 8 in the Monitor. Register r points to the start of the input buffer; n_1 is a 3-bit order code; n_2 is a 4-bit device address. Refer to Reality CPU, peripheral I/O for details.

IOO	r,n_1,n_2	I/O instruction Output; as above this instruction output to peripheral devices.
READ	r	A byte from the byte-I/O buffer in the PIB is stored at the location addressed by the addressing register. If the buffer is empty, or if there is data in the byte I/O) buffer yet to be output to the byte I/O device, the process executing the READ instruction will enter a quiescent state till data from the byte input device causes a re-activation.
WRITE	r	The byte addressed by the addressing register is moved into the byte I/O buffer of the PIB. If the buffer is empty, the byte is also output immediately to the byte I/O device. If the buffer is full, the process executing the write will enter a quiescent state till the byte output device has accepted the data from the buffer, and causes a re-activation. Execution of this instruction causes a loss of any input data in the byte I/O buffer, and inhibits any further data input from the byte I/O device.
MCAL	r,n_1,n_2	Monitor call to entry point "n_2" ($7 < n_2 < 16$). The word address of the addressing register; the 8-bit address of the addressing register; the 8-bit mask n_1; and the location of the PCB are passed, as parameters to the monitor, in the PIB.
RQM		Process releases the remainder of its time quantum to the monitor. Equivalent to: MCAL 0,0,9.
IB OB	r,n	Input/Output byte instruction. Refer to Reality CPU, peripheral I/O for details. The byte defined by the mask "n" is output as a control byte, and a data byte is input (IB) and stored at the location addressed by the addressing register, or output (OB) from the location addressed by the addressing register. These instructions are allowable in monitor mode only.

NOP		No OPeration is performed by this instruction.
HALT		This instruction halts the CPU and is executable in the Monitor mode only.
HLD	r	Halts the CPU and gates the eight-bit literal addressed by register r to the A bus, where it can be displayed in the least significant indicator lamps of the system panel by depressing the Data select switch. Executable in monitor mode only.
TEXT X'01'		Tests internal and external interrupts and traps to the appropriate monitor location if any interrupts are pending. Executable in monitor mode only.
ECS	r	The status of the eight low-order console command switches is placed in the byte addressed by register r. Executable in monitor mode only.
ESS	r	The status of the four console sense switches is placed in the four most significant bits of the byte addressed by register r.
SVP		Start virtual Process (Monitor level only) The 2-byte FID located at absolute core address X'127', X'128' (R4-FID of monitor) is treated as a PCB-FID, and the buffer-pool searched for a match. If found, register zero in the PCB is setup in an "attached" format, and the attachment process for register one (user program-mode register) is started. If not found, a frame fault request on the PCB-FID is stacked, and the monitor is re-entered.
RVP		Return to Virtual Process (monitor level only) should be executed when a trap to the monitor due to an external interrupt by devices 0-X'15' has caused a monitor trap. Selects primary file registers of the 1600 computer and resumes execution of the virtual Process. If this instruction is executed when the system is not in an interrupt-handling mode, no operation takes place.

Assembler Directives

1	ADDR	n,n	Defines the local symbol "1" as a storage register in unlinked format. The displacement is defined by the first operand. The FID is defined by the second operand.
1	AR	1 r n	Defines the local symbol "1" as an address register with a value defined by the operand.
1	CHR HTLY TLY DTLY SR	1 n	Defines the local symbol "1" (if present) as a character (CHR) half-word (HTLY), word (TLY), double-word (DTLY) or S/R (SR) respectively; object code of the appropriate length and value defined by the operand is assembled, except for the SR op code, which ignores the operand field.
	CMNT		Comment; the contents of this statement are treated as commentary, and ignored by the assembler. Note: A label field entry is allowable.
1	DEFA	a	Defines the local symbol "1" to be of type a.
1	DEFM	r,1 r,n n,1 n,n	Defines the local symbol "1" to be of type m; a mode-ID with entry point defined by the first operand and FID defined by the second operan .
1	DEFk	r,1 r,n n,1 n,n	Defines the local symbol "1" to be of type "k" (where k=b,c,d,h,l,s,t), with base register defined by the first operand and displacement defined by the second operand.
		r,*[string] n,*[string]	When the assembler location counter "*" is used as the second operand, an optional string can be used, with the following format: string = $n_2[\pm n_3]$ or string = $\pm n_3$ If n_2 is specified after the *, instructions referencing 1 will obtain a displacement (D field) appropriate for an operand length of n_2 bits. Values of n_2 = 1,8, and 16 are valid, with a default of n_2 = 8. If $+n_3$ is specified after the *n, the effective displacement will be adjusted n_3 bits, bytes or double-bytes, depending on whether n_2 = 1,8 or 16.

EXAMPLE --

```
         ORG     10
LABEL1   DEFT    1,*16
         STORE   LABEL1
```

produces the object code A10559 corresponding to the instruction:

opcode-1	register	D	L	opcode-2
1010	0001	00000101	01	011001

with a displacement (D field) of 5 words relative to the byte addressed by register 1.

EXAMPLE --

```
         ORG     1
LABEL2   DEFB    1,*1+7
         SB      LABEL2
```

produces the object code 810F corresponding to the instruction.

opcode	register	D
1000	0001	00001111

with a displacement of 15 bits relative to the byte addressed by register 1.

1	EQU	c h t d s 1 n	Equates the local label "1" to the symbol or literal value of the operand.
	FRAME	n	Must be the first assembled statement in a mode that is to be loaded; "n" defines the frame on which the object code is to be loaded.
	ORG	1 n	Resets the location counter to value defined by the operand. This statement may have a label field entry.
	SETAR	r	Causes all literals encountered from this point in the assembly to be defined as a displacement relative to register r. If no SETAR occurs, SETAR 1 is assumed.

TEXT	X'...' C'...'	Assembles binary equivalent of character strings (enclosed in quotes and preceded by a 'C') or hexadecimal values. Any number and combination of C and X literals separated by commas is permitted.

Address Register Usage

A storage operand is always referenced through an address register containing the byte address of the operand. For instructions with a D field, a displacement is added to the contents of the address register to form an effective address. The length of the operand is encoded in the L field of the instruction (see INSTRUCTION DESCRIPTIONS in Chapter XV).

For REAL instructions allowing an address register r in the operand field, the displacement relative to the register and the operand length can be specified using the following formats:

Format	Displacement Relative to Address Register n	Operand Length
Rn	0 bytes	1 byte
Rn;Bm	m bits	1 bit
Rn;Cm Rn;Hm	m bytes m bytes	1 byte
Rn;Tm	2*m bytes	2 bytes
Rn;Dm	4*m bytes	4 bytes
Rn;Sm	6*m bytes	6 bytes

EXAMPLE --

MCC	R0;C15,R15	Move low order byte of the Accumulator to the byte addressed by R15.

EXAMPLE --

SB	R5;B0	Set bit 0 of the byte addressed by R5.

EXAMPLE --

MOV	MBASE,R10;D4	Move double-word MBASE to the double-word starting 16 bytes past the byte addressed by R10.

REAL Instruction Side Effects

Many of the REAL op-codes use functional elements not specified as operands for execution. Those instructions are so footnoted in the previous listing and the following explanation of the various footnotes describes the state of these implied elements at the conclusion of instruction execution.

(1) R15 points to byte addressed by operand-2.

(2) R14 points to byte addressed by operand-1, R15 points to byte addressed by operand-2.

(3) R15 points to byte addressed by operand-1.

(4) R15 points one prior to last byte moved and T0 contains number of bytes moved into last frame.

(5) Contents of T0 are unpredictable.

(6) D0 contains the integer moved or compared.

(7) SYSR0 contains the byte pointer operand-1.

(8) T0 contains the 16-bit mode-ID; T1 is zero.

(9) H0 contains the number of digits converted into the last frame, if its high order bit (B0) is set; otherwise original value.

```
001                 ************************************************************************
002                 *
003                 * REAL INSTRUCTION REPERTOIRE
004                 *
005                 ************************************************************************
006                 *
007                 * DEFINE SYMBOLIC OPERANDS USED IN DEFINITIONS
008                 *
009                 B1          DEFB  1,11
010                 B2          DEFB  2,22
011                 C1          DEFC  1,11
012                 C2          DEFC  2,22
013                 H1          DEFH  1,11
014                 H2          DEFH  2,22
015                 T1          DEFT  1,11
016                 T2          DEFT  2,22
017                 D1          DEFD  1,11
018                 D2          DEFD  2,22
019                 S1          DEFS  1,11
020                 S2          DEFS  2,22
021                 *
022                 * DEFINE FUNCTIONAL ELEMENTS USED IN MACRO EXPANSIONS
023                 *
024                 T0          DEFT  0,7
025                 D0          DEFD  0,6
026                 A1          DEFA  X'1234'
027                 M1          DEFM  1,2
028                 *
029                 *
030                 *
031                 ************************************************************************
032                 *
033                 * CHARACTER OPERATIONS
034                 *
035                 ************************************************************************
```

```
036                 * MOVE CHARACTER TO CHARACTER
037                 LØØ       EQU   *
038 ØØ1 E2163F               MCC   C'A',C2          ONE CHARACTER MOVED
    ØØ4 4F4120
039 ØØ7 424120               MCC   C'A',R2
040 ØØA E216EF               MCC   C'A',S2
    ØØD 4F4120
041                 *
042 Ø1Ø F216Ø1ØB             MCC   C1,C2
043 Ø14 D1ØB12               MCC   C1,R2
044 Ø17 E216EF               MCC   C1,S2
    Ø1A D1ØB1F
045                 *
046 Ø1D D216Ø1               MCC   R1,C2
047 Ø2Ø 6219                 MCC   R1,R2
048 Ø22 E216EF               MCC   R1,S2
    Ø25 6F19
049                 *
050 Ø27 E2163E               MCC   S1,C2
    Ø2A E1ØBEF
    Ø2D 6EF9
051 Ø2F E1ØBEF               MCC   S1,R2
    Ø32 62F9
052 Ø34 E1ØBEE               MCC   S1,S2
    Ø37 E216EF
    Ø3A 6F39
053                 *
054                 * MOVE CHARACTER TO CHARACTER, INCREMENTING DESTINATION
055 Ø3C 424140               MCI   C'A',R2          ONE CHARACTER MOVED
056 Ø3F A21643               MCI   C'A',S2
    Ø42 E216EF
    Ø45 4F4120
057                 *
058 Ø48 32                   MCI   C1,R2
    Ø49 D1ØB12
```

```
059 04C E216EF          MCI   C1,S2
    04F 3F
    050 D10B1F
    053 E216DF
060                 *
061 056 621A            MCI   R1,R2
062 058 E216EF          MCI   R1,S2
    05B 6F1A
    05D E216DF
063                 *
064 060 E10BEF          MCI   S1,R2
    063 62FA
065 065 E10BEE          MCI   S1,S2
    068 E216EE
    068 6FEA
    06D E216DF
066                 *
067                 *
056 070 162F            MCI   C'A',R2,7       MOVE # CHARACTERS IN OPERAND 3
    072 424140
    075 A72E58
    078 6F24
069 07A A10B18          MCI   C'A',R2,H1
    07D A00685
    080 162F
    082 424140
    085 6F24
070 087 A10B58          MCI   C'A',R2,T1
    08A A00685
    08D 162F
    08F 424140
    092 6F24
071 094 A10B98          MCI   C'A',R2,D1
    097 A00685
    09A 162F
```

```
    09C 424140
    09F 6F24
072                 *
073                 * MOVE CHARACTER TO CHARACTER INCREMENTING SOURCE
074 0A1 31                  MIC   R1,C2             MOVE 1 CHARACTER
    0A2 D21601
075 0A5 6121                MIC   R1,R2
076 0A7 E216EF              MIC   R1,S2
    0AA 61F1
077                 *
078 0AC A10B43              MIC   S1,C2
    0AF E2163E
    0B2 E10BEF
    0B5 6EF9
079 0B7 E10BEF              MIC   S1,R2
    0BA 6F21
    0BC E10BDF
080 0BF E10BEF              MIC   S1,S2
    0C2 E216EE
    0C5 6FE1
    0C7 E10BDF
081                 *
082                 * MOVE CHARACTER TO CHARACTER, INC SOURCE AND DESTINATION
083 0CA 6122                MII   R1,R2             MOVE 1 CHARACTER
084 0CC E216EF              MII   R1,S2
    0CF 61F2
    0D1 E216DF
085                 *
086 0D4 E10BEF              MII   S1,R2
    0D7 6F22
    0D9 E10BDF
087 0DC E10BEE              MII   S1,S2
    0DF E216EF
    0E2 6EF2
    0E4 E10BDE
```

```
    0E7 E216DF
088                 *
089 0EA A72F58              MII   R1,R2,80          MOVE # CHARACTERS IN OPERAND 3
    0ED 6124
090 0EF A10B18              MII   R1,R2,H1
    0F2 6124
091 0F4 A10B58              MII   R1,R2,T1
    0F7 6124
092 0F9 A10B98              MII   R1,R2,D1
    0FC 6124
093                 *
094 0FE 163F                MII   R1,R2,R3          MOVE UNTIL 2ND OPERAND = 3RD OPERAND
    100 6123
095 102 E24FEF              MII   R1,R2,S3
    105 6123
096                 *
097                 * INSTRUCTIONS INCREMENTING SOURCE & DESTINATION REPEATEDLY
098 107 6120E0              MIID  R1,R2,X'E0'       MOVE CHAR TO CHAR THROUGH DELIMITER
099                 *
100 10A 6108A0              SCD   R1,X'A0'          SCAN CHARACTERS TO DELIMETER
101                 *
102 10D 6124                MIIT  R1,R2             MOVE NUMBER OF CHARS.  IN ACCUMULATOR
103                 *
104 10F 6123                MIIR  R1,R2             MOVE UNTIL R1 AND R15 ARE EQUAL
105                 *
106                 *EXCHANGE CHARACTER WITH CHARACTER
107 111 E10B3E              XCC   C1,C2
    114 E2163F
    117 6EF7
108 119 E10B3F              XCC   C1,R2
    11C 6F27
109 11E E10B3E              XCC   C1,S2
    121 E216EF
    124 6EF7
110                 *
```

```
111 126 E2163F              XCC   R1,C2
    129 6F17
112 12B 6127                XCC   R1,R2
113 12D E216EF              XCC   R1,S2
    130 61F7
114                 *
115 132 E10BEE              XCC   S1,C2
    135 E2163F
    138 6EF7
116 13A E10BEF              XCC   S1,R2
    13D 6F27
117 13F E10BEE              XCC   S1,S2
    142 E216EF
    145 6EF7
118                 *
119                 *LOGICAL OR CHARACTER WITH MASK
120 147 E10B3F              OR    C1,X'AB'
    14A 4FABC0
121 14D 41ABC0              OR    R1,X'AB'
122 150 E10BEF              OR    S1,X'AB'
    153 4FABC0
123                 *
124                 *EXCLUSIVE OR WITH MASK
125 156 E10B3F              XOR   C1,X'AB'
    159 4FABD0
126 15C 41ABD0              XOR   R1,X'AB'
127 15F E10BEF              XOR   S1,X'AB'
    162 4FABD0
128                 *
129                 *LOGICAL AND CHARACTER WITH MASK
130 165 E10B3F              AND   C1,X'AB'
    168 4FABE0
131 168 41ABE0              AND   R1,X'AB'
132 16E E10BEF              AND   S1,X'AB'
    171 4FABE0
```

```
133                 *
134                 *
135 174 41FEF0                 SHIFT X'FE',R1
136                 *
137                 *
138                 *
139                 *
140                 ****************************************
141                 *
142                 * CHARACTER INSTRUCTIONS (TESTS)
143                 *
144                 ****************************************
145                 *
146                 *
147                 *BRANCH CHARACTER EQUAL
148 177 E2163F                  BCE   C'A',C2,L1
    17A 4F410847
149 17E 42410843                BCE   C'A',R2,L1
150                 *
151 182 E10B3F
    185 4F41083C
152 189 F10B1216                BCE   C1,C2,L1
    18D 5836
153 18F B10B2832                BCE   C1,R2,L1
154                 *
155 193 4141082E                BCE   41,C'A',L1
156 197 B216182A                BCE   R1,C2,L1
157 19B 512827                  BCE   R1,R2,L1
158                 *
159                 *BRANCH CHARACTER UNEQUAL
160 19E E2163F                  BCU   C'A',C2,L1
    1A1 4F410020
161 1A5 4241001C                BCU   C'A',R2,L1
162 1A9 E10B3F                  BCU   C1,C'A',L1
    1AC 4F410015
```

```
163                     *
164 1B0 F10B1216                BCU   C1,C2,L1
    1B4 500F
165 1B6 B10B200B                BCU   C1,R2,L1
166                     *
167 1BA 41410007                BCU   R1,C'A',L1
168 1BE B2161003                BCU   R1,C2,L1
169 1C2 512000                  BCU   R1,R2,L1
170                     *
171                     *BRANCH CHARACTER LOW
172                     L1      EQU   *
173 105 E2163F                  BCL   C'A',C2,L1
    1C8 4F410607
174 1CC 4241060B                BCL   C'A',R2,L1
175 1D0 E10B3F                  BCL   C1,C'A',L1
    1D3 4F410C02
    1D7 1E14
176 1D9 F10B1216                BCL   C1,C2,L1
    1DD 561A
177 1DF B10B261E                BCL   C1,R2,L1
178 1E3 41410C02                BCL   R1,C'A',L1
    1E7 1E24
179 1E9 B2161C19                BCL   R1,C2,L1
    1ED 1E2A
180 1EF 51262D                  BCL   R1,R2,L1
181                     *
182                     *BRANCH CHARACTER LOW OR EQUAL
183 1F2 E2163F                  BCLE  C'A',C2,L1
    1F5 4F410E34
184 1F9 42410E38                BCLE  C'A',R2,L1
185 1FD E10B3F                  BCLE  C1,C'A',L1
    200 4F410402
    204 1E41
186 206 F10B1216                BCLE  C1,C2,L1
    20A 5247
```

```
187 20C B10B2E4B            BCLE  C1,R2,L1
188 210 41410402            BCLE  R1,C'A',L1
    214 1E51
189 216 B2161402            BCLE  R1,C2,L1
    21A 1E57
190 21C 512E5A              BCLE  R1,R2,L1
191                 *
192                 *BRANCH CHARACTER GREATER
193 21F E2163F              BCH   C'A',C2,L1
    222 4F410C02
    226 1E63
194 228 42410C02            BCH   C'A',R2,L1
    22C 1E69
195 22E 310B3F              BCH   C1,C'A',L1
    231 4F410670
196 235 F216110B            BCH   C1,C2,L1
    239 5676
197 23B B10B2C16            BCH   C1,R2,L1
    23F 1E7C
198 241 41410680            BCH   R1,C'A',L1
199 245 B2161684            BCH   R1,C2,L1
200 249 521687              BCH   R1,R2,L1
201                 *
202                 *BRANCH CHARACTER GREATER OR EQUAL
203 24C E2163F              BCHE  C'A',C2,L1
    24F 4F410402
    253 1E90
204 255 42410402            BCHE  C'A',R2,L1
    259 1E96
205 25B E10B3F              BCHE  C1,C'A',L1
    25E 4F410E9D
206 262 F216110B            BCHE  C1,C2,L1
    266 5EA3
207 268 B10B2402            BCHE  C1,R2,L1
    26C 1EA9
```

```
208 26E 41410EAD            BCHE  R1,C'A',L1
209 272 B2161EB1            BCHE  R1,C2,L1
210 276 521EB4              BCHE  R1,R2,L1
211                *CHARACTER TYPE TESTS
212 279 413A0C04            BCN   R1,L1           BRANCH CHARACTER NUMERIC
    27D 41300EBC
213 281 41470C0C            BCX   R1,L1           BRANCH CHARACTER HEXADECIMAL
    285 41410EC4
    289 413A0C04
    28D 41300ECC
214 291 415B0C04            BCA   R1,L1           BRANCH CHARACTER ALPHABETIC
    295 41410ED4
215                *
216                *
217                ***********************************************
218                *
219                * BIT INSTRUCTIONS
220                *
221                ***********************************************
222                *
223                *
224                *
225                *
226 299 810B                SB    B1              SET BIT
227 298 710B                ZB    B1              ZERO BIT
228 29D 910B0000            BBS   B1,L4           BRANCH BIT SET
229 2A1 910B0A04   L4        BBZ   B1,L4           BRANCH BIT ZERO
230                *
231                         CMNT  *               SEE MOV BIT TO BIT BELOW
232                ***********************************************
233                *
234                * DATA MOVEMENT AND ARITHMETIC INSTRUCTIONS
235                *
236                ***********************************************
237                *
```

```
238                 *MOVE DATA TO DATA BY AREA
239 2A5 A73058              MOV   32,H2
    2A8 A21619
240 2AB F2164730            MOV   32,T2
241 2AF F2168744            MOV   32,D2
242                 *
243 2B3 F216010B            MOV   H1,H2
244 2B7 A10B18              MOV   H1,T2
    2BA A21659
245 2BD A10B18              MOV   H1,D2
    2C0 A21699
246                 *
247 2C3 A10B58              MOV   T1,H2
    2C6 A21619
248 2C9 F216410B            MOV   T1,T2
249 2CD A10B58              MOV   T1,D2
    2D0 A21699
250                 *
251 2D3 A10B98              MOV   D1,H2
    2D6 A21619
252 2D9 A10B98              MOV   D1,T2
    2DC A21659
253 2DF F216810B            MOV   D1,D2
254                 *
255 2E3 7216                MOV   B1,B2           MOVE BIT TO BIT
    2E5 910B0802
    2E9 8216
256                 *
257                 *TEST AND SET ARITHMETIC FLAGS
258 2EB A10B02              TST   H1
259 2EE A10B42              TST   T1
260 2F1 A10B82              TST   D1
261                 *
262                 *
263                 *INCREMENT INSTRUCTIONS
```

```
264 2F4 A1ØBØ3              INC   H1              INCREMENT DATA BY 1
265 2F7 A1ØB43              INC   T1
266 2FA A1ØB83              INC   D1
267                 *
268 2FD A1ØB18              INC   H1,32            INCREMENT DATA AREA BY DATA
    3ØØ A73Ø53
    3Ø3 A1ØB19
269 3Ø6 F1ØB2216            INC   H1,H2
27Ø 3ØA A1ØB18              INC   H1,T2
    3ØD A21653
    31Ø A1ØB19
271 313 A1ØB18              INC   H1,D2
    316 A21693
    319 A1ØB19
272                 *
273 31C F1ØB673Ø            INC   T1,32
274 32Ø A21618              INC   T1,H2
    323 F1ØB6ØØ7
275 327 F1ØB6216            INC   T1,T2
276 32B A21698              INC   T1,D2
    32E F1ØB6ØØ7
277                 *
278 332 F1ØBA744            INC   D1,32
279 336 A21618              INC   D1,H2
    339 F1ØBAØØ6
28Ø 33D A2A658              INC   D1,T2
    34Ø F1ØBAØØ6
281 344 F1ØBA216            INC   D1,D2
282                 *
283                 *DECREMENT INSTRUCTIONS
284 348 A1ØBØ5              DEC   H1              DECREMENT DATA AREA BY 1
285 348 A1ØB45              DEC   T1
286 34E A1ØB85              DEC   D1
287
288 351 A1ØB18              DEC   H1,32            DECREMENT DATA AREA BY DATA
```

```
    354 A73055
    357 A10B19
289 35A F10B3216            DEC   H1,H2
290 35E A10B18              DEC   H1,T2
    361 A21655
    364 A10819
291 367 A10B18              DEC   H1,D2
    36A A21695
    36D A10B19
292                 *
293 370 F10B7730            DEC   T1,32
294 374 A21618              DEC   T1,H2
    377 F10B7007
295 378 B10B7216            DEC   T1,T2
296 37F A21698              DEC   T1,D2
    382 F10B7007
297                 *
298 386 F10BB744            DEC   D1,32
299 38A A21618              DEC   D1,H2
    38D F10BB006
300 391 A21658              DEC   D1,T2
    394 F10BB006
301 398 F10BB216            DEC   D1,D2
302                 *
303                 *ZERO OUT DATA AREA
304 39C A10B00              ZERO  H1
305 39F A10B40              ZERO  T1
306 3A2 A10B80              ZERO  D1
307                 *REPLACE DATA AREA WITH NUMBER 1
308 3A5 A10B01              ONE   H1
309 3A8 A10B41              ONE   T1
310 3AB A10B81              ONE   D1
311                 *NEGATE DATA AREA
312 3AE A10B08              NEG   H1
313 3B1 A10B48              NEG   T1
```

```
314 3B4 A10B88              NEG   D1
315                *LOAD DATA INTO ACCUMULATOR
316 3B7 A73C58              LOAD  876
317 3BA A10B18              LOAD  H1
318 3BD A10B58              LOAD  T1
319 3C0 A10B98              LOAD  D1
320                *STORE ACCUMULATOR INTO DATA AREA
321 3C3 A10B19              STORE H1
322 3C6 A10B59              STORE T1
323 3C9 A10B99              STORE D1
324                *ADD DATA TO ACCUMULATOR
325 3CC A74653              ADD   6547
326 3CF A10B13              ADD   H1
327 3D2 A10B53              ADD   T1
328 3D5 A10B93              ADD   D1
329                *SUBTRACT DATA FROM ACCUMULATOR
330 3D8 A74755              SUB   643
331 3DB A10B15              SUB   H1
332 3DE A10B55              SUB   T1
333 3E1 A10B95              SUB   D1
334                *MULTIPLY ACCUMULATOR BY DATA
335 3E4 A73A50              MUL   23
336 3E7 A10B10              MUL   H1
337 3EA A10B50              MUL   T1
338 3ED A10B90              MUL   D1
339                *DIVIDE ACCUMULATOR BY DATA
340 3F0 A73151              DIV   23
341 3F3 A10B11              DIV   H1
342 3F6 A10B51              DIV   T1
343 3F9 A10B91              DIV   D1
344                *
345                ******************************************
346                *
347                * REGISTER INSTRUCTIONS
348                *
```

```
349                 ****************************************
350                 *
351                 *MOV REGISTER TO REGISTER
352 3FC 1612                MOV   R1,R2
353 3FE E216D1              MOV   R1,S2
354 401 E10BE2              MOV   S1,R2
355 404 F216C10B            MOV   S1,S2
356                 *EXCHANGE REGISTER CONTENTS
357 408 1712                XRR   R1,R2
358 40A E10BEF              XRR   R1,S1
    40D E10BD1
    410 16F1
359 412 161F                XRR   S1,R1
    414 E10BE1
    417 E10BDF
360 41A E216EF              XRR   S1,S2
    41D F216C10B
    421 E10BDF
361                 *INCREMENT REGISTER
362 424 31                  INC   R1              BY 1
363 R25 A10B43              INC   S1
364 428 E73251              INC   R1,20           BY 2ND OPERAND
365 42B A10B18              INC   R1,H1
    42E E00751
366 431 E10B51              INC   R1,T1
367 434 A10B98              INC   R1,D1
    437 E00751
368                 *
369 43A F10B6732            INC   S1,20
370 43E A10B18              INC   S1,H1
    441 F10B6007
371 445 F10B610B            INC   S1,T1
372 449 A10B98              INC   S1,D1
    44C F10B6007
373                 *DECREMENT REGISTER
```

```
374 450 21                  DEC   R1              BY 1
375 451 A10B45              DEC   S1
376                 *
377 454 E74841              DEC   R1,25           BY 2ND OPERAND
378 457 A10B18              DEC   R1,H1
    45A E00741
379 45D E10B41              DEC   R1,T1
380 460 A10B98              DEC   R1,D1
    463 E00741
381 466 F10B7749            DEC   S1,1000
382 46A A10B18              DEC   S1,H1
    46D F10B7007
383 471 F10B710B            DEC   S1,T1
384 475 A10B98              DEC   S1,D1
    478 F10B7007
385                 *
386                 *LOAD ABSOLUTE ADDRESS DIFFERENCE
387                 L2      EQU   *
388 47C E0F6D1              LAD   R1,R2
    47F E0F6C2
389 482 E216C1              LAD   R1,S2
390 485 E10BC2              LAD   S1,R2
391 488 E216EF              LAD   S1,S2
    48B E10BCF
392                 *SET REGISTER TO ADDRESS
393 48E E21631              SRA   R1,C2
394 491 E21631              SRA   R1,H2
395 494 E21671              SRA   R1,T2
396 497 E216B1              SRA   R1,D2
397 49A E216F1              SRA   R1,S2
398 49D E10131              SRA   R1,L00
399                 *FLAG AND ATTACH ADDRESS REGISTER
400 4A0 423460              FAR   R2,X'34'
401                 *BRANCH REGISTER EQUAL (UNEQUAL)
402                 L3      EQU   *
```

```
403 4A3 E0F9D1              BE    R1,R2,L3
    4A6 C0F92A07
404 4AA C2161A0B            BE    R1,S2,L3
405 4AE F216D10B            BE    S1,S2,L3
    4B2 5A11
406 4B4 E0F9D1              BU    R1,R2,L3
    4B7 C0F92218
407 4BB C216121C            BU    R1,S2,L3
408 4BF C10B2220            BU    S1,R2,L3
409 4C3 C10B2A24            BE    S1,R2,L3
410 4C7 F216D10B            BU    S1,S2,L3
    4CB 522A
411                 *
412                 *
413                 *
414                 ***********************************************
415                 *
416                 * DATA COMPARISON INSTRUCTIONS
417                 *
418                 ***********************************************
419                 *
420                 *BRANCH 1ST OPERAND = 2ND OPERAND
421 4CD E10B3F              BE    25,H1,L6
    4D0 4F190949
422 4D4 F733510B            BE    26,T1,L6
    4D8 5943
423 4DA F734910B            BE    27,D1,L6
    4DE 593D
424                 *
425 4E0 A10B18              BE    H1,X'25',L6
    423 F0075736
    4E7 5934
426 4E9 F10B110B            BE    H1,H1,L6
    4ED 592E
427 4EF A10B18              BE    H1,T1,L6
```

```
    4F2 F00751ØB
    4F6 5925
428 4F8 A1ØB18              BE    H1,D1,L6
    4FB F00691ØB
    4FF 591C
429                 *
430 5Ø1 F1ØB573D            BE    T1,12345,L6
    5Ø5 5916
431 5Ø7 A1ØB18              BE    T1,H1,L6
    5ØA F1ØB5ØØ7
    5ØE 59ØD
432 51Ø F1ØB5216            BE    T1,T2,L6
    514 59Ø7
433 516 A1ØB58              BE    T1,D1,L6
    519 F00691ØB
    51D 58FE
434                 *
435 51F F1ØB974A            BE    D1,X'1234',L6
    523 58F8
436 525 A1ØB18              BE    D1,H1,L6
    528 F1ØB9ØØ6
    52C 58EF
437 52E A1ØB58              BE    D1,51,L6
    531 F1ØB9ØØ6
    535 58E6
438 537 F1ØB9216            BE    D1,D2,L6
    53B 58EØ
439                 *
44Ø                 *BRANCH 1ST OPERAND NOT EQUAL SECOND OPERAND
441 53D E1ØB3F              BU    25,H1,L6
    54Ø 4F19ØØD9
442 544 F733351ØB           BU    26,T1,L6
    548 5ØD3
443 54A F73491ØB            BU    27,D1,L6
    54E 5ØCD
```

```
444                 *
445 550 A10B18              BU    H1,X'25',L6
    553 F0075736
    557 50C4
446 559 F10B110B            BU    H1,H1,L6
    55D 50BE
447 55F A10B18              BU    H1,T1,L6
    562 F007510B
    566 50B5
448 568 A10B18              BU    H1,D1,L6
    56B F006910B
    56F 50AC
449                 *
450 571 F10B573D            BU    T1,12345,L6
    575 50A6
451 577 A10B18              BU    T1,H1,L6
    57A F10B5007
    57E 509D
452 580 F10B5216            BU    T1,T2,L6
    584 5097
453 586 A10B58              BU    T1,D1,L6
    589 5006910B
    58D 508E
454                 *
455 58F F10B974A            BU    D1,X'1234',L6
    593 5088
456 595 A10B18              BU    D1,H1,L6
    598 F10B9006
    59C 507F
457 59E A10B58              BU    D1,T1,L6
    5A1 F10B9006
    5A5 5076
458 5A7 F10B9216            BU    D1,D2,L6
    5AB 5070
459                 *
```

```
46Ø                  *BRANCH 1ST OPERAND LESS THAN 2ND OPERAND
461 5AD E1ØB3F               BL    25,H1,L6
    5BØ 4F19Ø469
462 5B4 F733510B             BL    26,T1,L6
    5B8 5463
463 5BA F734910B             BL    27,D1,L6
    4BE 545D
464                  *
465 5CØ A1ØB18               BL    H1,X'25',L6
    5C3 FØØ75736
    5C7 5454
466 5C9 F1ØB11ØB             BL    H1,H1,L6
    5CD 544E
467 5CF A1ØB18               BL    H1,T1,L6
    5D2 FØØ751ØB
    5D6 5445
468 5D8 A1ØB18               BL    H1,D1,L6
    5D8 FØØ691ØB
    5DF 543C
469                  *
47Ø 5E1 F1ØB573D             BL    T1,12345,L6
    5E5 5436
471 5E7 A1ØB18               BL    T1,H1,L6
    5EA F1ØB5ØØ7
    5EE 542D
472 5FØ F1ØB5216             BL    T1,T2,L6
    5F4 5427
473 5F6 A1ØB58               BL    T1,D1,L6
    5F9 FØØ691ØB
    5FD 541E
474                  *
475 5FF F1ØB974A             BL    D1,X'1234',L6
    6Ø3 5418
476 6Ø5 A1ØB18               BL    D1,H1,L6
    6Ø8 F1ØB9ØØ6
```

```
    60C 540F
477 60E A10B58              BL    D1,T1,L6
    611 F10B9006
    615 5406
478 617 F10B9216            BL    D1,D2,L6
    61B 5400
479                 *
480                 L6      EQU   *
481                 *BRANCH 1ST OPERAND LESS OR EQUAL 2ND OPERAND
482 61D 310B3F              BLE   25,H1,L6
    620 4F190E07
483 624 F733510B            BLE   26,T1,L6
    628 5E0D
484 62A F734910B            BLE   27,D1,L6
    62E 5E13
485                 *
486 630 A10B18              BLE   H1,X'25',L6
    633 F0075736
    637 5E1C
487 639 F10B110B            BLE   H1,H1,L6
    63D 5E22
488 63F A10B18              BLE   H1,T1,L6
    642 F007510B
    646 5E2B
489 648 A10B18              BLE   H1,D1,L6
    64B F006910B
    64F 5E34
490                 *
491 651 F10B573D            BLE   T1,12345,L6
    655 5E3A
492 657 A10B18              BLE   51,H1,L6
    65A F10B5007
    65E 5E43
493 650 F10B5216            BLE   T1,T2,L6
    664 5E49
```

```
494 666 A10B58              BLE   T1,D1,L6
    669 F00691OB
    66D 5E52
495                 *
496 66F F10B974A            BLE   D1,X'1234',L6
    673 5E58
497 675 A10B18              BLE   D1,H1,L6
    678 F10B9006
    67C 5E61
498 67E A10B58              BLE   D1,T1,L6
    681 F10B9006
    685 5E6A
499 687 F10B9216            BLE   D1,D2,L6
    68B 5E70
500                 *
501                 *BRANCH 1ST OPERAND GREATER THAN 2ND OPERAND
502 68D A10B18              BH    25,H1,L6
    690 F0075748
    694 5679
503 697 F10B5733            BH    26,T1,L6
    69A 567F
504 69C F10B9734            BH    27,D1,L6
    6A0 5685
505                 *
506 6A2 E10B3F              BH    H1,X'25',L6
    6A5 4F25068C
507 6A9 F10B110B            BG    H1,H1,L6
    6AD 5692
508 6AF A10B18              BH    H1,T1,L6
    6B2 F10B5007
    6B6 569B
509 6B8 A10B18              BH    H1,D1,L6
    6BB F10B9006
    6BF 56A4
510                 *
```

```
511 6C1 F73D510B            BH    R1,12345,L6
    6C5 566AA
512 6C7 A10B18              BH    T1,H1,L6
    6CA F00751ØB
    6CE 56B3
513 6DØ F21651ØB            BH    T1,T2,L6
    6D4 56V9
514 6D6 A10B58              BH    R1,D1,L6
    6D9 F10B9006
    6DD 56C2
515                 *
516 6DF F74A91ØB            BH    D1,X'1234',L6
    6E3 56C8
517 6E5 A10B18              BH    D1,H1,L6
    6E8 F00691ØB
    6EC 56D1
518 6EE A10B58              BH    D1,T1,L6
    6F1 F00691ØB
    6F5 56DA
519 6F7 F21691ØB            BH    D1,D2,L6
    6FB 56EØ
52Ø                 *
521                 *BRANCH 1ST OPERAND GREATER OR EQUAL 2ND OPERAND
522 6FD A10B18              BHE   25,H1,L6
    7ØØ F0075748
    7Ø4 5EE9
523 7Ø6 F10B5733            BHE   26,T1,L6
    7ØA 5EEF
524 7ØC F10B9734            BHE   27,D1,L6
    71Ø 5EF5
525                 *
526 712 E10B3F              BHE   H1,X'25',L6
    715 4F250EFC
527 7L9 F10B110B            BHE   H1,H1,L6
    71D 5F02
```

```
528 71F A10B18              BHE   H1,T1,L6
    722 F10B5007
    726 5F0B
529 728 A10B18              BHE   H1,D1,L6
    72B F10B9006
    72F 5F14
530                 *
531 731 F73D510B            BHE   T1,12345,L6
    735 5F1A
532 737 A10B18              BHE   T1,H1,L6
    73A F0075108
    73E 5F23
533 740 F216510B            BHE   T1,T2,L6
    744 5F29
534 746 A10B58              BHE   T1,D1,L6
    749 F20B9006
    74D 5F32
535                 *
536 74F F74A910B            BHE   D1,X'1234',L6
    753 5F38
537 755 AQ0B18              BHE   D1,H1,L6
    758 F0069108
    75C 5F41
538 75E A10B58              BHE   D1,T1,L6
    761 F0069108
    765 5F4A
539 767 F216910B            BHE   D1,D2,L6
    76B 5F50
540                 *
541                 *DECREMENT OPERAND 1 BY OPERAND 2 AND BRANCH IF RESULT IS ZERO
542 76D F10B1216            BDZ   H1,H2,L7
    771 7872
543 773 F10B5737            BDZ   T1,5,L7
    777 786C
544 779 F10B5216            BDZ   T1,T2,L7
```

```
    77D 7866
545 77F F10B973E            BDZ   D1,10,L7
    783 7860
546 785 F10B9216            BDZ   D1,D2,L7
    789 785A
547                 *DECREMENT OPERAND 1 BY OPERAND 2 AND BRANCH RESULT NOT ZERO
548 78B F10B1216            BDNZ  H1,H2,L7
    78F 7054
549 791 F10B5737            BDNZ  T1,5,L7
    795 704E
550 797 F10B5216            BDNZ  T1,T2,L7
    79B 7048
551 79D F10B973E            BDNZ  D1,10,L7
    7A1 7042
552 7A3 F10B9216            BDNZ  D1,D2,L7
    7A7 703C
553                 *DECREMENT OPERAND 1 BY OPERAND 2 AND BRANCH RESULT NEGATIVE
554 7A9 F10B1216            BDLZ  H1,H2,L7
    7AD 7436
555 7AF F10B5738            BDLZ  T1,5,L7
    7B3 7430
556 7B5 F10B5216            BDLZ  T1,T2,L7
    7B9 742A
557 7BB F10B973E            BDLZ  D1,10,L7
    7BF 7424
558 7C1 F10B9216            BDLZ  D1,D2,L6
    7C5 741E
559                 *DECREMENT OPERAND 1 BY OPERAND 2 AND BRANCH RESULT ZERO OR NEGATIVE
560 7C7 F10B1216            BDLEZ H1,H2,L7
    7CB 7C18
561 7CD F10B5737            BDLEZ T1,5,L7
    7D1 7C12
562 7D3 F10B5216            BDLEZ T1,T2,L7
    7D7 7C0C
563 7D9 F10B973E            BDLEZ D1,10,L7
```

```
    7DD 7C06
564 7DF F10B9216            BDLEZ D1,D2,L7
    7E3 7C00
565                 *DECREMENT OPERAND 1 BY ONE AND BRANCH RESULT ZERO
566                 L7      EQU   *
567 6E5 F10B574C            BDZ   T1,L7
    7E9 7A06
568 7EB F10B9740            BDZ   D1,L7
    7EF 7A0C
569                 *DECREMENT OPERAND 1 BY ONE AND BRANCH RESULT NOT ZERO
570 7F1 F10B574C            BDNZ  T1,L7
    7F5 7212
571 7F7 F10B9740            BDNZ  D1,L7
    7FB 72A8
572                 *DECREMENT OPERAND 1 BY ONE AND BRANCH RESULT NEGATIVE
573 7FD F10B574C            BDLZ  T1,L7
    801 761E
574 803 F10B9740            BDLZ  D1,L7
    807 7624
575                 *DECREMENT OPERAND 1 BY ONE AND BRANCH IF RESULT NEGATIVE OR ZERO
576 809 F10B574C            BDLEZ T1,L7
    80D 7E2A
577 80F F10B9740            BDLEZ D1,L7
    813 7E30
578                 *
579                 *
580                 ********************************************
581                 *
582                 * CONVERSION INSTRUCTIONS
583                 *
584                 ********************************************
585                 *
586                 *MOVE BINARY TO DECIMAL
587 815 A10B18               MBD   H1,R2
    818 162F
```

```
    81A 110008
    81D 16F2
588 81F A10B58              MBD   T1,R2
    822 162F
    824 110008
    827 16F2
589 829 1A0B98              MBD   D1,R2
    82C 162F
    82E 110008
    831 16F2
590 833 F0144739            MBD   8,H1,R2
    837 A10B18
    83A 162F
    83C 111008
    83F 16F2
591 841 F0144739            MBD   8,T1,R2
    845 A10B58
    848 162F
    84A 111008
    84D 16F2
592 84F F0144739            MBD   4,D1,R2
    853 A10B98
    856 162F
    858 111008
    85B 16F2
593                 MOVE DECIMAL TO BINARY
594 85D D21641              MDB   R1,T2
595 860 D21681              MDB   R1,D2
596                 MOVE BINARY TO HEX
597 863 D10B32              MBX   H1,R2
598 866 D10B72              MBX   T1,R2
599 869 D10BB2              MBX   D1,R2
600
601 86C A73A58              MBX   2,H1,R2          MAXIMUM DIGITS CONVERTED = OPERAND 1
    86F D10B32
```

```
602 872 A74D58              MBX   4,T1,R2
    875 D10B72
603 878 A73958              MBX   8,D1,R2
    87B D10BB2
604                 *
605 87E A74258              MBXN  2,H1,R2          DIGITS CONVERTED = OPERAND 1
    881 D10B32
606 884 A73B58              MBXN  4,T1,R2
    887 D10B72
607 88A A74358              MBXN  8,D1,R2
    88D D10BB2
608                 *
609 890 D21621              MXB   R1,H2            MOVE HEX TO BINARY
610 893 D21661              MXB   R1,T2
611 896 D216A1              MXB   R1,D2
612                 *
613                 *
614                 **********************************************
615                 *
616                 * EXECUTION TRANSFER INSTRUCTIONS
617                 *
618                 **********************************************
619                 *
620                 *
621 899 1C03                B     L5               BRANCH LOCAL
622 89B 0C1234              B     A1
623                 *
624 89E 1A02        L5      BSL   L5               BRANCH AND STACK LOCATION LOCAL
625 8A0 111002              BSL   M1               BRANCH AND STACK LOCATION EXTERNAL
626 8A3 0D1234              BSL   A1               BRANCH AND STACK LOCATION ABSOLUTE
627                 *
628 8A6 13                  BSLI  *                BRANCH AND STACK LOCATION INDIRECT THROUGH ACCUMULAT
629                 *
630 8A7 14                  RTN   *                RETURN
631 8A8 15                  TEXT  X'15'            RETURN WITHOUT TRACE
```

```
632                 *
633 8A9 101002              ENT   M1              BRANCH EXTERNAL
634                 *
635 8AC 12                  ENTI  *               BRANCH EXTERNAL INDIRECT THROUGH ACCUMULATOR
636                 *
637 8AD A10B18              BSL*  H1              BRANCH AND STACK LOCATION THROUGH HALF TALLY
    8B0 13
638 8B1 A10B58              BSL*  T1              BRANCH AND STACK LOCATION INDIRECT THROUGH TALLY
    8B4 13
639 8B5 A10B98              BSL*  D1              BRANCH AND STACK LOCATION INDIRECT THROUGH DOUBLE TA
    8B8 13
640 8B9 A10B18              ENT*  H1              BRANCH EXTERNAL INDIRECT THROUGH HALF TALLY/TALLY/D
    8BC 12
641 8BD A10B58              ENT*  T1
    8C0 12
642 8C1 A10B98              ENT*  D1
    8C4 12
643                 *
644                 *
645                 ***********************************************
646                 *
647                 * I/O AND CONTROL INSTRUCTIONS
648                 *
649                 ***********************************************
650                 *
651                 *
652 8C5 426778              IOI   R2,3,7          CALL MONITOR TO INPUT A BYTE
653 8C8 413578              IOO   R1,1,5          CALL MONITOR TO OUTPUT A BYTE
654                 *
655 8CB 6115                READ  R1              READ ONE BYTE FROM TERMINAL BUFFER
656 8CD 622D                WRITE R2              WRITE ONE BYTE TO TERMINAL BUFFER
657                 *
658 8CF 410274              MCAL  R1,2,4          MONITOR CALL
659                 *
660 8D2 400079              RQM   *               RELEASE QUANTUM
```

```
661                     *
662 8D5 411290                  IB      R1,X'12'        I/O INSTRUCTIONS
663 8D8 413480                  OB      R1,X'34'
664                     *
665 8DB 00                      NOP     *               NO OPERATION
666                     *
667 8DC 08                      HALT    *               HALT
668 8DD 4100A0                  HLD     R1              HALT AND DISPLAY
669                     *
670 830 01                      TEXT    X'01'           TEST INTERRUPTS
671                     *
672 8E1 4100A3                  ECS     R1              ENTER CONSOLE COMMAND SWITCHES
673 8E4 4100A1                  ESS     R1              ENTER SENSE SWITCHES
674                     *
675 8E7 0B                      SVP     *               START VIRTUAL PROCESS
676 8E8 0A                      RVP     *               RETURN TO VIRTUAL PROCESS
677                     *
678                     *
679                     ************************************************************
680                     *
681                     * ASSEMBLER DIRECTIVES
682                     *
683                     ************************************************************
684                     *
685 8E9 7FF00064                FRAME   100
686                     *
687                     LABEL1  EQU     *               EQUATES
688                     LABEL2  EQU     C1
689                     LABEL3  EQU     H1
690                     LABEL4  EQU     T1
691                     LABEL5  EQU     D1
692                     LABEL6  EQU     S1
693                     LABEL7  EQU     R1
694                     LABEL8  EQU     LABEL1
695                     LABEL9  EQU     X'600'
```

```
696                     *
697                     LABEL10   ORG    LØØ              ORG'S
698 ØØ1 534F4D45                  TEXT   C'SOME CODE'
    ØØ5 2Ø434F44
    ØØ9 45
699                               ORG    1
7ØØ                     *
7Ø1                     ABS       DEFA   X'Ø8ØD'          DEFINE ABSOLUTE ADDRESS
7Ø2                     *
7Ø3                               SETAR  7                SET PROGRAM ADDRESS REGISTER
7Ø4 ØØ1 A74358                    LOAD   1234
7Ø5                     *
7Ø6                     SPEC1     DEFB   R1,*1+7          SPECIAL DEF(K) FORMS
7Ø7                     SPEC2     DEFB   1,*1+11
7Ø8 ØØ4 712B                      MOV    SPEC1,SPEC2
    ØØ6 91272Ø8Ø2
    ØØA 812B
7Ø9                     *
71Ø                     SPEC3     DEFC   R1,*+3
711                     SPEC4     DEFC   1,*+11
712 ØØC F117Ø1ØF                  MCC    SPEC3,SPEC4
713                     *
714                     SPEC5     DEFH   41,*+3
715                     SPEC6     DEFH   1,*+11
716 Ø1Ø F11BØ113                  MOV    SPEC5,SPEC6
717                     *
718                     SPEC7     DEFT   R1,*16+3
719                     SPEC8     DEFT   1,*16+11
72Ø Ø14 F115412D                  MOV    SPEC7,SPEC8
721                     *
722                     SPEC9     DEFD   R1,*16+3
723                     SPEC1Ø    DEFD   1,*16+11
724 Ø18 F117812F                  MOV    SPEC9,SPEC1Ø
725                     *
726                     SPEC11    DEFS   R1,*16+3
```

```
727                     SPEC12    DEFS  1,*16+11
728 01C F119C111                  MOV   SPEC11,SPEC12
729                     *
730                     * PLEASE SEE ABOVE FOR DEF(K)
731                     *
732 020 41              LABEL12   CHR   C'A'
733 021 19                        HTLY  25
734 022 0200                      TLY   X'200'
735 024 0001E240                  DTLY  123456
736 028 00000000                  SR    0
    02C 0000
737                     *
738                     REG1      AR    R1              DEFINE TYPE ADDRESS REGISTER
739                     REG2      AR    2
740                     REG3      AR    HS
741                     *
742                     LABEL13   DEFM  0,77            DEFINE MODAL ENTRY
743                     LABEL14   DEFM  1,LABEL13
744 02E 11004D                    BSL   LABEL13
745 031 11104D                    BSL   LABEL14
746                     *
747 034 54484953                  TEXT  C'THIS IS A TEXT MESSAGE',C'THIS IS SOME MORE',X'FF'
    038 20495320
    03C 41205445
    040 5854204D
    044 45535341
    048 4745
    04A 54484953
    04E 20495320
    052 534F4D46
    056 204D4F52
    05A 45
    05B FF
748                               CMNT  *               THIS ALLOWS COMMENTS TO BE STARTED IN THE COMMENTS
749 05C 0006            *
```

```
    05E 0050
    060 0020
    062 0017
    064 0014
    066 001A
    068 0000001B
    06C 0025
    06E 0005
    070 0005
    072 0008
    074 0002
    076 0084
    078 0360
    07A 3039
    07C 0000000A
    080 00000001
    084 0082
    086 0088
    088 00000020
    08C 1993
    08E 0283
    090 0019
    092 03F8
    094 00001234
    098 0001
    09A 0004
    09C 04D2
EOF:
```

THIS PAGE INTENTIONALLY LEFT BLANK

Assembler Tables

The REAL Assembler is completely table-driven and is therefore both powerful and flexible in its definition of mnemonics. In addition, the assembler accesses a permanent symbol table, which allows the predefinition of a set of symbols used by all assemblies. Symbols defined in the source mode are placed in a temporary (local) symbol table, and such entries override corresponding entries in the permanent symbol file. It should be noted that forward references to local symbols that match entries in the permanent symbol table will, in general, cause assembly errors. Therefore, such overriding symbol definitions must precede the first reference to them.

At the start of the assembly process, the assembler searches the Master Dictionary (M/DICT) of the data-base for the following file definitions:

PSYM - Permanent symbol table.

TSYM - Temporary symbol table.

OSYM - Operation-code symbol table.

The assembly will abort if any of these file-definitions are missing, with a message indicating the one that was not found. The temporary symbol table is inititalized before the assembly starts. Since the TSYM is actually a permanently defined file on a user's account (M/DICT) it must be pre-defined and can be examined at the conclusion of the assembly. Furthermore, only one person may be using the REAL assembler per account.

TSYM/PSYM Table Entry Formats

The item format of the entries in the PSYM & TSYM files is as follows: (Entries are in character form):

Item - id: Symbol-name

Line 1 : Symbol-code (single character - see below)

Line 2 : Symbol-value (hexadecimal location or displacement)

Line 3 : Base-register value (single hexadecimal digit)

Symbol-Codes

The symbol code is a single character code that defines the type of the symbol, it is used in the operation code lookup to determine legal operands, and to flag undefined or multi-defined labels, etc.

Code	Description - Symbol Type	Unit of Displacement
B	Bit	Bits
C	Character Register	Bytes
D	Double-Word (4-byte)	Words
H	Half-Word (]-byte)	Bytes
L	Local Symbol, defined	Bytes
M	Mode - id	Undefined
N	Literal Value	Bytes
R	Address Register	Undefined
S	Storage Register (6 Bytes)	Words
T	Word (2 Bytes)	Words
U	Local Symbol, Undefined	Value=0

OSYM Table-Lookup Technique

All REAL mnemonic operation codes are stored in the OSYM file. An entry in this table may be either (1) the REAL mnemonic for the instruction (basic op-code), or (2) the REAL mnemonic suffixed by the symbol type-codes of all the operand field entries. The purpose of the suffixing is (1) to provide for the separate handling of REAL mnemonics with variable operand field entries; (2) to provide for a check on the number and type of operand field entries. (As an example, the basic REAL mnemonic for "move register to register" is MOV, but it has four different object code expansions, depending on whether the registers involved are address-(R), or storage-type (S). To allow for all cases, there are four entries in the OSYM file: MOVRR, MOVRS, MOVSR and MOVSS. The assembler will attempt to look up the basic op-code first, and, if it is not found, a second attempt will be made with the basic op-code suffixed as described above.

TSYM Table Entry Setup

As the assembler goes through the "suffixing" technique described above, it necessarily looks up each non-literal operand in the TSYM and PSYM files, in that order. If found, the type-code can be suffixed to the basic op-code. If no entry is found in the TSYM & PSYM files, the assembler then sets up an entry in the TSYM file with type "U" (undefined), and location zero. This has an important ramification with regard to literal generation.

OSYM Table Entry Format

Line one of the OSYM table entry may be one of the following:

M - Defines a macro; all further lines are macro substitution lines.

P - Defines "primitive" which calls one of the lower-level assem-ler functions.

Q - Equates this entry to another OSYM table entry specified in the next line.

Macro Definition Format

Each substitution line in a macro definition will generate a new source statement, to which parameters may be passed from the original source statement. This newly generated source statement will, in turn, be assembled as any other source statement. Thus a macro may cause the original statement to expand into an unlimited set of new statements; however, if any generated statement calls another macro, control is passed immediately to the new macro, and the previous one cannot regain control.

Data in a macro substitution line is transferred, as it is, to generate the new source statement, except for the substitution codes, which cause a specific parameter to be substituted. Substitution codes are enclosed by parentheses:

CODE	ACTION
AF Substitution: (n)	(n decimal) causes insertion of the n-th Argument-field entry of the original source statement; if such an entry is non-existene, no substitution takes place.
Label Substitution: (L ± n)	(The ±n is optional). Causes insertion of label internal to the macro; the label is created using the macro label counter (MLC), which is initialized by the assembler at the start of an assembly. If the label substitution is in the label-field of the generated source statement, it is replaced with a label of the form "=Lm" where m=MLC + 1, and the MLC is incremented by one. If the label substitution code is in the operand field, it is replaced with a symbol of the form "=Lm" where m=MLC ± n, the MLC being unaltered. (See paragraph "Example of REAL Macro Expansion" following).

"Primitive" Definition Formats

Each line in a primitive is an assembler-directive that calls a specific assembler process. The first character in each line is a code defining the process:

CODE	PROCESS
A	Align location counter on word boundary.
E	Exit to explicitly defined process.
G	Generate object-code (GEN)
R	Reset entry in TSYM file (RESET)

Exit Format

E:mode-ID

Where "mode-ID" is the hexadecimal mode-ID of the processor which is to be entered.

Gen Format

G, a_1, a_2.... b_1, b_2,.... (A & B-fields separated by one blank)

A-field B-field

The G-primitive causes the actual generation of object code. There should be a one-to-one correspondence between entries in the A- & B- fields. Each A-field entry is a decimal number, and specifies the number of bits of code to be generated using the corresponding B-field entry. The sum of the A-field values must be a multiple of 8, and must be less than or equal to 32.

ENTRY	VALUE RETURNED
*	Current location counter value, in bytes.
*n	As above, modulo "n" bits (n decimal)
B	Current base register value.
n	Decimal literal.
X'h'	Hexadecimal literal.

ENTRY	VALUE RETURNED
C'k'	Character literal (one character only).
An;m	Value returned is: (1) AFn if AFn is a literal; or (2) From line "m" of AFn if AFn is a symbol. (Zero is symbol undefined).
Jn;k	Used to compute relative displacement from current location (*), to the location defined by AFn. Value returned is signed magnitude, 10-bits: (Value (AFn)-*)=k +k if backward reference -k if forward reference

Reset Format

R,n $b_1,b_2,....b_m$

Resets values in TSYM entry for AFn. B-field entries refer to lines 1 through m, and entries are as above. This is used to re-define TSYM table entries: when the assembler finds a label-definition, it inserts the entry in the TSYM as type 'L', location as current location in bytes, and base register field from the current base register. To redefine the entry, a R-primitive is used. For example, the opcode 'TLY' is used to define a local tally, as in:

```
LABEL     TLY     123
```

The OSYM entry for 'TLY' is:

Line 1 (type)	:	P	
Line 2	:	R,0 C'T',*16,B	Redefines type as T; location module 16.
Line 3	:	G,16 A2;2	Generates 2-byte object-code value

Assembler Output

The assembler output consists of (1) macro statement expansions; (2) error messages and (3) generated object code, all appended to the original source statement.

A user-input source statement is of the format:

Source Statement (AM)

On output, the format is as follows:

Source Statement (SVM) location object-code (AM)

where 'location' is a 3-digit hexadecimal field, and the 'object code' is in hexadecimal.

Error messages are appended to the source statement as the assembler encounters errors; the messages are appended in the format:

..(VM)* message....

Messages may precede or follow the object code.

Macro expansions resemble source statements in terms of source statement, errors and object code, and are of the format:

Source Statement (VM) macro statement (SVM) loc. obj. code (VM)... (AM).

Note that regardless of what the assembler appends to the original source statement, the delimiters surrounding the entire statement remain unchanged; this ensures proper source statement input on subsequent assemblies.

Literal Generation

REAL statements that require assembly of literal should setup entries in the TSYM of the following format:

Item-ID	:	=k value	k=	S storage register
Symbol-type	:	U		D double word
Location	:	0		T word

This will be done simply by the reference of the symbol as an operand in a source statement. However, the OSYM table entry that the source statement references must be of the "suffixed" type rather than being the basic op-code. For example, the statement:

MOV =T23,CTRI

which references the OSYM table entry 'MOVTT', and, in so doing, sets up the TSYM entry '=T23' as type U to be assembled as a literal.

At the end of pass I, the assembler searches the TSYM file for undefined entries; if they are of the format shown above, a dummy source statement of the form:

LABEL	OPCODE	OPERAND
=k value	:k	value

is generated and assembled. Thus the entries ":S", ":D", ":T" in the OSYM are reserved and cause generation of 6-byte (storage register, type S), 4-byte (double-word, type D) and 2-byte (word, type T) literals, respectively.

Reassembly in Pass II

During the assembly process, statements which have a forward reference are flagged for re-assembly by prefixing the character "X" to the location counter and object code data that are appended to the source statement. The REAL assembler is not a true two-pass assembler; pass II consists of scanning the mode for statements that have been flagged for re-assembly, and re-assembling those statements exclusively. If they contain references to undefined symbols, the object code output will still have the "reassemble" flag stored with it, after pass II.

Assembler Error Messages

Message	Explanation
*UNDEF: $Symbol_1$ $Symbol_2$....	Undefined symbols at end of pass I (Message at end-of-mode).
*LABEL TYPE	Label-field format error.
*MULTIDEF	Label-field entry was previously defined.
*REF-UNDEF	Reference to undefined symbol.
*LABEL ?	Required label-field missing.
*OPCODE ?	Op-code-field entry missing.
*OPERAND ?	Required operand-field entry missing.
*OPCD ILLGL:opcode	Either the opcode was illegal, or the operand types were illegal for the opcode.

*OPRND TYPE	The operand-field entry was an illegal type; eg: ORG statement with undefined symbol, SETAR with non-numeric operand, etc.
*RANGE ERR	The range of the operand-field entry is illegal; eg: SETAR with n not $0 \leqslant n < 16$.
*TRUNCATION	Object code truncation may be due to: branch out-of-range; TSYM/PSYM table entry error; specification error in the GEN primitive.

The following are errors in the OSYM-table entry specifications.

*A-FIELD ? *B-FIELD ?	Error in A- or B-field specification.
*OPCODE-TYPE ERR	Opcode type not a P/Q/M, or primitive type was illegal.
*MACRO-SPEC ERR	Error in the macro specification

Example of REAL Macro Expansion

```
Location Counter  =  X'012'
        MLC       =  0

Source Statement      :     BCN        R4,LABX

(Branch if character
addressed by R4 is
numeric)
```

		Symbol	Type	Displacement
PSYM table entry	:	R4	R	0004
TSYM table entry	:	LABX	L	0024
OSYM table lookup	:	BCNRL		
Line 1 (type)	:	M		
Line 2	:	BCL (2),X'30',(L + 1)		If 0, skip next
Line 3	:	BCLE (2),X'39',(3)		Branch is < 9
Line 4	:	(L) EQU *		Define Internal label

Generated Source Statements

From Line 1 : BCL R4,X'30',=L01

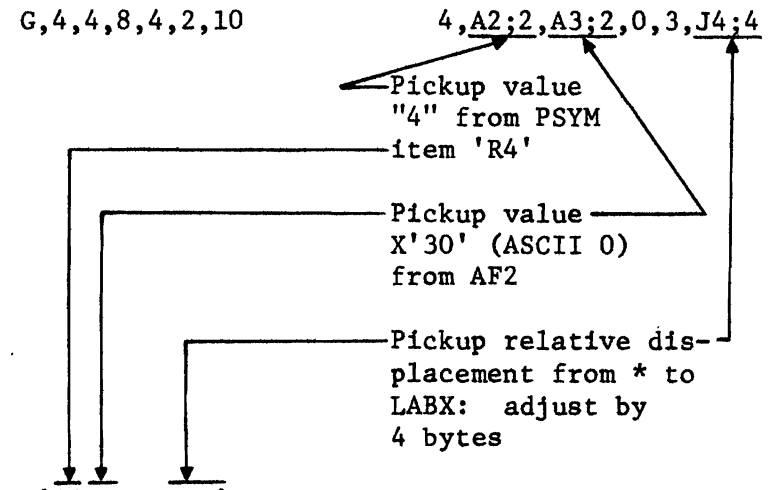
From Line 2 : BCLE R4,X'39',LABX

From Line 3 : =L01 EQU *

OSYM table lookup : BCLERNL

Line 1 (type) : P

Line 2 : G,4,4,8,4,2,10 4,A2;2,A3;2,0,3,J4;4



Object Code : X'4430 0 C0E'

Section XVII

THE INTERACTIVE DEBUGGER

INTRODUCTION

The interactive debugger (DEBUG) provides a means for monitoring and controlling program execution. DEBUG is normally used in the check-out phase of assembly language programs. DEBUG also has the ability to turn the print off at the terminal, and to terminate program execution.

To use the facilities of DEBUG other than turning the terminal printing on and off, and terminating program execution, requires system privileges level two. If the user has such privileges, he may control execution by the insertion of break-points in his program, and by executing a specific number of instructions. He may trace execution by displaying data at specific locations. DEBUG also allows the user to display data throughout the virtual memory of the system.

The prompt character of the debugger is the exclamation point (!). DEBUG is entered voluntarily by depressing the BREAK key on the terminal, or involuntarily when a hardware trap condition occurs. In the latter case, a message indicating the nature of the error causing the trap, and the location at which the trap occurred, is displayed prior to the DEBUG prompt character.

DEBUG Syntax

This section defines the terms "address", "indirect address", "window", and "format", which are used in the command descriptions in the following sections.

An Address (a) references a byte in the virtual memory, by specifying a frame-ID (FID), and an offset displacement within the frame. The FID and/or displacement may be in decimal or in hexadecimal; the general forms of an Address are as below:

f,d	Fid in decimal, displacement in decimal.
f.d	Fid in decimal, displacement in hexadecimal.
.f,d	Fid in hexadecimal, displacement in decimal.
.f.d	Fid in hexadecimal, displacement in hexadecimal.

If the Fid is omitted, the PCB-Fid is used as a default value. The displacement must be in the range $0 \leq f < 512$.

For example, the following Fid and displacement specifications are equivalent: 12.3C = 12,60 = .C.3C = .C,60

An Indirect Address (i) references a byte in the virtual memory by specifying an Address Register which therefore indirectly references a particular byte. Address Registers zero and one cannot be used in this manner. The Indirect Address specification takes the forms:

Rr	r in decimal	$2 \leq r \leq 15$
R.r	r in hexadecimal;	

Note that Indirect Addresses have an implied displacement within the Fid that the register is pointing to; this displacement depends on whether the register is in the "linked" or the "unlinked" format (described in the CPU section).

A Window (w) specifies the number of bytes to display (m), and optionally the negative displacement (n) from the Address or Indirect Address, from which to start the display. If n is not specified, it is assumed to be zero. The forms of the window are:

;m	m in decimal
;.m	m in hexadecimal
;n,m	n and m in decimal
;n.m	n in decimal, m in hexadecimal
;.n,m	n in hexadecimal, m in decimal
;.n.m	n and m in hexadecimal.

The default window is 0,4 (no negative displacement, display four bytes).

A Format (c) indicates whether the data is to be displayed in hexadecimal, character, or integer format. The single characters X, C and I specify hexadecimal, character, or integer display respectively. On initial entry to DEBUG, the format is an X.

General DEBUG Statement Format

Instruction Mnemonic or Format	Address or Indirect Address	Window

If format or window is not specified, the previously specified values will remain in effect.

DEBUG Commands

Instruction Mnemonic	Description
B a	Add address to Break Table
D	Display Break and Trace Tables

E n	Set Execution Counter to n, where n is a positive integer <250. EO turns off the execution counter.
END	Terminate execution. Return to TCL.
G G a	Go. Starts program at current address, or branches to Address to start.
K a	Kill break-point (Delete address from Break Table).
M	Reverse status of Modal Break Flag and print new status.
N n	Set Break-Point Counter to n. (Inhibit trap until "n" breaks have occurred.)
OFF	Terminate Session (LOGOFF).
P	Reverse status of Print List Flag and print new status.
T a T i	Add Address to Trace Table, or add register (Indirect Address to Indirect Trace Table.)
U a U i	Untrace. Delete Address from Trace Table, or delete register (Indirect Address) from Indirect Trace Table.

Data Display Commands:

Ca;w	Display data in character format
Ci;w	
Xa;w	Display data in hexadecimal format
Xi;w	
Ia;w	Display data in integer format; if w
Ii;w	is not 1, 2 or 4, only one byte will be displayed.

Immediately after the data at the specified address has been displayed, DEBUG prompts with an equal sign (=), and the user has the following options:

(r) Carriage Return -- Terminate display mode; DEBUG will prompt with an !

(l) Line Feed -- Display data in the next "window" (that is, the previously specified Address or Indirect Address is updated according to the currently specified window). The data is displayed on the same line.

N^C Control-N -- Display data in the next window, preceded by the address being displayed, in the format f.d (f in decimal, d in hexadecimal).

P^C Control-P -- Display data in the previous window, preceded by the address being displayed.

Note: On a display using the Indirect Address specification, the line-feed or control-N will cause an automatic crossing of linked frame boundaries if the register being used in the display is in the "linked" format.

Replacing Information:

Data may be altered by entering the new data in one of the following formats, before using one of the above control characters:

.xxxxxxxxxx...	Replaces data with hexadecimal string "xxxxxxx"; the string should contain an even number of hexadecimal digits, and can be up to 80 digits in length.
'ccccccccc...	Replaces data with character string data "ccccc", of up to 80 characters.
n	Replaces data with integer value; in this case, the window must have been 1, 2 or 4.

In the case of the hexadecimal or character string replace, the data actually replaced may extend beyond the currently defined "window".

Tables Provided for Debugging

DEBUG maintains three tables of four elements each: the Break Table, the Trace Table and the Indirect Trace Table. If there are entries in the Break Table, the address of every instruction is compared with the address in the Break Table and a break occurs if there is a match. If there are entries in the Trace or Indirect Trace Tables, the data pointed at by the entries, is printed whenever a break message is printed. Up to four entries can be placed in each of these tables.

Break Messages

DEBUG has the facility to break on intermodal transfers (BSL or ENT instructions). The command acts as an alternate action switch, to change the status from ON to OFF. A break can also be initiated with the

command E, causing a break after executing a specified number of instructions. The following set of standard messages are output, when a break in execution occurs:

Message	Condition
B f.d	Break-point address encountered. (Break Table match).
E f.d	Execution runout (specified number of instructions have been executed).
I f.d	Interrupt (Break key depressed).
M f.d	Modal break (Inter-frame branch - ENT or BSL encountered).
R f.d	Return (RTN) encountered.

where : "f" is the decimal FID and "d" the hexadecimal displacement, representing the location of the execution interruption point.

Note: The execution break and address break facilities are mutually exclusive: When the execution counter is positive, Break Table entries are ignoted. However, the Break Table of the Execution Break facility can be used with the Modal break facility.

Hardware Trap Conditions

Certain error conditions cause the CPU to execute a trap to the DEBUG state; processing of the current program will be aborted, and a message indicating the nature of the trap, and the location at which it occurred, will be displayed. The table below shows these error conditions:

Error No.	Message	Description
0	ILLGL OPCODE	An illegal (undefined) operation code has been found.
1	RTN STK EMPTY	A RTN (return) instruction was executed when the return-stack was empty (current pointer was at X'0184').
2	RTN STK FULL	A BSL or BSLI (subroutine call) instruction was executed when the return-stack was full (current pointer was at X'01B0'); the return-stack has been reset to an "empty" condition before the trap.
3 *	FRM-ID ZERO	An address register has an FID of zero.

4 *	CROSSING FRM LIMIT	An address register in the "unlinked" format 1) has been incremented or decremented off the boundary of a frame, or 2) has been used in a relative address computation that causes the generated relative address to cross a frame boundary.
5 *	FORW LNK ZERO	An address register in the "linked" format has been incremented past the last frame in the linked frame set.
6 *	BACKW LNK ZERO	An address register in the "linked" foramt has been decremented prior to the first frame in the linked frame set.
7	PRIV OPCODE	A Privileged operation code (one executable only in the Monitor mode of operation), has been found while executing in the Virtual mode.
8	ILLGL. FRAME-ID	An address register has an FID that exceeds the maximum value allowable in the current disc configuration.
11	STK FRMT ERR	The Return-stack pointers are in an illegal format - either the ending address is less than X'0184', or the current address is less than X'0184'. The pointers have been reset to an initial condition of X'01B0' and X'0184' respectively.
12	REGISTER ZERO DETACHED	Register zero has been detached by a user-program.

In the case of traps marked with an asterisk (*) in the table above, the following message will also be returned after the message shown:

```
REG = 0.X
```

where "x" is the hexadecimal Address Register number of the register causing the trap condition.

In all cases, the following message will also be returned:

```
ABORT @ f.d
```

where "f" is the decimal Fid of the frame, and "d" the hexadecimal displacement within it, of the location where the trap occurred. This corresponds the location counter in the assembly listing of the corresponding program.

Note that the G (Go) command, without an address specification, cannot be used after a trap to the DEBUG state.

EXAMPLE --

```
:                                                    BREAK key depressed
I 6.87
!X200.12;6(r).1E1327101881=.0123456789ABCDEF N^C     Display and change data.
 200.18  .CDEF34567890=N^C                           Display next window, no
 200.1E  .012C00000064='KJMN P^C                     change.
 200.18  .CDEF34567890=(1).4B4A4D4E0064=N^C          Change data in character
 200.24  .0004000A0000=(r)                           form.
!
!M(r)ON                                              Set Modal Trace on.
!N 5(r)                                              See delay counter.
!T .40(r)                                            Trace location .40 in PCB
!TR4(r)                                              Trace Register four.
!G(r)                                                Go.

:HHHH(r)                                             TCL statement.

R 5.49                                               RTN instruction
 512.40 = .000002060000                              Data from direct trace.
R 0.4 : 528. = .004154545249                         Data from indirect trace.
M 7.3
 512.40 = .000002060000
R 0.4 : 528. = .004154545249
R 5.78
 512.40 = .000020920000
R 0.4 : 528. = .004154545249
M 10.1
 512.40 = .000020920000
R 0.4 : 528. = .004154545249
M 8.1
 512.40 = .000020920000
R 0.4 : 528. = .004154545249
R 10.32
 512.40 = .000020920000
R 0.4 : 528. = .004154545249

!D(r)                                                Display Break table entries.
 BRK TBL: 0.  0.  0.  0.
 TRC TBL: 512.40  0.  0.  0.
%TRC TBL: 0.4  0.  0.  0.                            (Indirect trace table)
!END(r)                                              Terminate Execution.

:                                                    Back to TCL.
```

Section XVIII

SYSTEM MAINTENANCE

INTRODUCTION

This section describes the interaction with the front panel of the CPU, and the system maintenance procedures that are cataloged in the SYSPROG (system programmer's) account. The front panel sense switches are used when a bootstrap and cold-start sequence is to be initiated, as well as to halt the CPU.

Halting the CPU While in Execution

The CPU may be halted from the front panel by depressing either the STEP or the INT switch. Either switch causes the CPU to trap to the Monitor mode (if it was executing in Virtual mode), before halting.

Restarting After STEP/INT Halts

After an INT or STEP halt, the sequence RESET-INT is mandatory; the sense switches should be set appropriately before this sequence (numbered right to left):

All switches up	:	Continue execution
Switches 4 and 1 down	:	Bootstrap/Cold-start sequence with automatic configuration
Switches 4, 2 and 1 down	:	Bootstrap/Cold-start sequence with operator configuration.

Continuing execution after an INT halt may cause terminals that were in the output mode at the time of the INT halt to wait for a break-key interrupt before resuming output.

Operator Notes:

Never depress either the CLOCK or RESET switch while the CPU Run Light is on since this will cause indeterminate errors.

The RUN switch is used only after an address stop which is not covered in this manual and should only be used by a qualified Microdata Customer Engineer.

Bootstrap and Cold-Start Procedure

This section deals with bootstrapping the system from a cold-start magnetic tape.

The process of starting up the system involves initializing core memory from the cold-start tape. The Reality system operates with the monitor software and certain virtual memory tables core-resident. This data, as well as an initial set of software object code is read in from the tape during the bootstrap process.

The bootstrap process is controlled by the sense switches on the front panel of the CPU. Sense switches 4 and 1 control the reading of the bootstrap and the cold-start sections of the tape respectively.

Sense switch 2 controls whether the software is reconfigured by the operator or not. (See below.) The bootstrap switch (1) causes the CPU to read 512 bytes (one tape record) from device 9 into locations 0 through X'IFF'. The coldstart switch (4) causes the boot program to read the next 12 records into locations X'200' through X'18FF', after which the configuration of the system takes place. The boot program then gains control again and reads the next 24 records into locations X'1000' through X'3FFF'. At this time the cold-start process is complete, and process zero, which is the terminal connected to channel zero of the communications device (address X'18') is activated. The system will then output a message requesting the options that determine the next step.

OPTIONS (X/A/AF/F) =

Entering an X terminates the cold-start process; other options are described under File Restore Process.

Using Preset Configuration

To use the preset configuration on a coldstart tape, simply perform a bootstrap/coldstart sequence with sense switch 2 up.

Reconfiguring Software at Coldstart Time

To reconfigure software on a coldstart tape that was preset to any arbitrary configuration, it is necessary to put sense switch 2 down as well as 1 and 4. The first question is DO YOU WANT TO RUN THE DISC DIAGNOSTIC (Y/N). A response of Y gives control to the disc diagnostic covered in separate documentation. A response of N causes the configuration questions to be asked. Any other response causes the same question to be repeated. The following questions are then asked:

NUMBER OF DISC CONTROLLERS (only 1,2,3, or 4 is valid.)

NUMBER OF DISCS PER CONTROLLER (only 1 or 2 is valid)

CORE SIZE (only 16,24,32,40,48,56, or 64 is valid)

MAXIMUM FID (only 9743 through 155903 is valid)

NUMBER OF ENTRIES IN IOQ TABLE (only 2 through 8 is valid)

NUMBER OF DISC READS BEFORE REMOVAL FROM IOQ (any value 4 through 127 is valid)

NUMBER OF COMMUNICATION LINES (only 1 through 32 is valid)

The last question asked is IS CONFIGURATION CORRECT (Y/N). A response of Y will cause the rest of the coldstart tape to be read and the OPTIONS(X/A/AF/F)= message to appear. A response of N will cause all of the configuration questions to be asked again.

It is important to note that all communication between the computer and the operator is done on the terminal on line zero. Also, since the communication is being handled entirely by the monitor and not by the much more sophisticated firmware/virtual software combination, the communication is handled in a much more simplistic manner. None of the normal control characters have their usual effect (i.e., control H does not backspace etc.) All responses are terminated with a non-numeric character which is not considered as part of the response unless it is the character "X". If the character "X" is used to terminate a response, the previous question is asked again.

Programming Notes:

The bootstrap program MBOOT, which is read in by the firmware, first reads in the twelve monitor frames which includes MSETUP0, MSETUP1, MMONITORY/N2, DISC-DIAG and DISC-MSG. Control is then transferred to MSETUP0. If sense switch 2 is down, the program first determines whether the disc diagnostic is to be executed. If it is, MMONITOR and MMONITORX are overlayed by DISC-DIAG and DISC-MSG and control is passed to DISC-DIAG. If the disc diagnostic is not to be executed, the table in MSETUP1 is set up by communicating with the operator before the configurator is activated. If sense switch 2 is up, the table of configuration parameters is used as it comes in from the tape. Control is then passed back to MBOOT which reads the 24 virtual frames from the tape, overlaying MSETUP0, MSETUP1, MMONITORY/N2, DISC-DIAG and DISC-MSG so still only 16K of memory is needed for the coldstart even though there is 18.5K of data on the tape.

MSETUP1 contains a table which has the preset configuration parameters as well as all of the messages which are used to ask the operator questions concerning reconfiguration. MSETUP0 contains the code which communicates with the operator and which actually does the configuration. The program uses the values in the table in MSETUP1 to decide how to configure the system. The reconfiguration is just a matter of changing the values in the table before the configurator is activated. MMONITORY/N2 is the same as MMONITORY/N1 except that it is set up to use two discs per controller instead of one disc per controller and is in a different frame. MMONITORY/N1 is overlayed by MMONITORY/N2 by the configurator if two discs per controller are selected.

Further Explanation of Configuration Parameters

Most of the questions which concern the configuration parameters are self explanatory, but a few of them are not or have additional considerations which may not be obvious.

Maximum FID can be determined from the following table:
(size of system is in megabytes)

Size of System	5	10	15	20	30	40	60	80
Maximum FID	9743	19487	29231	38975	58463	77951	116927	155903

If a number other than those above is entered, one of two things will happen. If the number is larger than that allowable for the size of the system, the monitor will later allow a virtual process to reference a FID which does not exist which will cause that process to go into an infinite loop trying to reference that FID. The break key will get you out of such a loop in most cases. The other possibility is that the number is smaller than allowable for the size of the system. In this case a virtual process may later break with an illegal FID error when referencing a perfectly good FID. The only way to cure either of these problems is to coldstart the system again and supply a valid maximum FID.

The number of entries in the IOQ table can be 2 through 8. This is one of the parameters with which the user can tune his system. The IOQ table determines how many virtual processes can access the disc during the same time period. If a process does not have an entry in the table and the table is full, then that process must wait until another process has been removed from the table before he can be placed there in order to satisfy a frame fault. How many frame faults a process is allowed before he is removed from the table is covered on the next page.

The number of entries should be determined by the number of discs and the amount of core in your system. The following table is meant to be used only as a guide in setting up the number of IOQ entries. Users may find that one entry more or less may give them better response time depending on the use of the system.

		Amount of Core in 1024 Byte Multiples						
		16	24	32	40	48	56	64
No. of Discs	1	2	2	3	3	4	4	5
	2	2	3	4	4	5	5	6
	3	-	3	4	5	6	6	7
	4	-	-	5	6	6	7	8
	6	-	-	-	7	7	8	8
	8	-	-	-	-	8	8	8

The dashes in the above table represent unbalanced systems for which no number exists that would be the correct number of entries in the IOQ.

The number of disc reads before removal from the IOQ is synonymous with the number of frame faults that will be satisfied for a process before it is removed from contention for the disc. This number can range from 4 to 127. It should be chosen based on the number and type of processes a user will be running. Higher numbers give the edge to processes requiring many frames such as sorts, RPG compile and execution, and assemblies. Lower numbers give the edge to processes requiring few frames such as data entry or inquiry. Numbers in the range of 16 to 25 work quite well in a mixed process environment.

The number of communication lines is determined by adding the number of communication ports to the number of phantom processes, such as the print spooler. For example, if you have an 8-way and a 4-way communication board, that is 12 communication ports. If you are going to use the print spooler but no user written phantom processes, that's 13 communication lines all together.

FILE-RESTORE PROCESS

The File-Restore process consists of two sections:

1. Restoring the "Abs", or absolute data image (data which is assigned to particular frames). The extent and number of frames that are restored are fixed, and depends on the Abs parameter that was used when the File-save tape was created.

2. Restoring the files from the tape. All files on the tape will be restored; for a selective or partial restore see Selective Restore.

The above two sections can be done separately or in conjunction, depending on the option entered in response to the OPTIONS message, below:

X : No further action is needed; proceed to LOGON

A : Restore Abs (absolute data image) from the File-Restore tape, except for the following frames:

FID = 34 Overflow space management routine is restored, but overflow space tables are preserved.

FID = 47 Abs loader is not overwritten.

F : Initialize overflow space tables and restore files from the file-restore tape.

AF : Initialize: overflow space management routine, overflow space tables, and restore files from the file-restore tape. If the tape does not contain an Abs section, the files are restored with no indication of missing Abs.

As the system restores files, a message will be returned for each file (dictionary or data file) that is restored, indicating the file-name, and its base, modulo and separation parameters.

An example is shown on the following page.

On a cold-start, all processes other than the one executing the cold-start are set up to an initial execution address in the program TCL-INIT (Frame 4, displacement 1), which results in their going to the LOGON state when activated.

The restoring of files also has the effect of initializing the overflow disc space of the system; at the conclusion of a file restore, all data is grouped together and there is one contiguous block of overflow space extending from the end of file-space to the end of the available disc space.

FILE RESTORE FRAME LIMITS

File restore frame limits may now be entered at the time of the file restore. Just prior to the files portion of the restore (i.e., after the abs portion has completed or skipped), the message, FRAME LIMITS=, will appear on the terminal on which the file restore is being done. A response of (r) will cause the prestored limits on the tape to be used. A response of n-m (r) will cause n to be used as the base of the SYSTEM dictionary and m to be used as the highest frame which can be allocated to files. Four audits are performed on these numbers: n must be greater than or equal to 32 plus the PCB FID of the last communication line set up for the system; m must be greater than n; there must be a "-" between n and m; and no other characters may be entered after the m. If any of these audits fail the message, FRAME LIMITS, is repeated. The prestored limits may be used after any number of audit failures by responding with (r) to the repeated message.

Output From a File-Restore Process

```
OPTIONS (X/A/AF/F) = AF (r)        Response to OPTIONS? REQUEST.
FRAME LIMITS = (r)                 Response causing the limits stored
                                   on the tape to be used.
SSYSTEM^D^01024^0001^011           System dictionary
SDL/ID^D^01024^0001^011            System dictionary
  PACC^D^01035^0001^001            ACC file dictionary
    PDL/ID^D^01037^0005^001        ACCOUNT file
  PPICK^D^01088^0007^001           First user MDICT
```

```
SDL/ID^D^01088^0007^001
SM/DICT^D^01088^0007^001
SMD^D^      ^0007^001
  PACCOUNT-NO^D^01173^0007^001   First user dictionary
  PACCOUNT^D^01173^0007^001      Synonym to above
      PDL/ID^D^01183^0011^001    User data-file
  PBATCH^D^01220^0003^001        Next user dictionary
  SDL/ID^D^01220^0003^001        DL/ID pointing back to dict.
  PINVENTORY^D^01223^0003^001
  PDINV^D^01223^0003^001
  PINV^D^01223^0003^001
      PDL/ID^D^01226^0007^001
  PTSYM^D^01234^0003^001         User dictionary without DL/ID
  PSAMP-FILE^D^01237^0001^001
      PDL/ID^D^01238^0001^001
  PDUMMY^D^01239^0001^001        P element:  file-name updated
      PTTT^D^01240^0001^001      in previous dictionary
      PDL/ID^D^01241^0001^001    Base FID of file
  PDUMMYI^D^01242^0001^001       Modulo of file
      PDL/ID^D^01243^0001^001    Separ. of file
  PTEMP^D^01244^0001^001
      PDL/ID^D^01246^0001^001
  PEMP^D^01247^0001^001
  PDICK^D^01237^0001^001
      PDL/ID^D^01248^0001^001
  PJUNK^D^01249^0001^001
  PSAMPLE-FILE^D^01250^0001^001
      PDL/ID^D^01251^0003^002
  PXPSYM^D^01259^0029^001
  PXTSYM^D^01294^0029^001
  PMC^D^01323^0001^001
      PDL^ID^D^01324^0003^001
  PMACHINE^D^01327^0001^001      Second user M/DICT
PDICT^D^01328^0007^001
PDATA^D^01328^0007^001
PMURTHI^D^01328^0007^001
PMUX^D^01328^0007^001
SDL/ID^D^01328^0007^001          S element; file-name updated
SMD^D^01328^0007^001             into current file
SM/DICT^D^01328^0007^001
      PCSYM^D^01356^0001^001
        PDL/ID^D^01357^0017^002
```

If the prestored limits are used, the lower limit is checked to be certain that it is greater than or equal to 32 plus the PCB FID of the last communication line set up for the system. If it isn't, then the default is used which is the number the lower limit was being checked against.

The following table shows the relationship between the number of communication lines set up for a system and the lowest legal base FID for the SYSTEM dictionary:

No. of Lines	Base FID	No. of Lines	Base FID
1	544	17	1056
2	576	18	1088
3	608	19	1120
4	640	20	1152
5	672	21	1184
6	704	22	1216
7	736	23	1248
8	768	24	1280
9	800	25	1312
10	832	26	1344
11	864	27	1376
12	896	28	1408
13	928	29	1440
14	960	30	1472
15	992	31	1504
16	1024	32	1536

Initial System Setup

At the conclusion of a Cold-Start or a file-restore process there are certain system parameters that must be set to an initial condition. This is accomplished by executing the COLD-START PROC in the SYSPROG M/DICT. This is done by logging on to the SYSPROG:

LOGON PLEASE:SYSPROG (r)

and keying

COLD-START (r) (See Cold-Start PROC, XVIII-10)

SYSPROG Account PROCs and Verbs

The following is a list of special PROCs and verbs that are in the SYSPROG account:

PROCs: COLD-START - To be executed immediately after a cold-start or File-restore only.

CREATE-ACCOUNT - Creates a new user account; places a Q entry in the SYSPROG account to allow access.

FILE-RESTORE	- Initiates the file-restore process from a file-save tape.
FILE-SAVE	- Creates a file-save tape
RE-GEN	- Sets up the configuration parameters prior to creating a coldstart tape.
SETUP-ASSY	- Sets an account up to be able to assemble REAL programs.
SETUP-RPG	- Sets up an account to be able to compile and run RPG-II programs.
START-SPOOLER	- Starts the output spooler if that option was not exercised in the COLD-START proc.
SYS-GEN	- Generates a sys-gen tape.
SYS-LOAD	- Used only when updating the software system with a new system update tape.
SYS-UPDATE	- Used when updating to a new system release from a SYSGEN tape.
UPDATE-ACCOUNT	- Used to update an account to a new system release level.
VERIFY-SYSTEM	- Verifies the source programs against the object code of the system modes.

Verbs:

:DDUMP	- Evokes the file-save program.
:DLOAD	- Evokes the file-restore program
:INIT-LINES	- Sets the number of active lines which the monitor will look at
:INIT-SPOOLER	- Starts the output spooler operation and sets pointers.
:MSETUP	- Set configuration parameters before creation of a coldstart tape.
SET-DATE	- Reset system date
SET-TIME	- Reset system time

:START-SPOOLER - Starts the output spooler operation.

:SWD - Reverses the action of the :SWE verb.

:SWE - Changes "D"D/CODE entries to "E"D/CODE entries.

COLD-START PROC

This PROC should be used immediately after bringing the system up with a cold start tape to set the system time and date, set up the number of active lines on the system, and to optionally start the output spooler. It begins by prompting the user with:

```
THE CURRENT SYSTEM TIME AND DATE IS hh:mm:ss dd mmm yyy
DO YOU WISH TO CHANGE THE TIME OR DATE:  (Y/N)
```

If neither "Y" or "N" is entered in response, the PROC warns of an illegal response and repeats the prompt. If "N" is entered, the PROC skips the prompts for time and date and skips deleting current entries from the accounting file and goes directly to the communication lines section. If "Y" is entered, the user is prompted with TIME. If only a (r) is given, the PROC goes on to the DATE prompt. Otherwise it passes any response given to the SET-TIME verb without auditing the response. Then the DATE prompt is given. If only a (r) is given, the PROC bypasses setting the date. Otherwise it passes any response given to the SET-DATE verb without auditing the response. Next all current entries in the accounting file ACC are deleted. This is done to keep from charging any account an invalid amount of the time due to the change in date or time.

Next the user is prompted with:

```
HOW MANY COMMUNICATION LINES WILL BE ACTIVE ? (1,2,...32)
```

If the user responds with an invalid reply, a warning message is given and the prompt is repeated. Otherwise the response is passed on to the :INIT-LINES verb. This allows you to change the number of active lines within the range 1 through the maximum allowable with the value of the base FID of your system dictionary. Next the START-SPOOLER section is entered which is identical to the START-SPOOLER PROC. Finally the PROC logs the user off. This is done to force him to log back on thus putting an entry for him back into the ACC file.

CREATE-ACCOUNT PROC

This PROC creates a new account by performing the following steps:

1. Creating a new file (dictionary) with the new user name as the file definition item-id; the file definition item is placed in the SYSTEM dictionary.

2. Copying the contents of the NEWAC file, the prototype M/DICT, to the newly created user M/DICT.

3. Adding a file synonym definition entry in the SYSPROG account equated to the new M/DICT, to allow access from SYSPROG.

Usage

The format of the CREATE-ACCOUNT parameter list is as follows:

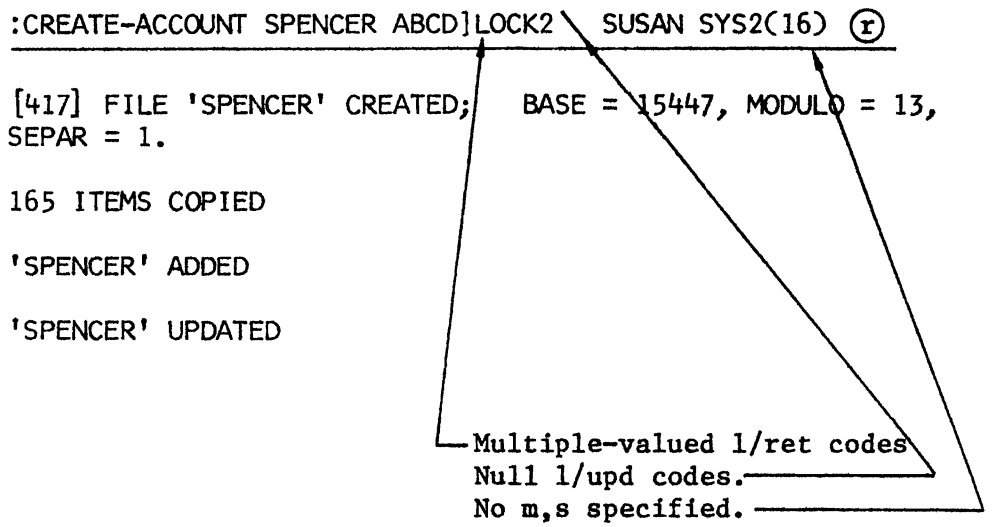
```
:CREATE-ACCOUNT user-name  1/ret-code(s)  1/upd-code(s)  password
                privileges m,s
```

where:

user name	:	is the name of the new account; it cannot contain any commas, but all other characters are legal.
1/ret-code(s)	:	is the set of retrieval lock-codes to be associated with the user; multiple values can be entered separated by the Control-Shift-N character (echoes as]). Described in the sections LOGON and SECURITY.
1/upd-code(s)	:	is the set of update lock codes associated with the user; as for 1/ret-codes.
password	:	is the password associated with the user's LOGON item in the SYSTEM dictionary.
privileges	:	is the code describing the user's privileges and workspace assignment; described in the LOGON section.
m,s	:	are the module and separation parameters for the user M/DICT; numerics separated by a comma.

All parameters except the user-name are optional; null values may be indicated by a backslash (\). Default values for module and separation are 13,1.

EXAMPLE--



The CREATE-ACCOUNT PROC should not be used to create a new synonym to an existent account; this should be done by using the Editor to create the file synonym definition entry in the SYSTEM dictionary.

FILE-RESTORE PROC

This PROC calls :DLOAD to do a file restore from a file-save tape. It first prompts the user with HAS EVERYONE LOGGED OFF? (Y/N) and will only accept either Y or N as a response to indicate yes or no. The reason for this is that :DLOAD will cause everyone to be logged off regardless of their current activity and any additional work space associated with those logons will be lost. For this reason it is advisable to always log on to a synonym of SYSPROG which requests no additional work space before using this PROC. The :DLOAD verb then enters the file restore process described earlier in this chapter.

FILE-SAVE PROC

The File-save PROC is called by the verb FILE-SAVE. This may be used to create a tape containing a dump of the executable object code in the system (Abs dump), and the data from all files in the system, or optionally a coldstart tape or all three.

The File-save processor uses the entry "MM/DICT" in the SYSPROG account to pick up a parameter associated with the Abs dump-portion of the tape, as well as to pick up the file pointers of the SYSTEM dictionary. For this reason, the MM/DICT entry in SYSPROG must be a file synonym definition item, which equates to the SYSTEM dictionary. Attribute seven of the entry contains a code described below:

$$\begin{matrix} C \\ A \end{matrix}\Big] \; n_1\text{-}n_2, n_3\text{-}n_4 \cdots\cdots$$

where C specifies that a Cold-start section is being dumped preceding the Abs dump; if A is specified, the cold-start section will not be dumped. The sets of numeric parameters following represent FID ranges which are to be dumped in the Abs section of the tape. If only one FID is to be dumped in any set, the "-n" may be omitted; if the entire set of numeric parameters is omitted (therefore the code is A or C only), no Abs section is created. Attribute eight of the entry contains the code below:

F
X f_1-f_2

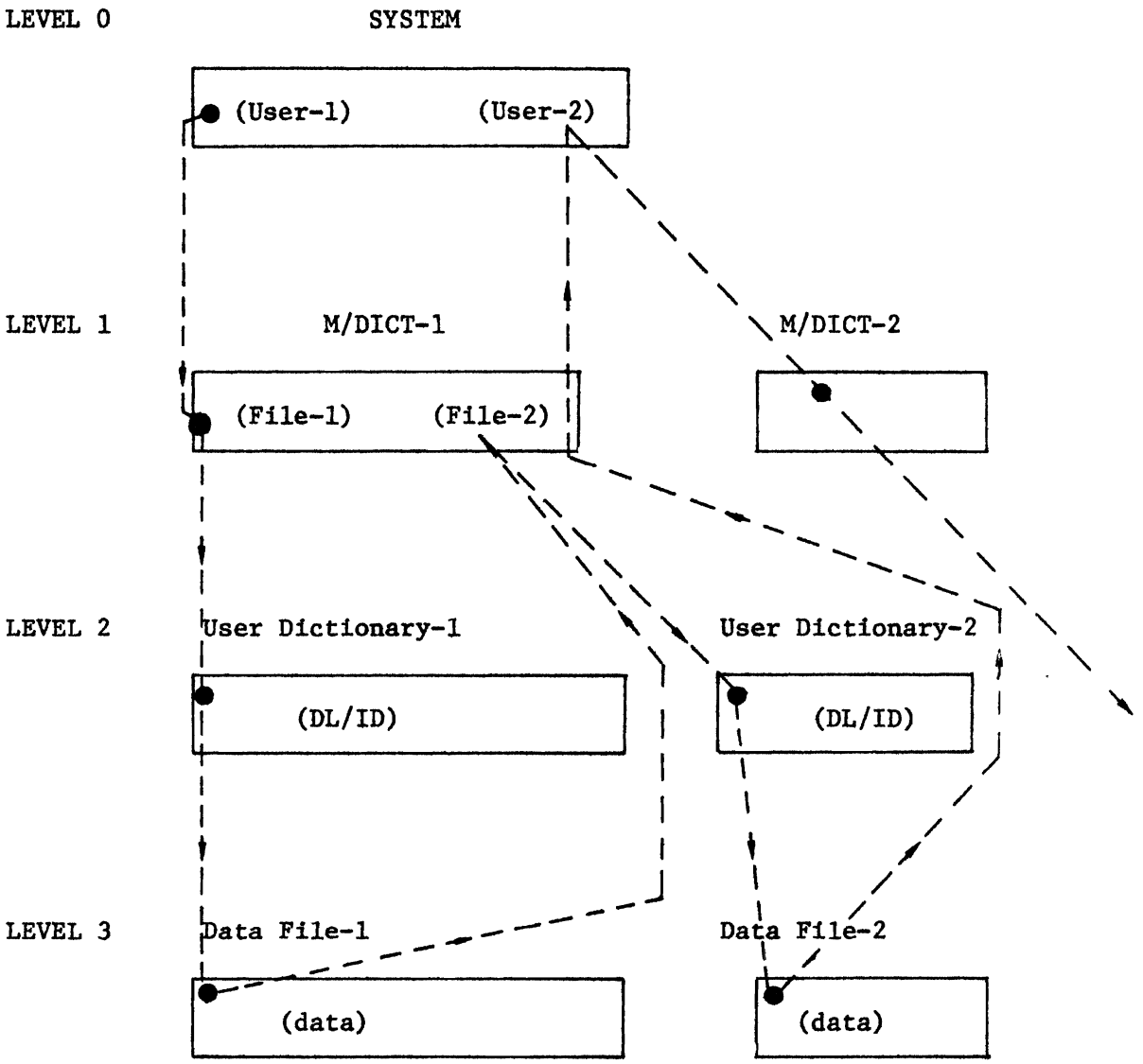
The F specifies a file dump section is to be created after the Abs sections. If X is used, the files are not dumped.

The values f_1 and f_2 are FID limits for the FILE-RESTORE process; the table under FILE RESTORE FRAME LIMITS in this chapter may be consulted.

Method of Operation

The File-save processor uses entries in the SYSTEM dictionary to search for and dump data. At each of the levels 0, 1 and 2 (see following map), the respective dictionary is searched for file defining items (D/CODE"D"), and the file at the next lower level with the lowest base FID is dumped; then the one with the next lowest FID, and so on, until all files defined from that dictionary have been dumped. Then the process continues at the next higher dictionary level. Thus the contents of a user's M/DICT (except for file definition items themselves) are dumped preceding the contents of each of his dictionary-file combinations, in ascending base FID sequence. The sequence in which the user accounts (M/DICTs) were created determines the sequence of user accounts dumped.

The dashed lines in the map represent the path taken by the File-save processor while dumping data from each file. Note that the process stays at each level till all files at the next level have been dumped before going up one level. Dots are intended to represent the dumping of data within the corresponding file.



Output From the File-Save Process

```
:FILE-SAVE (r)
[91] ATTACHED
[94] END OF FILE
TOP
007 C1-402,421,453
008 F1024-18887
-XIT-
ABS DUMP SPECS ? (r)
DISK INIT. SPECS ? (r)
9:41:24  16 AUG 1974
```

```
TAPE LABEL IF DESIRED FILE-SAVE/032 (r)
.... START DISK DUMP ....                          <-- Level-0 File (MM/DICT)
D0^1024^1^11^
  SSYSTEM^
  SDL/ID^
  D1^1035^1^1^                                     <-- Account file and ACC Dictionary
  PACC^
    D2^1037^5^1^
    PDL/ID^
  D1^1043^13^1^                                    <-- Level-1 File (user M/DICT)
  PSYSPROG^
    SM/DICT^
    SMD
    SDL/ID^
    SPROC^
    D2^1093^17^3^                                  <-- Level-2 File (user Dictionary)
    PERRMSG^
      SDL/ID^
    D2^1148^13^1^
    PNEWAC^
    D2^1172^1^1^
    PSYSTEM-MODES^
      D3^1173^37^5^
      PDL/ID^
    D2^2120^1^1^
    PSYSGEN-FILE^
      D3^2121^17^5^
      PDL/ID^
    D2^2386^1^1^
    PMODE-UPDATE^
      D3^2387^13^1^
      PDL/ID^
    D2^2400^29^2^                                  <-- Level 2 File with DL/ID
    POSYM^                                             pointing back to dictionary
      SDL/ID^
    D2^2458^29^2^
    PPSYM^
      SDL/ID^
    D2^2516^1^1^
     :
     :
```

Output from the File-save process is in the following formats

(1) File Header format

Dn^BASE^MODULO^SEPAR^ 0<n<3

where n indicates the level (0 through 3) of the file being processed; base is the base FID of the file (this is not used as the base FID when doing a File-Restore); modulo and separ are the actual values from the file definition item unless Reallocation has been specified, in which case they represent the reallocation values.

(2) P and S Code Formats

```
Pfile-name^00000^000^000^...  REMAINDER OF ITEM ...^

Sfile-name^00000^000^000^...  REMAINDER OF ITEM ...^
```

These indicate the file definition entries that are to be recreated on the File-Restore process; a P-code entry is to be created in the preceding file one level higher; an S-code entry is to be created in the current file itself, on the File-restore. For instance, preceding the dump of a user M/DICT, his user name will have a P-code entry (to be updated into the SYSTEM dictionary) and one or more S code entries (usually M/DICT and DL/ID, to be updated into his M/DICT itself).

Operator Use of FILE-SAVE PROC

When the FILE-SAVE PROC is activated, it first attaches to the magnetic tape unit, writes an EOF on the tape which is currently mounted, and then rewinds the tape to load point. It then displays lines 7 and 8 of the MM/DICT entry in the SYSPROG M/DICT. The significance of these lines was described earlier. The PROC will then prompt for ABS DUMP SPECS, whereupon the operator may change line 7 of the MM/DICT entry by keying in the new value. A carriage return alone will leave line 7 of MM/DICT unchanged. Next the PROC will prompt for DISK INIT. SPECS which applies to line 8 of MM/DICT, which can be updated in the same fashion as described for line 7. The PROC now prints the system time and date and prompts for TAPE LAVEL IF DESIRED. Whatever is entered at this point is passed to :DLOAD as a tape label. If only a carriage return is entered, an unlabeled tape is created.

RE-GEN PROC

The RE-GEN PROC is used to preset the configuration parameters before creating a coldstart tape. This is necessary whenever an abs restore overlays the MSETUP1 mode with parameters not consistent with your system. (i.e. an abs restore from a tape created on the same system would not necessitate running RE-GEN but an abs restore from a SYSGEN tape would.)

The actual presetting operation is performed by the MSETUP program which is described elsewhere. The proc, RE-GEN, is used to call the MSETUP program so a user need not refer to documentation everytime the procedure is necessary. RE-GEN prompts the user for all of the information which is necessary to preset the configuration. To use the RE-GEN proc it is first necessary to log on to SYSPROG and then simply key in RE-GEN (r). The proc first prompts you with the question: ARE MONITOR MODES LOADED? (Y/N) only Y or N will be accepted as legal responses. If the response is N, the following modes are loaded from SYSTEM-MODES: MBOOT, MBUFFERS, MMONITOR, MMONITORX, MMONITORY/N1, MMONITORY/N2, MMONITORZ, MSETUP0, MSETUP1, PIB0, PIB1, PCB0, ABSD, and MSETUP. The proc next prompts for all of the configuration parameters.

An example follows:

```
:RE-GEN (r)
ARE MONITOR MODES LOADED (Y/N)?Y (r)
NUMBER OF DISC CONTROLLERS 1(r)
NUMBER OF DISCS PER CONTROLLER 1(r)
CORE SIZE 48(r)
MAXIMUM FID 19487(r)
NUMBER OF ENTRIES IN IOQ TABLE 3(r)
NUMBER OF DISC READS BEFORE REMOVAL FROM IOQ 25(r)
NUMBER OF COMMUNICATION LINES 9(r)
```

SETUP-ASSY PROC

```
:SETUP-ASSY account-name (r)
```

This PROC sets an account up to be able to assemble REAL programs. If the account name is missing the PROC will exit. However, PROCs cannot verify the validity of an account name so whatever name is given to it will be used. First a file named TTSYM is created with the file definition item in "SYSPROG M/DICT. This "D" item is converted to an "E" item by calling :SWE. The "E" item is copied to the user's M/DICT renaming it to TSYM and deleting it from the SYSPROG M/DICT. The TSYM entry in the user's M/DICT is then converted back to a "D" item by calling the editor with appropriate commands to the editor placed in the output stack. This PROC also copies the AS verb into the user's M/DICT and creates "Q" entries for OSYM and PSYM which point to the OSYM and PSYM defined in the SYSPROG M/DICT. The user's LOGON entry in the SYSTEM dictionary is then updated to have the same L/RET and L/UPD locks codes as SYSPROG, SYS2 privileges, and 25 frames of linked work space. For this reason, SETUP-ASSY should only be used when no one else is logged onto the system.

SETUP-RPG PROC

```
:SETUP-RPG Account-Name (r)
```

This PROC sets an account up to be able to compile and run RPG-II programs. Further details on this PROC are available in the REALITY RPG-II manual.

START-SPOOLER PROC

This PROC is used to start the output spooler in case that option was not exercised in the COLD-START proc. It is identical to that portion of the COLD-START PROC which is dedicated to the output spooler. It prompts the user with:

> WHICH COMM. LINE IS DEDICATED TO THE PRINT-SPOOLER? (NONE, 0,1...31) HOW MANY PAGES ARE TO BE EJECTED AT THE END OF A LISTING? (0,1,2,3,4,5) IS THE PRINT QUEUE STATUS TO BE RESET TO EMPTY? (Y/N) SEPARATE YOUR ANSWERS WITH BLANKS.

If the user replies "NONE" the PROC exits immediately. If an invalid reply is given for anyone of the three necessary replies, the PROC admonishes the user about the first invalid answer detected and repeats the prompt. Either :START-SPOOLER or "INIT-SPOOLER is called depending on whether "N" or "Y" was the reply to the question concerning setting the print queue status to empty. See the discussion about "INIT-SPOOLER and :START-SPOOLER for further details and cautions.

Hold file queue entries are destroyed if the answer "Y" is given to the START-SPOOLER PROC.

SYS-GEN PROC

Since this PROC changes with each release of the system it will not be described here in detail. Also, since it is completely self-explanatory and automatic no detailed instructions are necessary to run it.

The SYS-GEN PROC is used to create a SYSGEN tape from which either an initial system load or a system update may be performed. The SYSGEN tape will always consist of a coldstart section, an abs section, a files sections, and a T-DUMP section of system files. The first EOF mark on the tape is immediately after the files section and an EOF occurs after each T-DUMP file.

During the creation of the coldstart, abs, and files sections of the tape, the system cannot be used. This is due to the fact that most of the "D" D/CODE entries in the SYSTEM dictionary and the SYSPROG M/DICT have been changed to "E" D/CODEs so the associated files will not be saved in the files section. The "E" D/CODE entries will be saved, however. The SYS-LOAD PROC will remove them when an initial system load is performed. At the end of the files section the "E" D/CODEs are restored to "D"'s and the system can then be used again.

SYS-LOAD PROC

This PROC must always be executed after the file restore from a SYSGEN tape to load the system files. Since its use and operation are peculiar to each release of the system, detailed instructions will not be given here.

SYS-UPDATE PROC

This PROC must always be executed as part of an update to a new release of the system from a SYSGEN tape. It is used to update system files. Since its use and operation are peculiar to each release of the system, detailed instructions will not be given here.

UPDATE-ACCOUNT PROC

This PROC is used to update accounts after a SYS-UPDATE to reflect changes to verbs, PROCs, and other elements of a M/DICT. It should never be used in an attempt to update the SYSPROG account, which is always done separately as part of the system update procedure. Since the use and operation of this PROC is peculiar to each release of the system, detailed instructions will not be given here.

VERIFY-SYSTEM PROC

The PROC verifies the source programs against the object code of the system modes. Typically a few modes will have one or two bytes mismatched. This is usually nothing to be alarmed at. If in doubt, the mode can be MLOADed from the SYSTEM-MODES file and re-verified.

Special SYSPROG Verbs

:DDUMP Verb

:DDUMP is a TCL-I verb which evokes the file-save program. This verb is used by the FILE-SAVE PROC after the MM/DICT entry has been verified and/or updated. It requires that the mag tape be attached and online. It uses the MM/DICT entry (attributes 7 and 8) in the SYSPROG M/DICT to determine what it is supposed to do. If the MM/DICT entry is invalid error message 990 or 991 is returned. Any characters keyed in with the verb will be used as a tape label, otherwise an unlabeled tape will be created.

:DLOAD Verb

:DLOAD is a TCL-I verb which evokes the file-restore program. This verb is used by the FILE-RESTORE PROC. It causes the OPTIONS message, etc. as after a coldstart from mag tape. It should be noted that any additional work space acquired at LOGON time is lost when this verb is used.

:INIT-LINES Verb

:INIT-LINES is a TCL-I verb which sets the number of active lines which the monitor will look at. Zero, negative numbers, and numbers which are too large for the system are refused with error message 330. The program determines whether the number is too large by the FID of the SYSTEM dictionary. It subtracts the PCB FID for line zero from the FID of the SYSTEM dictionary and divides by 32 (the number of work frames for each process) to determine the maximum number of lines acceptable. The stop bit in the PIB status of the appropriate line is then zeroed and all other stop bits in the status bytes of the PIBs of all lower lines are then set. This verb is called by the COLD-START PROC.

:INIT-SPOOLER Verb

:INIT-SPOOLER is a TCL-I verb which does all of the same things that :START-SPOOLER does plus it also sets the pointers to the beginning and end of the print queue to zero. This verb is called from the COLD-START or the START-SPOOLER PROC when the user requests the print queue to be set to the empty status. If the line number is not valid, error message 330: "ILLEGAL LINE NUMBER" is printed and control returns to TCL.

:MSETUP Verb

:MSETUP is a TCL-I verb which allows a user to preset the configuration parameters before the creation of a coldstart tape. This verb is called by the PROCs RE-GEN and SYS-GEN. It is called from TCL as follows:

```
:MSETUP cn, dn, cs, fmax, ioq, imax, ncl
```

where the parameters are decimal numerics:

cn	Number of disc controllers; must be 1 through 4
dn	Number of discs per controller; must be 1 or 2
cs	Core size in kilobytes; must be 16 through 64 and a multiple of 8
fmax	Maximum FID; must be 9743 through 155903
ioq	Number of entries in IOQ table, must be 2 through 8
imax	Maximum number of disc reads before removal from IOQ table; must be 4 through 127
ncl	Number of communication lines; must be 1-32

Parameters that are to remain unchanged need not be specified (may be null). If a range error occurs, processing is terminated, and the message:

[329] PARAMETER RANGE ERROR AT : xxx

will be returned.

:START-SPOOLER Verb

:START-SPOOLER is a TCL-I verb which is called either by the COLD-START PROC or the START-SPOOLER PROC. Its function is to start the output spooler operating on some line defined by the user and to set the number of page ejects which will automatically occur whenever the output spooler finishes the last print file in its queue.

Both this verb and the :INIT-SPOOLER verb should be used with extreme caution. Never start the spooler on a different line than the one it is currently running on without doing a coldstart form tape first. Remember that the line numbers start from zero; therefore, if eight lines are active, the spooler cannot run on line eight (really the ninth line). Never start the spooler while it is actively printing since overflow frames will be lost by doing so. The spooler may only be started on lines 0 through 31. The general practice for systems with 4, 8 or 12, etc., communication lines is to run the spooler on line 4, 8, or 12, etc. (one line past the last available communication line). This cannot be done on a 32 line system.

If the line number is not valid, error message 330: "ILLEGAL LINE NUMBER" is printed and control returns to TCL.

Only accounts with SYS2 privileges are allowed to execute :INIT-SPOOLER or :START-SPOOLER.

:SWD Verb

:SWD is an ENGLISH verb which is called by the SYS-GEN PROC. It is used to reverse the action of the :SWE verb after a :DDUMP has been executed.

:SWE Verb

:SWE is an ENGLISH verb which is called by the SYS-GEN proc. It is used to change "D" D/CODE entries in the SYSTEM dictionary and the SYSPROG M/DICT to "E" D/CODE entries except for a few selected entries. The change is performed in place. This is done to cause the file-save on the SYS-GEN tape to contain only the SYSTEM dictionary, the ACC file, and the SYSPROG M/DICT. :SWE is also called by the SETUP-ASSY PROC so that a D item may be copied from the SYSPROG M/DICT to a user M/DICT.

Standard SYSPROG PROCs

The following PROCs are standard, and are available to the user logged on to SYSPROG.

ADDS

This PROC sets the terminal characteristics for an ADDS 580 CRT terminal 79,23,1,3,1,21.

CHOO-CHOO

This PROC prints a picture of a choo choo train on your terminal... just for fun.

COMPILE

This PROC prints the starting and ending times of an RPG II compile and invokes the compiler.

CT

Copies an item or items from the file specified to the terminal.

DEL-OBJ

Deletes RPG object text from object space by releasing the space and then writing an "E" entry in RPG-object.

DELETE

Deletes an item from the file specified or if no item specified prompts for as many items as are desired to delete.

EXEC

This PROC invokes the RPG-II run time executive.

FILE-RESTORE

This PROC initiates a file restore process or an absolute load depending upon the options selected.

FILE-SAVE

Displays the ABS and file specs in the MM/DICT entry and allows them to be modified before saving the system on magnetic tape.

FTC

This PROC sets the tabs for RPG calculation specifications after calling the tab editor.

FTE

This PROC sets the tabs for RPG extension specifications after calling the tab editor.

FTF

This PROC sets the tabs for RPG file specifications after calling the tab editor.

FTH

This PROC sets the tabs for RPG header specifications after calling the tab editor.

FTI

This PROC sets the tabs for RPG input specifications after calling the tab editor.

FTL

This PROC sets the tabs for RPG line counter specifications after calling the tab editor.

FTO

This PROC sets the tabs for RPG output specifications after calling the tab editor.

LISTACC

List all accounting data or if account names are specified only data for those accounts (including total charges for services.)

LISTCONN

Sorts all connectives in any account dictionary and lists them on the terminal or the line printer if LPTR is specified.

LISTDICTS

Sort all attribute or synonym definitions in any dictionary and list them on the terminal or the lineprinter if LPTR specified.

LISTFILES

Sorts all "D" and "Q" items in any account dictionary and lists them on the terminal or on the lineprinter if LPTR is specified.

LISTPROCS

Sorts all PROCs in any account dictionary and lists them along with a brief abstract on the terminal or the lineprinter if LPTR is specified.

LISTU

Lists the account name of all users currently active on the system with their logon time and channel number.

LISTVERBS

Sorts all verbs (not PROCs) in any account dictionary and lists them on the terminal or on the lineprinter if LPTR is specified.

LP106

Sets the terminal characteristics so listings will be formatted properly on data products lineprinter (106 columns wide).

LP132

Sets the terminal characteristics so listings will be formatted properly on data products lineprinter (132 columns wide).

PRINT-TAPE

Spools one file from mag tape to the line printer.

RPG-CLEAN

Prints the status of each frame of object text creates and deletes file 'RPGC' to hold the data

RPG-DUMP

Dumps formatted object text

XREF

Clears the XSYM file then cross references from file CSYM to XSYM then sorts XSYM.

Section XIX

SYSTEM MESSAGES

MESSAGE NUMBER	MESSAGE
1	ILLEGAL USE OF DOUBLE-QUOTE IN ITEM-ID
2	UNEVEN NUMBER OF SINGLE OR DOUBE QUOTE-SIGNS (' ")
3	VERB?
4	ILLEGAL RATE REQUEST
5	VERB 'verb' IS ILLEGAL
6	FILE NAME 'file-name' IS ILLEGAL
7	A HEADING TEXT LINE MUST FOLLOW THE "HEADING" CONNECTIVE
8	'DICT' MODIFIER MISSING
10	FILE NAME MISSING
13	'DL/ID' MISSING.
15	THE FILE-NAME IS PRECEDED BY AN ILLEGAL CONNECTIVE
16	"TO" VALID ONLY IN ADD STMNT
17	"WITHIN" VALID ONLY IN COUNT/LIST STMNTS
18	LAST WORD MAY NOT BE A CONNECTIVE.
19	VALUE WITHOUT ATTRIBUTE NAME IS ILLEGAL.
20	MEANINGLESS ITEM-ID IN STMNT.
21	CONFLICT BETWEEN USER & SYSTEM NAMES.
22	"TO" BEFORE ID VALID ONLY IN CHANGE STMNTS
23	"TO" FOLLOWED BY ID MUST ALSO BE PRECEDED BY AN ID
24	THE WORD "word" CANNOT BE IDENTIFIED.
25	"WITH" MAY NOT IMMEDIATELY PRECEDE A VALUE.
30	FORMAT ERROR IN M/DICT ENTRY DEFINING VERB.
32	A REQUIRED ITEM-ID IS MISSING.

MESSAGE NUMBER	MESSAGE
42	ILLEGAL MULTIPLE CONNECTIVES EXIST.
65	THE WORD TO MAY NOT PRECEDE BOTH THE DATA LIST NAME AND ATTRIBUTE
71	THE ATTRIBUTE "attribute-name" HAS AN ILLEGAL CONNECTIVE
72	THE VALUE "value" IS MEANINGLESS
75	'parameter' FAILS TO PASS ITS MAXIMUM SIZE RESTRICTIONS
77	'parameter' FAILS TO PASS ITS TYPE RESTRICTIONS
78	'parameter' FAILS TO PASS ITS PATTERN AUDIT
80	A SYSTEM ERROR HAS OCCURRED IN MODE: mode-name
81	SECURITY CODE VIOLATION
82	YOUR SYSTEM PRIVILEGE LEVEL IS NOT SUFFICIENT FOR THIS STATEMENT
90	DETACHED
91	ATTACHED
92	IN USE!
93	ATTACH THE TAPE UNIT
94	END OF FILE
95	NOT ON-LINE
96	BOT
97	EOT
98	PARITY ERROR!
99	FILE PROTECTED!
111	ITEM 'item-id' IS NOT ON FILE
117	A DELETE STATEMENT MUST CONTAIN EITHER ITEM-IDS OR SELECTION CRITERIA.
120	'value' NEGATIVE BALANCE NOT PERMITTED
136	'value' DOES NOT MATCH THE G-CORRELATIVE SPECS

MESSAGE NUMBER	MESSAGE
158	AN ILLEGAL CONNECTIVE OF THE FORM "Cx" MODIFIES "value"
163	ATTRIBUTE FOR SORT-KEY MISSING
200	FILE NAME?
201	"file-name" IS NOT A FILE NAME
202	'item-id' NOT ON FILE
203	ITEM NAME?
204	FILE DEFINITION 'file-name' IS MISSING
205	NO STATEMENTS TO BE ASSEMBLED
206	'item-id' ASSEMBLED
207	UNDEFINED SYMBOLS
208	ERROR IN ITEM-ID LIST
209	ERROR IN OPTION LIST
210	FILE 'file-name' IS ACCESS PROTECTED
211	NO ASSEMBLED CODE CAN BE FOUND
212	"FRAME" STATEMENT MISSING
213	LOCATION COUNTER SPECIFICATION ERROR AT STATEMENT NO. n
214	THE MODE EXCEEDS THE MAXIMUM SIZE OF 512 BYTES AT STATEMENT NUMBER n
215	ERROR IN HEX CODE DESIGNATION AT STATEMENT NO. n
216	'item-id' LOADED ON FRAME #n SIZE = m (DEC), x(HEX)
217	MODE 'item-id' VERIFIED
218	MODE 'item-id' HAS n BYTES OBJECT CODE MIS-MATCHES
220	-XIT-
221	'item-id' FILED
222	'item-id' DELETED
230	CARD READER NOT READY

MESSAGE NUMBER	MESSAGE
231	CARD READER MECHANICAL ERROR
232	CARD READER EBCDIC ERROR
233	CARD READER HOPPER EMPTY
234	ITEM SIZE EXCEEDS 32,000 BYTES
240	SYSTEM OVERFLOW NOT IN INITIAL CONDITION; NO RPG FRAMES ASSIGNED
241	TOO MANY RPG FRAMES REQUESTED; NONE ASSIGNED
242	ILLEGAL RPG FRAME REQUEST FORMAT; NO FRAMES ASSIGNED
268	THE DESTINATION OF THE PROC "GO" STATEMENT: 'statement' CANNOT BE FOUND.
269	INPUT BUFFER OVERFLOW AT PROC STATEMENT: 'statement'
270	FORMAT ERROR IN THE PROC STATEMENT: 'statement'
271	ONE PROC CANNOT CALL ANOTHER
272	A VALUE EXISTS FOR THE ATTRIBUTE REFERENCED BY THE ELEMENT: 'element'
273	ERROR IN COLUMN-NUMBER/FIELD-WIDTH OR FORMAT SPECIFICATION AT: 'statement'
274	UNRECOGNIZABLE BATCH-STRING element: 'element'
275	Y OR F SUB-ELEMENT ERROR AT BATCH-STRING ELEMENT: 'element'
276	D-2 UPDATE WITHOUT D-1 BEING SPECIFIED. AT BATCH-STRING ELEMENT: 'element'
277	J ELEMENT MISSING AT BATCH-STRING ELEMENT: 'element'
278	ERROR IN PROCESSING SECONDARY BATCH-STRING ELEMENT: 'element'

MESSAGE NUMBER	MESSAGE
279	INCORRECT SCALING FACTOR IN F* BATCH-STRING ELEMENT: 'element'
280	FILE-DEFINITION BATCH ELEMENT ERROR AT: 'element'
281	D1 MUST HAVE Y1 or Y4 STORAGE CORRELATIVE ...ERROR, AT: 'element'
282	DATA INPUT LINE TO BATCH AFTER A SELECT MUST CONTAIN AT LEAST ONE ITEM-ID SUBSTITUTION CODE (ASTERISK *)
283	END OF PRINT QUEUE
284	INPUT ENTRY NO. MUST BE 1-32
285	ENTRY IS NOT A HOLD FILE
286	NO. OF LINES MISSING
287	FILE BUSY BEING PRINTED
288	NO. OF PAGE EJECTS GREATER THAN 10
289	LINE PRINTER ALREADY ATTACHED TO LINE 'line'
290	STRING NOT FOUND IN HOLD FILE
291	'file-name' FILE-DEFINITION IS MISSING
298	FORMAT ERROR IN SPECIFICATIONS
329	PARAMETER OR RANGE ERROR AT: address
330	ILLEGAL LINE NUMBER
331	THE ACCOUNT FILE IS MISSING
333	THE FORMAT OF THE ADDITONAL WORK-SPACE PARAMETER: parameter IS ILLEGAL FOR THIS ACCOUNT NAME. ADDITIONAL WORK-SPACE HAS NOT BEEN ASSIGNED.
334	REQUESTED NUMBER OF ADDITIONAL WORK-SPACE FRAMES: parameter IS NOT AVAILABLE: ADDITIONAL WORK-SPACE HAS NOT BEEN ASSIGNED.
335	***WELCOME TO MICRODATA REALITY***
	time date

MESSAGE NUMBER	MESSAGE
336	CONNECT TIME = m CHARGE-UNITS = n LOGGED OFF AT time ON date
337	USER IS NOT LOGGED ON
339	IMPROPER OR UNDEFINED FORMAT FOR DATE CONVERSION
401	NO ITEMS PRESENT
403	END OF LIST
404	n ITEMS SELECTED
405	n ITEMS LISTED
407	n ITEMS COUNTED
408	ONE ITEM COUNTED
409	OVERFLOW DISK SPACE IS INSUFFICIENT FOR THIS SORTED LISTING - NUMBER OF ITEMS EXTRACTED AT THIS POINT WAS n
410	A SYNONYM (Q-TYPE) FILE CANNOT BE SPECIFIED IN THIS STATEMENT
411	"DICT' OR 'DATA' MUST BE SPECIFIED IN A CLEAR-FILE STATEMENT
412	INSUFFICIENT DISK SPACE AVAILABLE FOR THE FILE
413	THE FILE NAME ALREADY EXISTS IN THE MASTER DICTIONARY
414	ILLEGAL OR MISSING MODIFIER USED IN DEFINING THE FILE AREA(S)
415	"item-id' EXISTS ON FILE
416	RANGE ERROR IN MODULO OR SEPARATION PARAMETER
417	FILE 'file-name' CREATED: BASE = base; MODULO = modulo; SEPAR = separation
418	FILE-DEFINITION ITEM 'item' WAS NOT COPIED
419	THE SPECIFIED FILE CANNOT BE CLEARED OR DELETED!
420	DICTIONARY FILE DELETION CANNOT BE DONE WITHOUT DELETION OF DATA FIRST
421	STATISTICS OF attribute-name: TOTAL = t; AVERAGE = a; COUNT = c.

MESSAGE NUMBER	MESSAGE
423	TOTAL OF attribute-name IS: t
520	NO DATA FOR BLOCK OUTPUT
521	TOO MANY CHARACTERS IN WORD TO BLOCK
522	BLOCK CONVERT FILE MISSING OR IMPROPERLY DEFINED
523	BLOCK OUTPUT WOULD EXCEED PAGE WIDTH
524	INPUT CHARACTER 'x' IS NOT IN BLOCK CONVERT FILE
525	INPUT CHARACTER 'x' IS IMPROPERLY FORMATTED IN BLOCK CONVERT FILE
700	THE FORMAT OF THE F-CORRELATIVE IS INCORRECT
701	INVALID FUNCTION CORRELATIVE DEFINITION: function-definition
705	ILLEGAL CONVERSION CODE : conversion-code
706	THE TRANSLATE CONVERSION CODE: 'code' IS ILLEGAL
707	DL/ID ENTRY FOR T-CONVERSION : conversion-code NOT FOUND
708	'item-id' CANNOT BE CONVERTED
709	CONFLICT IN T-CONV DEFINITION: REQUESTS VERIFY BUT THERE IS A NULL AMC
711	VALUE value WAS NOT TRANSLATED BY T-CONVERSION
780	'item-id' NOT ON FILE.
781	'item-id' ADDED
782	'item-id' UPDATED
783	'item-id' DELETED
785	'value' CHGD TO 'value'
800	<n>ITEMS DUMPED
802	n ITEMS DUMPED
803	n ITEMS LOADED

MESSAGE NUMBER	MESSAGE
805	n ITEMS COPIED
810	'file-name' NOT ON HDR FILE
990	ERROR IN ABSOLUTE DUMP SPECIFICATION IN MM/DICT ENTRY
991	MM/DICT ENTRY MISSING. OR REQUIRED SPECIFICATION(S) MISSING
1004	ITEM 'item-id' IS NOT ON FILE
1006	ITEM 'item-id' EXISTS ON FILE
9999	****** END OF OVERFLOW DISK SPACE ******
LOGON	(variable)
&	Disc Error (Available only with SYS2 privileges)

Section XX

SYSTEM SOFTWARE

INTRODUCTION

Assembly level programming in the REALITY system is facilitated by a set of system subroutines that allow easy interaction with the disc file structure, terminal i/o, and other routines. These subroutines work with a standard set of addressing registers, storage registers, tallys, character registers, bits, and buffer pointers, collectively called functional elements. In order to use any of these routines, therefore, it is essential that the calling routine set up the appropriate functional elements as required by the called routine's Input Interface.

The standard set of functional elements is pre-defined in the permanent symbol file (PSYM), and is therefore always available to the programmer. Included in the PSYM are all the mode-id's (program entry points) for the standard system subroutines. There is no reason that a symbol internal to an assembly program cannot have the same name as a PSYM-file symbol, if the PSYM-file symbol is not also referenced in that program; such symbolic usage cannot be a "forward" reference in the assembly program. To avoid confusion, however, it is best to treat the entire set of PSYM symbols as reserved symbols.

Address Registers

All data reference in the system is made indirectly through one of the sixteen address registers (A/R). Registers zero and one have special, firmware-defined meaning; the other fourteen may be considered general-purpose registers, with the limitation that system software conventions determine the usage of most A/R's. Registers zero and one should never be changed in any way by assembly programs. Register two always points to the SCB after the debugger has been entered.

Register zero always addresses byte zero of the process' PCB; register one always addresses byte zero of the frame in which the process is currently executing. Thus all elements in the PCB may be relatively addressed using register zero as a base register; this includes the individual segments of the address registers themselves (e.g. R15WA, referencing the word-address segment of R15). Address registers can thus be setup explicitly by setting up their segments appropriately; the more conventional way of setting up an A/R is to move a S/R into it. For example, the sequences below are functionally identical.

```
FRM100    ADDR    0,X'100'        DEFINE A LITERAL S/R
          .                       REFERENCING FRAME X'100'
          .
          .
          MOV     FRM100,R15

and       .
          .
          ZERO    R15WA
          ZERO    R15DSP
          MOV     =DX'80000100',R15FID
```

It is important to note that, in the first sequence, the address register is automatically set to the "detached" format when the "MOV" instruction executes; in the second sequence, the address register is explicitly set to the "detached" format by the "ZERO R15WA" instruction. The word=address of an A/R must be zeroed before other segments of the A/R are modified.

Attachment and Detachment of A/R's

All instructions that reference data force "attachment" of the A/R(s) used in the reference. Not all instructions do the same; for example, the "increment A/R by tally" instruction will not cause a "detached" A/R to attach before execution.

This point may lead to programming errors; consider the following sequence:

```
          .
          .
          .
L1    BCU   AM,R6,NXT     R6 "ATTACHED" AT THIS POINT"
L2    INC   R6, SIZE      R6 MAY "DETACH" DUE TO THIS INSTRUCTION
L3    MOV   R6, SR4       SAVE R6
          .
          .
          .
```

The instruction at L2 may force R6 to "detach" (if the contents of SIZE are such that the resultant address is beyond the limits of the current frame); storing R6 in SR4 will then cause SR4 to have a large positive displacement, and a FID equal to that in R6 at the time of execution of the instruction at L1. Subsequently, a register comparison instruction of the form:

```
          BE      R15,SR4,L20
```

may execute incorrectly due to the fact that if the FID's of R15 and SR4 are unequal at the time of execution, it is assumed that the two frames are contiguously linked (See Branch Register Equal/Unequal instructions in Section XVI). Therefore, it is best to force "attachment" of R6 before L3; a convenient way of doing so is to execute the instruction:

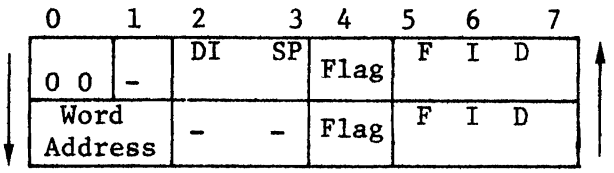
```
L3A    FAR    R6,0
```

though any data reference instruction would serve as well.

The following table summarizes the attachment/detachment process:

ATTACHMENT & DETACHMENT OF ADDRESS REGISTERS

A/R is Attached when:	A/R is Detached when:
(1) Any instruction that references data via the A/R is executed.	(1) Process is deactivated due to: terminal I/O; disk I/O; peripheral I/O; timer run-out; monitor call.
(2) Execution of INC r DEC r instructions.	(2) A S/R is moved to the A/R.
(3) Execution of FAR r,n instruction.	(3) Execution of INC r,t DEC r,t if a frame boundary is crossed.



Re-entrancy

In practically all cases, the system software is re-entrant, that is, the same copy of the object code may be used simultaneously by more than one process. For this reason, no storage internal to the program

is utilized; instead the storage space directly associated with a process is used; this is part of the process' Primary, Secondary, Debug (or Tertiary) and Quadrenary Control blocks. The Primary Control Block (PCB) is addressed via address register zero; the SCB via register two. The Debug Control Block is used solely by the Debug Processor as a scratch area, and should not be used by any other programs. The Quadrenary Control Block has no register addressing it; it is used by some system software (magnetic tape routines, for example) which temporarily setup a register pointing to it; its use is reserved for future software extensions.

A user program may utilize storage internal to the program if it is to be used in a non-re-entrant fashion; however in most cases it will be found that the functional elements defined in the PSYM will be sufficient.

In some cases it may be required to setup a program to be executable by only one process at a time; that is, the code is "locked" while a process is using it, and any other process attempting to execute the same code waits for the first process to "unlock" it. The following sequence is typical:

```
         ORG    0
         TEXT   X'01'              INITIAL CONDITION FOR LOCK BYTE
         CMNT                      (NOTE USAGE OF STORAGE INTERNAL TO
         CMNT                       PROGRAM)
         .
         .
LOCK     MCC    X'00',R2           SET "LOCKED" CODE AT R2
         XCC    R2,R1              EXCHANGE BYTES AT R2 AND R1
         BCE    R2,X'01',CONTINUE  OK TO CONTINUE; PROGRAM IS NOW LOCKED
         RQM                       WAIT (RELEASE QUANTUM)
         B      LOCK               TRY AGAIN
         .
         .
         .
UNLOCK   MCC    X'01',R1           UNLOCK PROGRAM
```

Work-spaces or buffers

There is a set of work-spaces, or buffer areas, that is pre-defined and available to each process. If the system conventions with regard to these buffers are maintained, they should prove adequate for the majority of assembly programming. There are three "linked" buffers, or work-spaces, of equal size, symbolically called the IS, the OS, and the HS. These are at least 3000 bytes in length each; more space for each area can be assigned to a process at LOGON time. There are five other work-spaces, the BMS, CS, AF, IB and the OB which may vary between 50 and 140 bytes in length, and are all in one frame. There is the TS, a one-frame unlinked work-space of 512 bytes, and the PROC work-space, 2000 bytes in length which is used normally by the PROC processor alone;

finally there are four additional frames (PCB+28 through PCB+31) that are unused by the system, and are freely available. PCB+28 is used internally by the RPG processor.

Each work-space is defined by a beginning pointer and an ending pointer, both of which are storage registers (S/R's). When the process is at the TCL level, all these pointers have been set to an initial condition. At other levels of processing, the beginning pointers should normally be maintained; the ending pointers may be moved by system or user routines. The address registers (A/R's) that are named after these work-spaces (IS,OS,AF,etc.) need not necessarily be maintained within their associated work-spaces; however, specific system routines may reset the A/R to its associated work-space. The table below discusses these points for each work space. Note that, conventionally, a buffer beginning pointer addresses one byte before the actual location where the data starts. This is because data is usually moved into a buffer using one of the "move incrementing" type of instructions, which increment the A/R before the data movement.

Work-space	Location (offset from PCB)	Size	Linked?	Remarks
BMS	4 (disp.=0)	50	No	Normally contains an item-id when communicating with the disc file i/o routines. Typically, the item-id is copied to the BMS area, starting at BMSBEG+1. BMSBEG may be moved to point within any scratch area. BMSEND normally points to the last byte of the item-id. BMS (A/R) is freely usable except when explicitly or implicitly calling a disc file i/o routine.
AF	4 (disp.=50)	50	No	This work-space is not used by any system subroutine, though the AF A/R is used as a scratch register.
CS	4 (disp.=100)	100	No	As above.
IB	4 (disp.=200)	≤140	No	Is used by the terminal input routines to read data. IBBEG may be moved to point within any scratch area before use. IBEND conventionally points to the logical end of data. IB A/R is freely usable except when explicitly or implicitly calling a terminal input routine.

Work-space	Location (offset from PCB)	Size	Linked?	Remarks
OB	4 (disp.=201 + IBSIZE)	140	No	Is used by the terminal output routines to write data. OBBEG & OBEND should not be altered; they always point to the beginning and end of the OB area. OB A/R conventionally points one before the next available location in the OB buffer.
TS	5	511	No	This work-space is not used by the system subroutines, though the TS A/R is used as a scratch register.
PROC	6-9	2000	Yes	Used exclusively by the PROC Processor for working storage. User-exits from Proc's may change pointers in this area.
HS	10-15	3000+	Yes	Used as a means of passing messages to the WRAPUP processor at the conclusion of a TCL statement. May be used as a scratch area if there is no conflict with the WRAPUP history-string formats. HSBEG should not be altered; HSEND conventionally points one byte before the next available location in the buffer (initial condition is HSBEG=HSEND).

Work-space	Location (offset from PCB)	Size	Linked?	Remarks
IS OS	16-21 22-27	3000+	Yes	These work-spaces are used interchangeably by some system routines since they are of the same size (and are equal in size to the HS). Specific usage is noted under the various system routines. ISBEG and OSBEG should not be altered, but may be interchanged if necessary. Initial condition is that ISEND and OSEND point 3000 bytes past ISBEG & OSBEG respectively (not at the true end if additional work-space is assigned at LOGON time). IS & OS A/R's are freely usable except when calling system subroutines that use them.

Defining a separate buffer area

If it is required to define a buffer area that is unique to a process, the unused frames PCB+28 through PCB+31 may be used (Note that PCB+28 is used by RPG). The following sequence of instructions is one way of setting up an A/R to a scratch buffer:

```
      .
      .
      .
MOV     R0,R15
ZERO    R3WA          SET R3 "DETACHED"
ZERO    R3DSP         INITIALIZE DISPLACEMENT FIELD
INC     R3FID,29      SET R15 TO PCB+29
      .
      .
      .
```

Register 3 can now be used to reference buffer areas, or functional elements that are addressed relative to R3. None of the system subroutines use R3, so that a program has to setup R3 only once in the above manner. However, exit to TCL via WRAPUP will reset R3 to PCB+3.

Usage of XMODE

In several cases, the multiple-byte move instructions can be used (say, when building a table) even when it is not known whether there is enough room in the current linked frame set to hold the data. Normally, if the end of a linked frame set is reached, DEBUG is entered with a "forward link zero" abort condition. However, the tally XMODE may be setup to contain a mode-id of a user-written subroutine that will gain control under such a condition. This subroutine can then process the end-of-frame condition, and, by executing a 'RTN' instruction, normal processing will continue. Instructions then can be handled by this scheme are: INC register; MCI; MIC; MII; MIID; SCD; MIIR. Care should be taken in the case of MIIR to save register R15 in the subroutine. MIIT cannot be handled because DEBUG destroys the accumulator in the process of transferring control via XMODE.

For example:

```
          .
          .
          .
          MOV     XXX,XMODE       SETUP XMODE FOR NEXT INSTRUCTION
          MII     R12,R13,SR4     COPY FROM R12 TO R13, TILL R12=SR4
          ZERO    XMODE
          .
          .
          .

!XXX      EQU     *               ENTRY POINT FOR SUBROUTINE
          MOV     R15, SR1        SAVE R15
          SRA     R15, ACF        SET TO SAVE REGISTER NUMBER
          BCE     X'0D',R15,OK    ENSURE TRAP WAS DUE TO R13
          MOV     0,XMODE         PREVENT DEBUG RE-ENTRY
          ENT     5,DB1           NO! : REENTER DEBUG TO PRINT
          CMNT                    "FORWARD LINK ZERO" MESSAGE

*
OK        MOV     500,R13DSP      RESET DISPLACEMENT FIELD OF R13, SINCE
          CMNT                    FIRMWARE HAS LEFT IT IN A STRANGE STATE.

*         HANDLE END-OF-FRAME CONDITION HERE

          MOV     R13FID,RECORD   SETUP INTERFACE FOR ATTOVF
          BSL     ATTOVF          GET ANOTHER FRAME FROM OVERFLOW

          MOV     SR1,R15         RESTORE R15
          RTN                     RETURN TO CONTINUE EXECUTION OF
                                  MII INSTRUCTION.
```

Initial Conditions

At any level in the system, the following elements are assumed to be setup; they should not be altered by any programs:

MBASE	D	Contain base-FID, modulo and separation of the M/DICT associated with the process.
MMOD	T	
MSEP	T	
USER	T	Contains the low-order 16 bits of the base-FID of the M/DICT.

Special PSYM elements

Certain elements have a "global" significance to the system; in addition to those described above they are:

Functional Element	Description
Arithmetic condition flags:	These are altered by any arithmetic instruction, as well as the branch instructions that compare two relatively addressed fields.
ZROBIT	Set if result of operation is zero (equal).
NEGBIT	Set if result of operation is negative.
OVFBIT	Set if arithmetic overflow resulted.
H0 through H7	Overlays accumulator and extension; H7 is high-order byte of D1; H0 is low-order byte of D0.
INHIBIT	If set, the "BREAK" key on the terminal is inhibited; used by processes that should not be interrupted.
OVRFLCTR	See WRAPUP for usage.
RSCWA	Return-stack current word address; contains address one byte past current entry in stack; stack is null if RSCWA=X'184'.
RSEND	Return-stack ending address; contains address one byte past last allowable entry in stack; for a stack depth of 11 entries, RSEND=X'1B0'.
SYSPRIV1	If set indicates system privileges, level one.
SYSPRIV2	If set in addition to SYSPRIV1, indicates system privileges, level two.

Functional Element	Description
T0 through T3	Overlays accumulator and extension.
XMODE	This tally may be setup to a mode-id of a subroutine that is to gain control when a "forward link zero" condition occurs.

Program Documentation Conventions

In the following documentation, the functional description briefly describes the action taken by the routine. Unless otherwise specified, the program described is called as a subroutine, using the BSL instruction, and it returns to the calling program via a RTN (return) instruction.

The Input Interface, Internal Usage, and Output Interface sections describe the elements used by the subroutine. The single letter following the element name describes its type (C=character, D=double word, H=half word, R=address register, S=storage register, T=word). Unless otherwise specified, it should be assumed that the following elements may be internally destroyed by the routine:

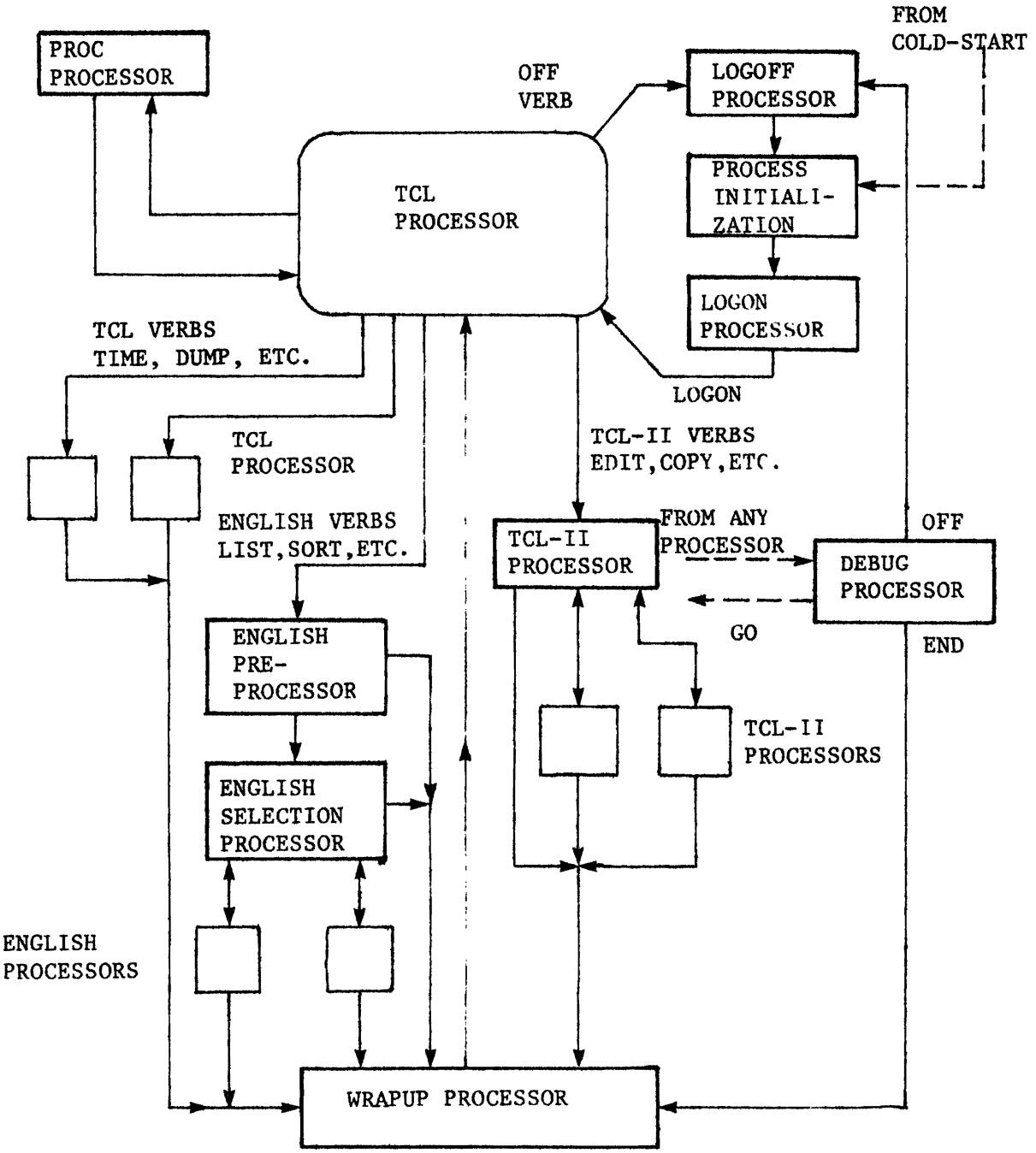
Registers	:	R14 and R15.
Storage Registers	:	SYSR0, SYSR1, SYSR2.
Tallies	:	Accumulator (D0, D1), D2, T4, T5.
Bits	:	Arithmetic condition flags, SB60, SB61.

If no description follows the element name, it indicates that the element is used as a scratch element.

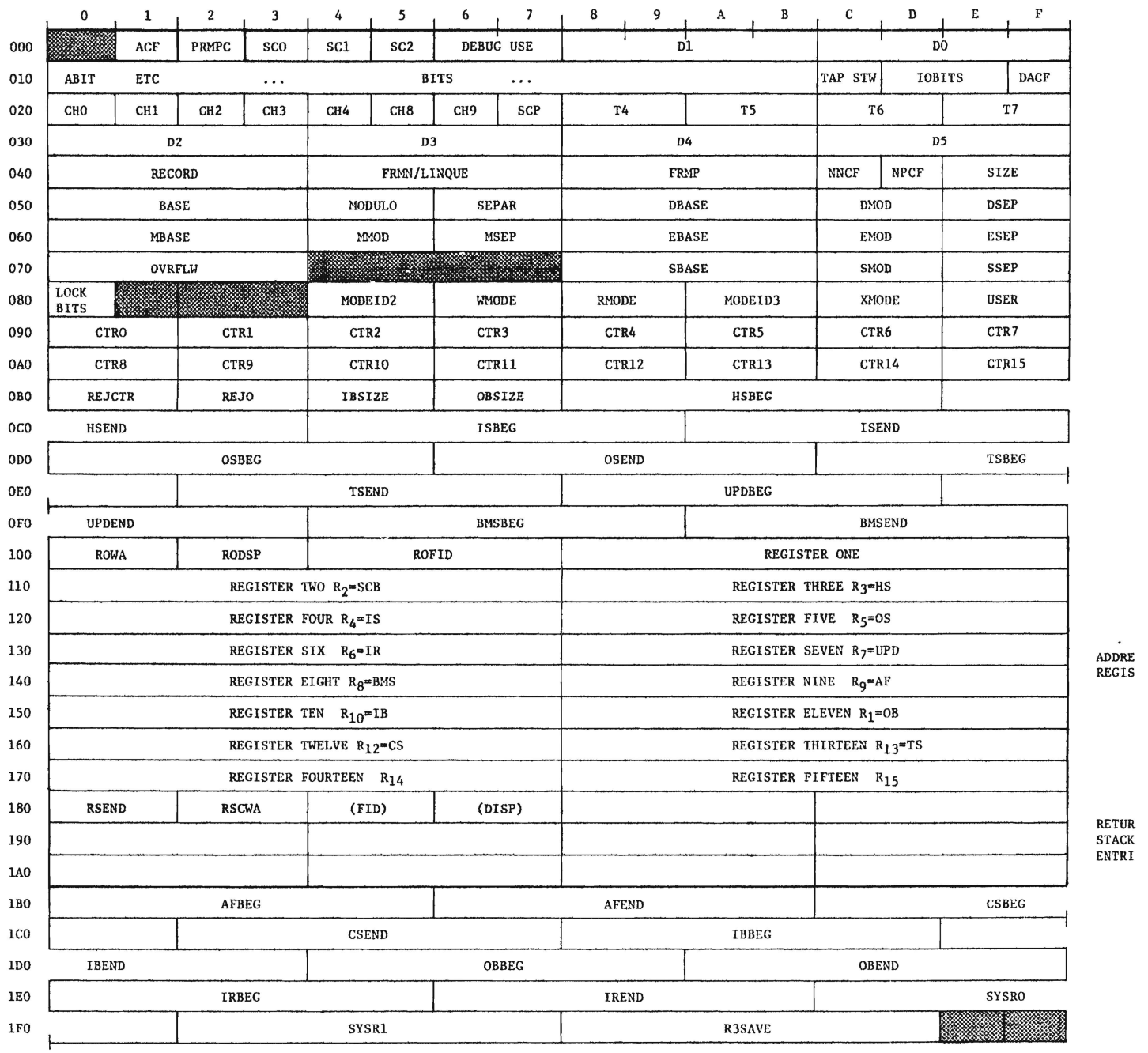
The system delimiters are symbolically referred to as:

Hex. value	Name and description	
FF	SM	Segment Mark.
FE	AM	Attribute Mark.
FD	VM	Value Mark.
FC	SVM	Secondary Value Mark.
FB	SB	Start Buffer.

OVERALL VIEW OF SYSTEM SOFTWARE LINKAGE



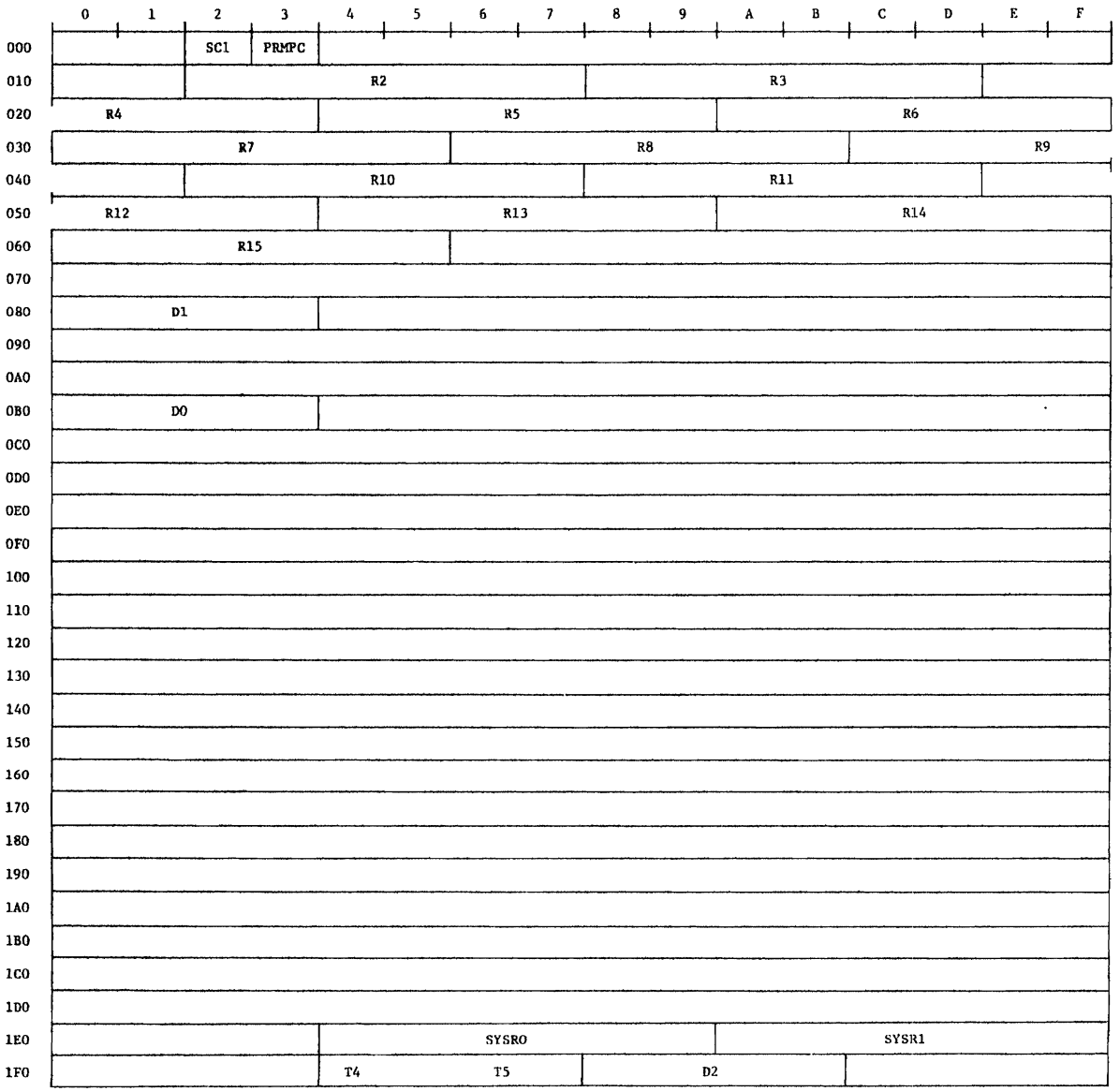
PRIMARY CONTROL BLOCK - Addressing-register RO set to PCB.
Areas bordered by heavy lines are accessed by hardware.
Shaded areas are reserved for future system software use.



SECONDARY CONTROL BLOCK - Adressing-register R2 set to SCB.
SCB = PCB +1

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
000		BSP	C1		C2		C3		C4		C5		C6		C7	
010	C8		C9		CTR16		CTR17		CTR18		CTR19		CTR20		CTR21	
	CTR22		CTR23		CTR24		CTR25		CTR26		CTR27		CTR28		CTR29	
030	CTR30		CTR31		CTR32		CTR33		CTR34		CTR35		CTR36		CTR37	
040	CTR38		CTR39		CTR40		CTR41		CTR42							
050			NEXT		FP1						FP2					
060	FP3						D6				D7				D8	
070			D9				REJ1		REJ2							
080	SYSR2						NXTITM						S0			
090			S1						S2							
0A0	S3				S4						S5					
0B0	S6						S7						S8			
0C0			S9						SR0							
0D0	SR1				SR2						SR3					
0E0	SR4						SR5						SR6			
0F0			SR7						SR8							
100	SR9				SR10						SR11					
110	SR12						SR13						SR14			
120			SR15						SR16							
130	SR17				SR18						SR19					
140	SR20						SR21						SR22			
150			SR23						SR24							
160	SR25				SR26						SR27					
170	SR28						SR29						PQBEG			
180					PQCUR				PQEND							
190	STKINP						STKBEG						SR35			
1A0	LOCKSR						ASBEG						ASEND			
1B0			ASTR						II							
1C0	IIBEG				IIEND						PBUFBEG					
1D0	PBUF						PBUFEND						OVFLCTR			
1E0					CHARIN				CHAROUT				LINESOUT			
1F0	PAGNUM		PAGHEAD						LINCTR		PAGSIZE		PAGSKIP		LFCTR	

DEBUG CONTROL BLOCK - (PCB save areas)
DCB = PCB +2



PSYM.....	D/CODE..	LINE 2	LINE 3
ABIT	B	080	0
ACF	H	001	0
AF	R	009	9
AFBEG	S	0D8	0
AFDSP	T	0A5	0
AFEND	S	0DB	0
AFFID	D	0A6	0
AM	N	0FE	0
ASBEG	S	0D3	2
ASEND	S	0D6	2
ASTR	S	0D9	2
ATTACH	B	0F2	0
B0	B	07F	0
B13	B	072	0
B15	B	070	0
B30	B	061	0
B31	B	060	0
B4	B	07B	0
BASE	D	028	0
BBIT	B	081	0
BITS	C	010	0
BKBIT	B	09B	0
BMS	R	008	8
BMSBEG	S	07A	0
BMSDSP	T	0A3	0
BMSEND	S	07D	0
BMSFID	D	0A2	0
BMSWA	T	0A0	0
BSPCH	C	001	2
C1	T	001	2
C2	T	002	2
C3	T	003	2
C4	T	004	2
C5	T	005	2
C6	T	006	2
C7	T	007	2
C8	T	008	2
C9	T	009	2
CBBIT	B	0D0	0
CBIT	B	082	0
CCDEL	B	0EB	0
CH0	C	020	0
CH1	C	021	0
CH2	C	022	0
CH3	C	023	0
CH4	C	024	0
CH8	C	025	0
CH9	C	026	0
COLHDRSUPP	B	0CB	0
CR	N	00D	0
CS	R	00C	C
CSBEG	S	0DE	0
CSDSP	T	0B1	0
CSEND	S	0E1	0
CSFID	D	0B2	0
CSWA	T	0B0	0
CTR	T	048	0
CTR0	T	048	0
CTR1	T	049	0
CTR10	T	052	0
CTR11	T	053	0
CTR12	T	054	0
CTR13	T	055	0
CTR14	T	056	0
CTR15	T	057	0
CTR16	T	00A	2
CTR17	T	00B	2
CTR18	T	00C	2
CTR19	T	00D	2
CTR2	T	04A	0
CTR20	T	00E	2
CTR21	T	00F	2
CTR22	T	010	2
CTR23	T	011	2
CTR24	T	012	2
CTR25	T	013	2
CTR26	T	014	2
CTR27	T	015	2
CTR28	T	016	2
CTR29	T	017	2
CTR3	T	04B	0
CTR30	T	018	2
CTR31	T	019	2
CTR32	T	01A	2
CTR33	T	01B	2
CTR34	T	01C	2
CTR35	T	01D	2
CTR36	T	01E	2
CTR37	T	01F	2
CTR38	T	020	2
CTR39	T	021	2
CTR4	T	04C	0
CTR40	T	022	2
CTR41	T	023	2
CTR42	T	024	2
CTR5	T	04D	0
CTR6	T	04E	0
CTR7	T	04F	0
CTR8	T	050	0
CTR9	T	051	0

PSYM.....	D/CODE..	LINE 2	LINE 3	PSYM.....	D/CODE..	LINE 2	LINE 3
D0	D	006	0	H2	H	00D	0
D1	D	004	0	H3	H	00C	0
D2	D	018	0	H4	H	00B	0
D3	D	01A	0	H5	H	00A	0
D4	D	01C	0	H6	H	009	0
D5	D	01E	0	H7	H	008	0
D6	D	033	2	HBIT	B	087	0
D7	D	035	2	HDRSUPP	B	0C9	0
D8	D	037	2	HS	R	003	3
D9	D	039	2	HSBEG	S	05C	0
DACF	H	01F	0	HSEND	S	05F	0
DAF1	B	0D1	0	H7	H	009	0
DAF10	B	0C7	0	IB	R	00A	B
DAF2	B	0D2	0	IBBEG	S	0E4	0
DAF3	B	0D3	0	IBEND	S	0E7	0
DAF4	B	0D4	0	IBFID	D	0AA	0
DAF5	B	0D5	0	IBIT	B	088	0
DAF6	B	0D6	0	IBSIZE	T	05A	0
DAF7	B	0D7	0	IBWA	T	0A8	0
DAF8	B	0D8	0	IDSUPP	B	0C6	0
DAF9	B	0D9	0	II	S	0DC	2
DATEQ	T	040	0	IIBEG	S	0DF	2
DBASE	D	02C	0	IIEND	S	0E2	2
DBIT	B	083	0	INDEBUG	B	OF3	0
DBLSPC	B	0CA	0	INHIBIT	B	0F0	0
DEBUG5	B	035	0	INHIBITSV1	B	0DA	0
DEBUG6	B	036	0	INHIBITSV2	B	0DB	0
DEBUG7	B	037	0	IOBIT14	B	0F6	0
DEBUGBYTE	H	006	0	IOBIT2	B	0EA	0
DISKERR	B	0F6	0	IOBIT4	B	0EC	0
DMOD	T	02F	0	IR	R	006	6
DSEP	T	02F	0	IRBEG	S	0F0	0
EBASE	D	034	0	IRDSP	T	099	0
EBIT	B	084	0	IREND	S	0F3	0
ECONVBIT	B	0CF	0	IRFID	D	09A	0
EMOD	T	036	0	IRWA	T	098	0
ENDBIT	B	09A	0	IS	R	004	4
EOFBIT	B	03A	0	ISBEG	S	062	0
EOTBIT	B	0E1	0	ISDSP	T	091	0
ESEP	T	037	0	ISEND	S	065	0
FBIT	B	085	0	ISFID	D	092	0
FP1	D	02A	2	ITAPEBIT	B	OCE	0
FP2	D	02D	2	JBIT	B	089	0
FP3	D	030	2	KBIT	B	08A	0
FRMN	D	022	0	LBIT	B	088	0
FRMP	D	024	0	LF	N	00A	0
GBIT	B	086	0	LFCTR	T	0FF	2
GMBIT	B	09C	0	LFDLY	T	0FF	2
H0	H	00F	0	LINCTR	T	0FC	2
H1	H	00E	0	LINESOUT	D	0F6	2

PSYM.....	D/CODE..	LINE 2	LINE 3
LINQUE	D	022	0
LISTFLAG	B	0F1	0
LOCK	H	000	0
LOCKSR	S	0D0	2
LPBIT	B	0CD	0
MBASE	D	030	0
MBIT	B	08C	0
MMOD	T	032	0
MODEID2	T	042	0
MODEID3	T	045	0
MODULO	T	02A	0
MSEP	T	033	0
NBIT	B	08D	B
NEGBIT	B	00E	0
NEXT	T	029	2
NNCF	H	04C	0
NOBIT	B	09A	0
NOBLNK	B	0EE	0
NPCF	H	04D	0
NREC	D	020	2
NXTITM	2	043	2
OB	R	00B	B
OBBEG	S	0EA	0
OBDSP	T	0AD	0
OBEND	S	0ED	0
OBFID	D	0AE	0
OBIT	B	08E	0
OBSIZE	T	05B	0
OBWA	T	0AC	0
OS	R	005	5
OSBEG	S	068	0
OSBEGF	D	069	0
OSDSP	T	095	0
OSEND	S	06B	0
OSFID	D	096	0
OSWA	T	094	0
OVFBIT	B	00F	0
OVRFLCTR	D	0EE	2
OVRFLW	D	038	0
OVRFLWO	Z	006	E
PAGFRMT	B	0CC	0
PAGHEAD	S	0F9	2
PAGINATE	B	0F7	0
PAGNUM	T	0F8	2
PAGSIZE	T	0FD	2
PAGSKIP	T	0FE	2
PARITY	B	0E4	0
PBIT	B	08F	0
PBUF	S	0E8	2
PBUFBEG	S	0E5	2
PBUFEND	S	0EB	2
PQBEG	S	0BE	2
PQCUR	S	0C1	2
PQEND	S	0C4	2
PQFLG	B	0DE	0
PRMPC	C	002	0
PROTECT	B	0E0	0
PTIME	T	025	2
QBIT	B	090	0
QSTR	S	0E8	2
RO	R	000	0
RODSP	T	081	0
ROFID	D	082	0
ROWA	T	080	0
R1	R	001	1
R10	R	00A	A
R10FID	D	0AA	0
R10WA	T	0A8	0
R11	R	00B	B
R11DSP	T	0AD	0
R11FID	D	0A3	0
R11WA	T	0AC	0
R12	R	00C	C
R12DSP	T	0B1	0
R12FID	D	0B2	0
R12WA	T	0B0	0
R13	R	00D	D
R13DSP	T	0B5	0
R13FID	D	0B6	0
R13WA	T	0B4	0
R14	R	00E	3
R14DSP	T	0B9	0
R14FID	D	0BA	0
R14WA	T	0BC	0
R15	R	00F	F
R15DSP	T	0BD	0
R15FID	D	0BE	0
R15WA	T	0BC	0
R2	R	002	2
R2DSP	T	089	0
R2FID	D	08A	0
R2WA	T	088	0
R3	R	003	3
R3DSP	T	08D	0
R3FID	D	08E	0
R3SAVE	S	0FC	0
R3WA	T	08C	0
R4	R	004	4
R4DSP	T	091	0
R4FID	D	092	0

PSYM.....	D/CODE..	LINE 2	LINE 3	PSYM.....	D/CODE..	LINE 2	LINE 3
R4WA	T	090	0	SB11	B	0AB	0
R5	R	005	4	SB12	B	0AC	0
R5DSP	T	095	0	SB13	B	0AD	0
R5FID	D	096	0	SB14	B	0AE	0
R5WA	T	094	0	SB15	B	0AF	0
R6	R	006	6	SB16	B	0B0	0
R6DSP	T	099	0	SB17	B	0B1	0
R6FID	D	09A	0	SB18	B	0B2	0
R6WA	T	098	0	SB19	B	0B3	0
R7	R	007	0	SB2	B	0A2	0
R7DSP	T	09D	0	SB20	B	0B4	0
R7FID	D	09E	0	SB21	B	0B5	0
R7WA	T	09C	0	SB22	B	0B6	0
R8	R	008	8	SB23	B	0B7	0
R8DSP	T	0A1	0	SB24	B	0B8	0
R8FID	D	0A2	0	SB25	B	0B9	0
R8WA	T	0A0	0	SB26	B	0BA	0
R9	R	009	9	SB27	B	0BB	0
R9DSP	T	0A5	0	SB28	B	0BC	0
R9FID	D	0A6	0	SB29	B	0BD	0
R9WA	T	0A4	0	SB3	B	0AE	0
RBIT	B	091	0	SB30	B	0BE	0
RECORD	D	020	0	SB31	B	0BF	0
REJO	T	059	0	SB32	B	0CO	0
REJ1	T	03B	2	SB4	B	0A4	0
REJ3	T	03D	2	SB5	B	0A5	0
REJ4	T	03E	2	SB6	B	0A6	0
REJ5	T	03F	2	SB60	B	0DC	0
REJCTR	T	058	0	SB61	B	0DD	0
RMBIT	B	09E	0	SB7	B	0A7	0
RMODE	T	044	0	SB8	B	0A8	0
RNIBIT	B	033	0	SB9	B	0A9	0
RNICTR	H	007	0	SBASE	D	03C	0
RSCWA	T	0C1	0	SBIT	B	092	0
RSEND	T	0C0	0	SC0	C	003	0
RTNSTK	T	0C2	0	SC1	C	004	0
S0	S	046	2	SC2	C	005	0
S1	S	049	2	SCP	C	027	0
S2	S	04C	2	SEPAR	T	02B	0
S3	S	04F	2	SIZE	T	027	0
S4	S	052	2	SM	N	0FF	0
S5	S	055	2	SMBIT	B	09F	0
S6	S	058	2	SMCONV	B	0ED	0
S7	S	05B	2	SMOD	T	03E	0
S8	S	05E	2	SR0	S	064	2
S9	S	061	2	SR1	S	067	2
SB	N	0FB	0	SR10	S	082	2
SB0	B	0A0	0	SR11	S	085	2
SB1	B	0A1	0	SR12	S	088	2
SB10	B	0AA	0	SR13	S	08B	2

PSYM.....	D/CODE..	LINE 2	LINE 3
SR14	S	08E	2
SR15	S	091	2
SR16	S	094	2
SR17	S	097	2
SR18	S	09A	2
SR19	S	09D	2
SR2	S	06A	2
SR20	S	0A0	2
SR21	S	0A0	2
SR22	S	0A6	2
SR23	S	0A9	2
SR24	S	0AC	2
SR25	S	0AF	2
SR26	S	0B2	2
SR27	S	0B5	2
SR28	S	0B8	2
SR29	S	0BB	2
SR3	S	06D	2
SR35	S	0CD	2
SR4	S	070	2
SR5	S	073	2
SR6	S	076	2
SR7	S	079	2
SR8	S	07C	2
SR9	S	07F	2
SSEP	T	03F	0
STKBEG	S	0CA	2
STKFLG	B	0E8	0
STKFLGX	B	0E9	0
STKINP	S	0C7	2
SVM	N	0FC	0
SYSPRIV1	B	0F4	0
SYSPRIV2	B	0EF	0
SYSR0	S	0F6	0
SYSR1	S	0F9	0
SYSR2	S	040	2
T0	T	007	0
T1	T	006	0
T2	T	005	0
T3	T	004	0
T4	T	014	0
T5	T	015	0
T6	T	016	0
T7	T	017	0
TAPSTW	C	01C	0
TBIT	B	093	0
TPRDY	B	0E7	0
TS	R	00D	D
TSBEG	S	06E	0
TSDSP	T	0B5	0
TSEND	S	071	0
TSWA	T	0B4	0
TTLY	T	003	0
TYMO	T	022	2
UBIT	B	094	0
UPD	R	007	7
UPDBEG	S	074	0
UPDDSP	T	09D	0
UPDEND	S	077	0
UPDFID	D	09E	0
UPDWA	T	09C	0
USER	T	047	0
VBIT	B	095	0
VM	N	0FD	0
VOBIT	B	0DF	0
VT	N	00B	0
WBIT	B	096	0
WMBIT	B	09D	0
WMODE	T	043	0
XBIT	B	097	0
XMODE	T	046	0
XNFID	D	001	F
XNLOCK	H	000	F
XNNCF	H	001	F
XNPCF	H	00A	F
XNRES	H	00B	F
XPFID	D	003	F
YBIT	B	098	0
ZBIT	B	099	0
ZEROBIT	B	00D	0

TCL-I & TCL-II

PROCESSORS

&

PROC INTERFACE

VERB FORMAT

General

A verb is an entry in the master dictionary whose D/CODE begins with the character "P". The special sequence "PQ" is reserved, and defines a PROC; otherwise the format of the verb is as defined below. The verb contains three mode-id's - hexadecimal character fields that specify the transfer of control from one processor to another. The first mode-id is mandatory and specifies the location to which TCL-I transfers control after editing the input statement; the second mode-id is also mandatory for TCL-II and ENGLISH verbs, and specifies a secondary processor exit. The third mode-id is usually optional. In addition, an option string may be present for TCL-II verbs.

Verb Format:

Attribute Number	Description	Examples
1	"Px"; x is a single character (not "Q") stored in the character register SCP. X may be null.	LIST verb: PA COUNT verb: PB
2	Primary mode-id, to which TCL-I transfers control.	ALL ENGLISH verbs: 35 ALL TCL-II verbs: 2
3	Secondary mode-id; stored in the tally MODEID2	EDIT verb: D ASSEMBLE verb: 17
4	Tertiary mode-id; stored in the tally MODEID3	
5	Option string, for TCL-II verbs (see TCL-II documentation)	

TCL-I

Functional Description

TCL-I is the basic entry point (not a subroutine) for the terminal control language process. It is entered solely from the WRAPUP processor after WRAPUP has completed processing of the previous TCL statement.

The primary functions of the TCL-I processor are as follows:

1) Determine if a PROC is in control, and if it is, exit to the PROC processor for continuation of the PROC.

2) If not, obtain a line of input from the terminal.

3) Attempt to retrieve the verb (first set of contiguous non-blank data in the input buffer) from the master dictionary and validate it as such.

4) Set up the parameters from the verb; edit and copy the remainder of the data in the input buffer to the work-space IS.

5) Exit to the processor specified in the primary mode-id parameter of the verb.

Editing Features

1) All control characters, and system delimiters (SB, SM, AM, VM, SVM) in the input buffer are ignored.

2) Redundant blanks (sequence of two or more blanks) are not copied, except in strings enclosed by single or double quote signs.

3) Strings enclosed in single quote signs are copied as: (SM) I string (SB)

4) Strings enclosed in double quote signs are copied as: (SM) V string (SB)

5) End of data is marked with a: (SM) Z

Output Interface

ISBEG	S	Defines start of work space where edited input data has been copied.

IS	R	=ISBEG
IR	R	Points to AM following attribute 4 of the verb, or to end-of-data AM of verb.
SR4	S	Points to AM at end-of-data of verb.
IBBEG	S	Points to start of last input line from terminal.
IB	R	Points to SM terminating input line.
IBEND	S	As above.
SCP	C	Contains character following "P" in verb attribute one; blank if none specified.
SC0	C	Contains a blank
SC1	C	Contains a blank
SC2	C	Contains a SB
MODEID2	T	Contains secondary mode-id from verb; zero if none specified.
MODEID3	T	Contains tertiary mode-id from verb; zero if none specified.
IBIT	B	Set if any string enclosed by single-quote signs has been found.
VBIT	B	Set if any string enclosed by double-quote signs has been found.
PQFLG	B	Set if a PROC is in control.
BASE MODULO SEPAR		Contains base-FID, modulo and separation of of master dictionary.

All other bits are zero (A through Z bit and SB0 through SB32. All other process work-space pointers are set to their initial condition. (See WSINT documentation).

Subroutines Used

RETIX: CVTHIR

Error Conditions

The following cause an exit to the WRAPUP processor with the message indicated:

271: One PROC cannot call another

3: Verb cannot be identified in the M/DICT.

30: Verb format error. (Premature end-of-data, non-hex character in mode-id's)

2: Uneven number of single or double quote signs in input data.

TCL-II

Functional Description

TCL-II is entered from TCL-I by those verbs requiring access to a file, and to all, or explicitly specified items, from the file. TCL-II exits to the processor whose mode-id is specified in MODEID2; typically processors such as the EDITOR, ASSEMBLER, LOADER, etc. use TCL-II to feed them the set of items which was specified in the input data

On entry, TCL-II checks the verb definition for a set of option characters (in attribute 5 of the verb); verb options are single characters as below; any combination may be specified.

Option Character	Meaning
C	Copy - the item retrieved is copied to the workspace IS.
F	File access only - the item-list is ignored; only the file parameters are set up by TCL-II. If this option is present, any others are ignored.
N	New item acceptable - if the item specified is not on file, the secondary processor still gets control (example: the EDITOR can process a new item).
P	Prints item-id on a full-file retrieval, as each item is retrieved.
U	Updating sequence flagged - if items are to be updated as retrieved, this option is mandatory.
Z	Final entry - the secondary processor will be entered once more after all items have been retrieved (example: COPY processor, to print a message).

The input data string to TCL-II consists of the file-name (optionally preceded by the modifier DICT, which specifies access to the dictionary of the file), followed by a list of items, or an asterisk (*) specifying retrieval of all items on the file. The item-list may be followed by an option list (options to the secondary processor) which must be enclosed in parentheses. These options must consist of a sequence of single characters, or a decimal number, or two decimal numbers separated by a minus sign (specifying a range of numbers). Multiple options are separated by commas. The option characters A through Z set the corresponding bits ABIT through ZBIT (A sets ABIT, etc.); the numbers are stored in tallies D4 and D5, and the bit NOBIT set if the numerical option is found.

Output Interface

DAF1	B	Set if update option was found.
DAF2	B	Set if "C" otpion was found.
DAF3	B	Set if "P" option was found.
DAF4	B	Set if "N" option was found.
DAF5	B	Set if "Z" option was found.
DAF6	B	Set if "F" option was found.
DAF8	B	Set if accessing a dictionary-file (DICT in input).
RMBIT	B	Set if item found and retrieved.
BASE	D	First exit only: base FID, modulo and separation of file being accessed.
MODULO	T	
SEPAR	T	
SBASE	D	Base FID, modulo and separation of file being accessed.
SMOD	T	
SSEP	T	
IS	R	Points to non-blank following file-name if "F" option specified.
DBASE	D	Contains base-FID, modulo and separation of dictionary of file being accessed if "F" option is specified.
DMOD	T	
DSEP	T	

Following specifications meaningful only if "F" option is not present:

BMSBEG	S	Points one prior to area containing the item-id, which is terminated by a AM.
ABIT through ZBIT	B	Set according to option list (see description above).
NOBIT	B	Set if numerical option found.
D4	D	Contains numerical option value(s).
D5	D	
SR0	S	Points one prior to count field of item on file.
SIZE	T	Contains count field of item.
SR4	S	Points to last AM of item on file.

MODEID3	T	Contains tertiary mode-id from verb.	
		"C" Option in Verb	No "C" Option in Verb
ISBEG	S	Points one prior to beginning of copied item (including item-id, not including count field)	
IS	R	Points to last AM of copied item.	Points to end of string.
ISEND	S	=IS	
IR	R	Points to last AM of item on file.	Points to AM following item-id on file.

Internal Usage

RMODE	T	Contains mode-id of entry-point within TCL-II that WRAPUP processor exits to; must be maintained by lower-level processors.

All elements as used by GETITM (q.v.)

Subroutines Used

RETIX, GBMS, GDLID, GETITM, GETOPT, GETFILE.

Error Conditions

The following conditions cause an exit to the WRAPUP processor with the error number indicated:

200	File-name not specified
201	File name illegal or incorrectly defined in the M/DICT.
202	Item not on file (will not abort processing).
203	Item list missing.
204	Error in format of option list.
13	DL/ID item not found in dictionary-file.

User Exits From PROC

A user- program can gain control during execution of a PROC, by using the Uxxxx command in the Proc, (where xxxx is the hex. mode-id of the user-program). The user-program can perform special processing, and then return control to the PROC processor. Necessarily, certain elements used by PROC must be maintained by the user-program. These elements are marked with an asterisk in the table below:

Input Interface

*BASE *MODULO *SEPAR	D T T	Contains FID, modulo and separation of M/DICT.
*PQBEG	S	Points one prior to the first PROC statement.
*PQEND	S	Points to terminal AM of the PROC.
PQCUR	S	Points to AM following the Uxxxx element.
IR	R	=PQCUR
*PBUFBEG	S	Points to buffer containing primary and secondary (if any) input buffers. Format: (SB) ... primary input ... (SM) (SB) ... secondary input ... (SM)
*ISBEG	S	Points to buffer containing primary output line.
*STKBEG	S	Points to buffer containing "stacked input" (secondary output)
*SBIT	B	Set if ST ON command is in effect.
IB	R	Current input buffer pointer (may be within primary or secondary input buffers).
*SC2	C	Contains a blank.

		SBIT on	SBIT off
IS	R	Last byte moved into secondary output buffer.	Last byte moved into primary output buffer.
UPD	R	Last byte moved into primary output buffer.	Last byte moved into secondary output buffer.

Output Interface

IR	R	Points to AM preceding next PROC statement to be executed; may be altered to change location of continued PROC execution.
IS UPD IB	R R R	May be altered as needed to alter data within input and output buffers; but formats described above must be maintained.

Exit Convention

The normal method of returning control to the PROC processor is to execute an external branch to 2, PROC-I. If it is necessary to abort PROC control and exit to WRAPUP, set PQFLG off, and execute an external branch to one of the WRAPUP entry points.

Note that, when the PROC eventually transfers control to TCL, (via the P operator), certain elements are expected to be in an initial condition. Therefore, if the user-program uses these elements, they should be reset before returning to PROC, unless the elements are deliberately set up as a means of passing parameters to other processors. Specifically, the bits ABIT through ZBIT are expected to be zero by the TCL-II and ENGLISH Processors. It is best to avoid usage of these bits in PROC user-exits. The scan character registers SC0, SC1 and SC2 must also contain an SB. a blank and a blank respectively.

WRAPUP PROCESSOR

General

The WRAPUP Processor is entered under the following conditions:

1) Termination of a TCL statement, when it is required to "wrap-up" processing and to return to the TCL level.

2) Intermediate stage in processing a statement, when it is required to print messages from the ERRMSG file, or to perform disc updates, and then return to the calling processor.

3) User-initiated termination of processing, via the 'END' command in DEBUG.

The WRAPUP Processor has several entry points, depending on the type of action required; several of these entry points are provided to simplify the interface requirements when an error message is required (Note, for instance, that MD995 may be entered immediately after return from a call to, say, RETIX, if the item is not found on file, by setting up C1 to the error message number).

WRAPUP also performs the following functions before returning to TCL:

1) Closes all open spool files, if LPBIT is set.

2) Releases linked overflow space if OVRFLCTR ≠ 0.

WRAPUP-I

Functional Description

1. Prints, or sets up for printing, messages that are stored in the file "ERRMSG" (must be catalogued in the process M/DICT.

2. Performs disk updates as specified in the history string.

3. Terminates processing of a TCL statement; re-initializes elements.

Interface Requirements

History String. The history string is from HSBEG through HSEND. If HSBEG=HSEND, the string is null; this is the initial condition on

entry to TCL. If HSBEG ≠ HSEND, elements in the string are processed by WRAPUP. There are three types of elements; all other element types are ignored.

1. Output message.

   (SM) 0 (AM) Message-Id (AM) (Parameter (AM) ...) (SM)

   where Message-Id is the item-id of an item in the ERRMSG file. (Normally a decimal numeric).

   Parameters are character string that is to be passed to the message formatter (PRTERR, WRAPUP-III).

2. Disk Update/Delete string.

   (SM) DU (AM) base (VM) modulo (VM) separ (AM) item-id (AM) .. (item-body) (AM) (SM)

   (SM) DD (AM) base (VM) modulo (VM) separ (AM) item-id (AM) (SM)

   DU - Disk update; replaces entire item in the file specified by the decimal parameters base, modulo and separ.

   DD - Disk delete; deletes item from the file.

3. End-of-string element

   (SM)Z

Conventionally, a process wishing to add data to the history string begins at HSEND+1; when the entire additional element(s) has been added, the string is terminated with a (ZM) Z, and HSEND reset to the (SM).

If WMODE is non-zero on any entry to WRAPUP, an indirect subroutine call (3SL*) via WMODE will be executed. This allows special processing to be done on every WRAPUP entry.

WRAPUP may be called as if it is a subroutine by setting RMODE to the Mode-ID of the program to which WRAPUP returns control to; note however, that the return-stack is always set to a null or empty condition by WRAPUP. On the error-message setup entry points (MD99/MD993/MD994/MD995), if VOBIT is set and RMODE non-zero, the appropriate messages are stored in the history string, for printing on a final entry with RMODE zero.

If OVRFLCTR is non-zero, it is assumed that it contains the starting FID of a linked set of overflow frames that is to be released to the system overflow pool. This tally is used, for instance, by the SORT processor to store the beginning FID of the sorted table; the overflow space used by the Sort is thus always released to the system even if the sort is aborted by the debug 'END' command.

Entry Points

MD993	A message number is stored in C1; a numeric parameter is stored in C2. Sets up the message in the history string and exits to MD99.
MD994	A message number is stored in C1; a character string parameter is stored for IS+1 through an AM or SM. Sets up the message and exits to MD99.
MD995	As above, except that the parameter is from BMSBEG+1 through an AM or SM.
MD99	Message numbers (without any parameters) may be stored in REJCTR, REJ0 and REJ1 (no action is taken if zero.) After setting up the messages in the history string, exits to MD999 (If VOBIT set, skips history string processing in MD999).
MD999	Processes all elements in the history string. Reinitializes process work spaces; exits to TCL if RMODE = 0; to calling program via RMODE if non-zero.
TCL	Kills history string; PROC control; exits to TCL (Entry point of "END" command for DEBUG).

Subroutines Called

PRTEER	(WRAPUP-III) to print error messages.
UPDITM	(WRAPUP-II) to perform disk updates.
ISINIT	To re-initialize ISBEG/ISEND/OSBEG/OSEND.
WSINIT	To re-initialize process work areas. (BMS/CS/AF/IB/OB/TS)
CVTNIS	To convert a decimal character string, from the IS, to binary.

External Branches

LOGOFF	If USER = 0

Error Messages

"DISK UPDATE STRING ERROR"; self explanatory.

UPDITM (WRAPUP-II)

Functional Description

This subroutine performs updates to the disc files in the system. It is described in the next section (DISC FILE I/O).

PRTEER (WRAPUP-III)

Functional Description

This subroutine is used primarily by the WRAPUP processor to retrieve and print error messages from the system file ERRMSG. A parameter string may be passed to the subroutine, which will format and insert the parameters as specified by the codes in the message item. System message item-ids are numeric; however, any item-id can be specified. See description of error messages for format of the codes.

Input Interface

TS	R	Points one prior to the message item-id, which must be terminated by an AM. Parameters optionally follow, being delimited by AM's. The parameter string must terminate with a SM.

Output Interface

TS	R	Points to AM following message-id, or AM or SM following last parameter output.

Internal Usage

EBASE EMOD ESEP	D T T	Set up by PRTERR to ERRMSG file, if EBASE = 0 on entry; otherwise these elements are assumed to be already initialized.

All elements are used by WRTLIN and RETIX.

Error Message Formats

A (dec-number)	Parameter insertion code; the next parameter from the history string, if any, is placed in the output buffer. If "dec-number" is specified, the parameter is left-justified in a blank field of the specified length.

R (dec-number)	As above; the parameter is right-justified in the blank field of the specified length.
E (char-string)	The message ID, surrounded by brackets, is placed in the OB, followed by the optional character string.
H (char-string)	The character string is placed in the OB.
L (dec-number)	The output buffer is printed, and the specified number of line feeds are output (one if "dec-number" not specified).
X (dec-number)	The OB is incremented by the number of spaces specified. If the end of the line is reached, the output buffer is printed and a new line is started.
T	Adds system time in the format HH:MM:SS to the output buffer.
D	Adds system date in the format DD MMM YYYY to the output buffer.

On exit from this subroutine, the output buffer is printed and a new line started, unless the last character in the OB is an "+", which causes printing of the buffer only.

Functional Element Usage By All WRAPUP Modes

Bits	Description of Use	Mode
VOBIT	Store/Print Flag; Store Messages if set	ALL
SB60	Scratch	Wrapup-III
SB61	Scratch	Wrapup-III
RMBIT	As used by RETIX	II, III

Tallys

REJCTR REJ0 REJ1	Input error messages	I(MD99)
C1	Input error messages	I (MD993/994/995)
C2	Input error messages	I (MD993)

Bits	Description of Use	Mode
C3	Scratch	II
D2	Scratch	II
D3	Scratch	II
SIZE	Scratch	II
NEXT	Scratch	II
RECORD, LINK(S)	As used by disc-I/O subroutines	II
OVRFLW	Scratch	I, II
D4	Scratch	II
T4	Scratch	II
BASE	Various	II
MODULO		I, II, III
SEPAR		
EBASE	Base-FID, modulo and separation	
EMOD	of the ERRMSG file.	III
ESEP		
RMODE	Return mode-ID	I
WMODE	Special processing exit mode-ID	I

Registers

HSBEG	Beginning and end of history	I
HSEND	string	
BMSBEG	Reinitialized on exit from MD999	
BMSEND		
CSBEG		
CSEND		
ISBEG		
ISEND		
OSBEG		
OSEND		
IBBEG		
IBEND		

OBBEG OBEND AFBEG AFEND		
UPD	Scratch	I, II
IR	Scratch	
TS	Scratch	I, II, III
AF	Scratch	III
Return Stack	Reset to null condition on exit from MD999	

DISC FILE I/O

RETIX AND RETI

Functional Description

Retrieves an item stored in a file. The item-id is explicitly specified to this routine, as are the file parameters base, modulo and separation. The subroutine performs a "hashing" algorithm (see HASH documentation) to determine the group within which the item may be present, and then searches sequentially down the data in the group for a matching item-id. If found, the subroutine returns pointers to the beginning and end of the item, and the item size (from the item count-field).

The item-id is specified in a buffer defined by the register BMSBEG; if the entry RETIX is used, the item-id must be terminated by an AM; if RETI is used, the register BMS must point to the last byte of the item-id. An AM will be appended to the item-id by RETI.

Input Interface

BMSBEG	S	Points one prior to the item-id required.
BMS	R	RETIX: not an input interface requirement. RETI: points to the last byte of the item-id.
BASE	D	Contains the base FID of the file.
MODULO	T	Contains the modulo (number of groups) of the file.
SEPAR	T	Contains the separation (number of frames per group) of the file.

Internal Usage

XMODE	T	Used to exit to subroutine IROVF if a group format error occurs.

Output Interface

BMS BMSEND	R S	Points to last character of item-id.
RECORD		Contains the beginning FID of the group to which the item-id hashes.

NNCF FRMN FRMP NPCF	T D D T	Contain the link fields of the frame above.
XMODE	T	Zero

		Item Found	Item Not Found
RMBIT	B	Set.	Zero.
SIZE	T	Item count-field.	Zero.
R14	R	One prior to item count field.	Points to last AM of last item in group.
IR	R	Points to first AM of item	Points to AM indicating end of group data (=R14 +1).
SR4	S	Points to last AM of item	=R14

Subroutine Usage

RDREC, HASH, IROVF — Requires one additional level of subroutine linkage; three if a group format error occurs.

Error Conditions

If the data in the group is bad - premature end of linked frames, or non-hexadecimal character encountered in the count field-the message:

```
****GROUP FORMAT ERROR AT:      .xxxxxx
```

is returned (xxxxxx is six character hexadecimal FID indicating where the error was found), and the routine returns with an "item not found" condition. Data is not destroyed, and the group format error will remain.

GETITM

Functional Description

Sequentially retrieves all items in a file. This routine is called repetitively to obtain items from a file one at a time until all items have been retrieved. The order in which the items are returned is the same as the pseudo-random storage sequence.

If the items retrieved are to be updated by the calling routine (using The routine UPDITM), this should be flagged to GETITM, which will then perform a two-stage retrieval process by first storing all item-id's (per group) in a table, then will use this table to actually retrieve the items on each call. This is necessary because, if the calling routine updates an item, the data within this group shifts around; GETITM cannot simply maintain a pointer to next item in the group, as it does if the "update" option is not flagged.

An initial entry condition has also to be flagged to GETITM; it then sets up and maintains certain pointers which should not be altered by the calling routines until all the items in the file have been retrieved.

Note the functional equivalence of the output interface elements with those of RETIX.

Input Interface

DAF7	B	Initial entry flag; must be zeroed on first call to GETITM.
DAF1	B	If set, "update" option is in effect.
DBASE	D	Contains base FID, modulo and separation of the file
DMOD	T	
DSEP	T	
BMSBEG	S	Points one prior to an area where the item-id of items retrieved may be copied. Must be at least 50 bytes in length.

Internal Usage

RECORD	D	
NNCF, FRMN, FRMD, NPCF, XMODE, EOFBIT		Used internally.

These elements should not be altered by any other routine while GETITM is used.

DAF7	B	See above
DAF1	B	
DBASE	D	Current group beginning FID.
DMOD	T	Number of groups left to be processed.
DSEP	T	Separation parameter of file (unchanged).
SBASE	D	Saved values of original base FID, modulo and separation.
SMOD	T	
SSEP	T	
NXTITM	S	If DAF1 set: Points one before next item-id in prestored table. If DAF1 zero: Points to last AM of item previously returned.
OVRFLCTR	D	Overflow space table starting FID; used if DAF1 set.

Output Interface

RMBIT	B	Set if item found; zeroed if all items exhausted.
SIZE	T	Contains item count-field.
R14	R	Points one prior to count-field.
SR0	S	=R14
IR	R	Points to first AM of item.
SR4	S	Points to last AM of item.
XMODE	T	Zero

Subroutines Used

RDREC, GNSEQI, GNTBLI (last two local); requires one additional level of subroutine linkage.

Error Conditions

As for RETIX (q.v.); except that the routine will continue retrieving items, if any more exist, after the error condition is reported.

UPDITM

Functional Description

This routine performs updates to a disc file defined by its base FID, modulo and separation. If the item is to be deleted, the routine will compress the remainder of the data in the group in which the item resided; if the item is to be added, it will be added at the end of the current data in the group; if the item is to be replaced, functionally a deletion and then an addition takes place. UPDITM does not perform a merge into an already existent item.

If the change of data in the group reaches an end of the linked frames, UPDITM will obtain another frame from the overflow space pool and link it to the previous linked set; as many frames as required will be added. If the deletion or replacement of an item causes an empty frame at the end of the linked frame set, and that frame is not in the "primary" area of the group (see Data Structure), it will be released to the overflow space pool.

Once this routine is entered, the processing cannot be interrupted until completed.

Input Interface

BMSBEG	S	Points one prior to the item-id to be updated; the item-id must be terminated by an AM.
TS	R	Points one prior to the item body; the data must be terminated by a SM. This is not an input interface element for item-deletes.
CH8	C	Contains the character 'U' for an item-addition or replacement; 'D' for an item-delete.
BASE MODULO SEPAR	D T T	Contains the base FID, modulo and separation of the file being updated.

Internal Usage

RMBIT	B	Various
INHIBITSV1	B	Saves condition of INHIBIT
CTR0	T	
CTR1	T	

D2	D
D3	D
XMODE	T

Output Interface

TS	R	Points to SM terminating item-body for item-add or replace.
IR	R	Points to AM terminating group data
UPD	R	Points to last byte of last frame in group
CS	R	Points one prior to item-id on item-add or replace
BMS	R	Points to AM following item-id
SIZE	T	Contains new item size on add or replace

Subroutine Usage

RETIX, RDREC, RDLINK, WTLINK, IROVF, BMSOVF, ATTOVF, RELOVF. Uses two additional levels of subroutine linkage.

Error Conditions

1. If the group data is bad, premature end of linked data set, or non-hexadecimal character found in a count field, the group data is terminated at the last good item, and the message:

   *** GROUP FORMAT ERROR AT: . xxxxxx

   is returned (xxxxxx is a six digit hexadecimal FFD indicating the location of the error), before the process continues.

2. If the file being updated is the M/DICT (as indicated by BASE being equal to MBASE), and the system privilege level one flag is not set, the routine aborts with the message:

   YOUR SYSTEM PRIVILEGE LEVEL IS NOT SUFFICIENT FOR THIS STATEMENT

   The update is not performed.

3. If the item-id is greater than 50 characters, the update is not performed; no indication is returned.

4. If the item exceeds the maximum size (32767 bytes, X'7FFF'), the item is truncated to 32767 bytes; no indication is returned.

GBMS

Functional Description

Sets up the base FID, modulo and separation parameters of a file from the file definition item that has been retrieved. Typically this routine will be called after a call to RETIX which retrieves the filename from the master dictionary.

The routine handles both 'D' and 'Q' code items; a 'D' code item is a direct file-pointer, and has the base FID, modulo and separation of the file in attributes 2, 3 and 4. A 'Q' code item is a synonym pointer to a file defined in any account in the SYSTEM dictionary. This subroutine also performs the file access-protection checks. It is assumed that register LOCKSR points to the user's lock codes (in his logon entry in the SYSTEM dictionary; if the file has a lock code, a matching lock code is required for GBMS to return successfully. A non-match causes an exit to WRAPUP with message 210.

Input Interface

DAF1	B	If zero, uses retrieval lock-codes in LOGON entry for lock-code comparison; if set uses update lock-codes.
IR	R	Points to, or one prior, 'D' or 'Q' code in attribute one of file-definition item.
SR4	S	Points to AM at end of file-definition item.
LOCKSR	S	Points one prior to the user's lock-code field in his SYSTEM dictionary entry.

Output Interface

RMBIT	B	Set if base, modulo, separation successfully converted; zero if error in format, 'Q' item not found, etc.
BASE MODULO SEPAR	D T T	Contain base FID, modulo and separation of the file (if RMBIT set)
IR	R	Points to AM following attribute four of the file-definition item.

Subroutine Usage

CVDR15; recursive call to GBMS and RETIX if 'Q' code item; two further levels of subroutine linkage if 'D' Code; three if 'Q' code item.

Errors

On failing the lock code comparison test, an exit is taken to WRAPUP with message 210 (FILE IS ACCESS PROTECTED); to ensure termination of the curent process, RMODE is zeroed, PQFLG is set off and the history string set null before the exit.

GDLID

Functional Description

This subroutine gets the base, modulo and separation parameters from the DL/ID item in a dictionary. Typically this routine is called immediately after the dictionary base, modulo and separation have been obtained by GBMS.

GDLID retrieves the DL/ID item from the dictionary, and then enters GBMS to pick up its base modulo and separation.

Input Interface

BASE	D	Contains b, m, s of a file, containing the DL/ID item.
MODULO	T	
SEPAR	T	

Output Interface

RMBIT	B	Set if DL/ID found; zero otherwise.
BASE	D	Contains base FID, modulo and separation of the data-file (if RMBIT is set).
MODULO	T	
SEPAR	T	

Other elements as from GBMS and RETIX except that BMS/BMSBEG/BMSEND do not contain the DL/ID item-id.

TERMINAL I/O

GETIB AND GETIBX

Functional Description

GETIB and GETIBX are the standard terminal input routines. Register IBBEG points to a buffer area where the routine will input the data. Input continues to this area until either a carriage return or line feed is encountered, or until a number of characters equal to the count stored in IBSIZE have been input. The carriage return or line feed terminating the input line is overwritten with a segment mark (SM) and register IBEND points to this character on return. If the input is terminated because the maximum number of characters have been input, a SM will be added at the end of the line. This subroutine calls the subroutine GETBUF to read input data from the terminal. On return, GETIB then determines if the last character was a carriage return or a line feed which terminates the line, and causes a CR/LF echo to the terminal; if not, it either accepts or deletes the control character, depending upon the setting of the bit CCDEL, and calls GETBUF again.

The entry GETIB also provides the facility for taking the input from a stack instead of directly from the terminal. The bit STKFLG, if set, indicates that stacked input is present and register STKINP points to the area where the stacked input is stored. Input is copied from the stack area to the buffer, through the delimiter AM. The delimiter SM is used to signal the end of stack input. When this is encountered, STKFLG is turned off to indicate no more stacked input, and the routine then goes to the terminal for further input. The entry GETIBX does not test for stacked input. The stacked input feature is used primarily by the processor PROC to store input lines, which are returned to requesting processors as if they originated the terminal. If the last character of a stacked line is a "<", it will be replaced with a SM.

Input Interface

IBBEG	S	Points one prior to buffer area where input is to be stored. Size of the buffer must be two characters greater than the value in IBSIZE.
IBSIZE	T	Contains maximum number of input characters to be accepted.
STKFLG	B	If set, indicates that "stacked" input may exist; if it does, terminal input will not be requested until the stack is exhausted.
STKINP	S	Points to next "stacked" input line; lines are delimited by AM's; a SM indicates end of stack.
PRMPC	C	Terminal "prompt" character; output before data is requested from the terminal.

LFDLY	T	Low-order byte contains the number of "fill" characters (nulls) to be issued after a carriage-return/line feed is output to the terminal.
CCDEL	B	If set, control characters are deleted from terminal input.

Output Interface

IB	R	Set to IBBEG.
IBEND	S	Points to SM one past last character input. (Overwrites CR or LF character).
STKFLG	B	Zeroed if end of stacked input reached.
STKINP	S	Points to next line of stacked input.

Subroutines Usage

GETBUF, PCRLF (both local); uses one additional level of subroutine linkage.

Error Conditions and Abnormal Exits

If the "stacked" input line exceeds IBSIZE, the line is truncated at IBSIZE; the remainder of the line is lost.

GETBUF

Functional Description

Inputs data from the terminal, and performs line editing functions. Returns control when a non-editing control character is input, or when the number of characters specified in T0 has been input.

Line editing features:

Character Input	Action
Control-H	Logically backspaces buffer pointer; echoes character defined at BSPCH.
Control-X	Logically deletes (or cancels) entire input buffer; echoes a carriage return, line feed, re-issues prompt character.
Control-R	Re-types input line.
Rubout	Ignored; the character is echoed, but is not stored in the buffer.
Control-shift-K Control-shift-L Control-shift-M Control-shift-N Control-shift-O	These characters are converted to the internal delimiters SB, SVM, VM, AM, and SM respectively; they echo as the characters [,/,], ,_

Note: The high-order bit of all characters input is zeroed.

Input Interface

R14	R	Points one prior to input buffer area.
T0	T	Maximum number of characters to be accepted.
PRMPC	C	Character output as a "prompt" when input is first requested.
BSPCH	C	Character echoed when a control-H is input.

Output Interface

R15	R	Points to control character causing return to caller.

WRTLIN AND WRITOB

Functional Description

Standard terminal output routines. Outputs data to the terminal or line-printer; controls pagination and page-heading routines. Entry WRTLIN adds a carriage-return/line-feed to the data; WRITOB does not. The data to be output starts at OBBEG, and continues through to the location addressed by OB.

The data is output to the terminal if LPBIT is off; it is stored in the printer spooling area if it is set. Pagination and page-heading are controlled by PAGINATE, if set. In this case, when the number of lines output in the current page (in LINCTR) exceeds the page size (in PAGSIZE), the following actions take place: 1) The number of lines specified in PAGSKIP are skipped; 2) the page number in PAGNUM is incremented, and 3) A new page heading is printed (see PRNTHDR documentation). A zero value in PAGSIZE suppresses pagination regardless of the setting of PAGINATE.

Editing Features

The internal delimiters SM, AM, VM, SVM, SB, are converted to the characters _, ,],/,[, respectively, if SMCONV is off; the output buffer area is blanked if NOBLNK is off. Trailing blanks are always deleted by the entry WRTLIN.

Carriage returns and line-feeds should not occur in the buffer if pagination is to be used.

Input Interface

OBBEG	S	Points one prior to the output data buffer.
OB	R	Points to the last character in the buffer; the buffer must extend two characters beyond this location.
LPBIT	B	If set, routes output to the spooler.
LISTFLAG	B	If set, suppresses all output to the terminal.
SMCONV	B	If set, suppresses conversion of internal delimiters.
NOBLNK	B	If set, suppresses blanking of output buffer.
PAGINATE	B	If set, pagination and page-headings are invoked.
PAGHEAD	S	Location of page-heading message.

LINCTR	T	Number of lines printed in current page.
PAGSIZE	T	Number of printable lines per page.
PAGSKIP	T	Number of lines to be skipped at bottom of page.
		(Above four elements used only if PAGINATE IS set.)
LINESOUT	T	Incremented on every entry.
LFDLY	T	Lower byte contains number of "fill" characters to be output after CR/LF.

Internal Usage

RECORD, LINQUE, OVRFLW, IR	Destroyed if LPBIT is set causing output to go to the print spooler.

Output Interface

OB	R	Reset to OBBEG

LINCTR, PAGNUM, LINESOUT: reset appropriately.

Subroutines Used

PPUT (printer spooler); PCRLF: NPAGE. Requires two additional levels of subroutine linkage.

Errors and Abnormal Exits

None

PCRLF

This subroutine may be used to output a CR/LF, with appropriate delays if necessary. Its use is not compatible with pagination.

Input Interface

LFDLY	See above.

PRNTHDR AND NEWPAGE

Page-heading control routine; PRNTHDR is used to initialize the pagination control elements; NEWPAGE is used to cause a skip to a new page, and to output a new page heading.

PRNTHDR sets the page number to one, the line counter to zero; sets the pagination flag, and outputs the first page heading. The page heading must be stored in a buffer defined by PAGHEAD; the header message is a string of data terminated by an SM; other system delimiters are used as below:

Delimiter	Action
SM,X'FF'	Terminates header line with a CR/LF.
AM,X'FE'	The current page-number is inserted in the heading.
VM,X'FD'	Prints header line, starts a new header line.
SVM,X'FC'	The current time and date are inserted in the heading.

Carriage-returns, line feeds and form-feeds should not be included in the header message.

Input Interface

LPBIT	B	If set, output is routed to the print spooler.
PAGHEAD	S	Points one prior to the beginning of the header message, which must be terminated by an SM.
LINCTR	T	Line number in current page.
PAGNUM	T	Current page number.
LFDLY	T	Lower byte contains number of "fill" characters to be output to terminal after a CR/LF. Upper byte: if non-zero, a form-feed (X'0C') character will be output before the start of a new page, and that number of "fill" characters will be output.
OBBEG	S	Points one prior to a buffer when the translated header message is built; this buffer area must be 16 bytes greater than the longest header line. (Not total header message size.)

Internal Usage

OB	R	Used to build the header message and to output it.

Output Interface

LINCTR, PAGNUM	Reset appropriately.

Subroutine Usage

WRITOB, TIMDATE	Uses two additional levels of subroutine linkage (three if time and date are inserted in header)

Errors

None

PRINT AND CRLFPRINT

Functional Description

Sends a message to the terminal from textual data in the calling program; used primarily for printing error messages. These subroutines are not compatible with output conventions to the lineprinter, and with the pagination routines. The message is a string of characters assembled immediately following the subroutine call in the calling program. The message must be terminated by one of the four delimiters SM, AM, VM, or SVM. Control is returned to the instruction at the location immediately following the terminal delimiter.

Delimiter	Action
SM,X'FF' AM,X'FE'	End of message; print carriage-return/line-feed before return.
VM,X'FD'	Print carriage-return/line-feed, continue.
SVM,X'FC'	End of message, no carriage return/line-feed.

Input Interface

Message follows the call to this routine.

LFDLY	T	Low-order byte contains number of "fill" characters after CR/LF is output.

Subroutine Usage

PCRLF; one additional level of subroutine linkage.

Errors

None

VIRTUAL MEMORY I/O

RDREC

Functional Description

RDREC is used to set up the registers IR, IRBEG, and IREND to the beginning and ending of the frame as defined by the tally RECORD. The subroutine assumes the frame has the linked format and therefore, IR and IRBEG are set pointing to the eleventh byte of the frame, that is, one prior to the first data byte of the frame. IREND is set up pointing to the last or 511th byte of the frame. Additionally the subroutine RDLINK is entered to set up R15 pointing to the link portion of the frame, and to set up the link elements NNCF, NPCF, FRMN, FRMP.

Input Interface

RECORD	D	Contains FID required.

Output Interface

IR	R	Points one prior to first data byte of frame.
IRBEG	S	As above.
IREND	S	Points to last data byte of frame.
R15	R	Points to zero-th byte of frame.
NNCF	H	Contains "nncf" field of frame.
FRMN	D	Contains forward link FID of frame.
FRMP	D	Contains backward link FID of frame.
NPCF	H	Contains "npcf" field of frame.

Subroutines Usage and Error Condition

None

RDLINK AND WTLINK

Functional Description

These routines read or write the link fields from or to a frame, to or from the tallies NNCF, FRMN, FRMP and NPCF. The FID of the frame is specified in RECORD.

Input/Output Interface

RECORD	D	FID of frame whose links are to be written or read.
NNCF	H	Number of next contiguous frames.
FRMN	D	Next or forward link FID.
FRMP	D	Previous or backward link FID.
NPCF	H	Number of previous contiguous frames.
R15	R	Points to zero-th byte of frame.

Subroutines Used and Error Conditions

None

LINK

Functional Description

Sets up the link fields of a group of unlinked contiguous frames. Up to 127 frames can be so linked. In each frame of the linked set, this subroutine sets up the number of next contiguous frames field, the next or forward link field, the previous or backward link field, and the number of previous contiguous frames field.

Input Interface

RECORD	D	Contains first FID of the group to be linked.
NNCF	H	Contains one less than the number of frames in the group (NNCF <127).

Output Interface

R14	R	Points one prior to the first data byte of first frame of linked set.
R15	R	Points to the last data byte of the last frame of linked set.

Subroutines Called and Error Conditions

None

OVERFLOW SPACE MANAGEMENT

GETOVF, GETBLK

Functional Description

These routines obtain overflow frames from the overflow space pool maintained by the system. GETOVF is used to obtain a single frame; GETBLK is used to obtain a block of contiguous space (used mainly by the CREATEFILE processor). Note that the link fields of the frame(s) obtained by a call to GETBLK are not reset or initialized in any way; this is a function of the calling routine (also see ATTOVF and NEXTOVF documentation); GETOVF zeroes all the link fields of the frame it returns.

Input Interface

D0	D	(Accumulator) contains number of frames needed (block size), for GETBLK only.

Output Interface

OVRFLW	D	Contains FID of the frame obtained (GETOVF) or first FID of the block obtained (GETBLK).

Subroutines Used

SYSGET	One additional level of subroutine linkage required.

Error Conditions

Zero returned in OVRFLW if system overflow space is exhausted.

RELOVF, RELCHN AND RELBLK

Functional Description

These routines are used to release frame(s) to the overflow space pool. RELOVF is used to release a single frame; RELCHN is used to release a chain of linked frames (which may or may not be contiguous); RELBLK is used to release a block of continguous frames. A call to RELCHN specifies the first FID of a linked set of frames; the routine will release all frames in the chain until a zero forward link is encountered.

Input Interface

OVRFLW	D	Contains the FID of the frame to be released (RELOVF), or the first FID of the chain or block to be released.
D0	D	(Accumulator) contains the number of frames in the block to be released (block-size), for RELBLK only.

Output Interface

None

Subroutines Called

SYSREL (RELBLK only); one additional level of subroutine linkage required.

Errors

None

ATTOVF

Functional Description

ATTOVF is used to obtain a frame from the overflow space pool and to link it to the frame specified in RECORD. The forward link field of the frame specified in RECORD is set to point to the overflow frame obtained; the backward link field of the overflow frame is set to point to that in RECORD, and the other link fields of this frame are zeroed.

Input Interface

RECORD	D	Contains FID of the frame to which an overflow frame is to be linked.

Output Interface

OVRFLW	D	Contains FID of the frame obtained from overflow space.

Subroutines Used

GETOVF	Requires two additional levels of subroutine linkage.

Error Conditions

Zero returned in OVRFLW if system overflow space is exhausted.

NEXTIR AND NEXTOVF

Functional Description

These routines obtain the foward linked frame of the frame to which the register IR currently points; if the forward link is zero, an available frame from the system overflow space pool is obtained and linked appropriately (see ATTOVF documentation). In addition, the IR register triad is then set up before return, using the subroutine RDREC.

NEXTOVF may be used in a special way to automatically handle end-of-linked-frame conditions, on register six (IR), on single or multiple byte move on scan instructions. Set tally XMODE to the mode-id of the subroutine NEXTOVF before the move or scan instruction is executed; if the instruction causes register IR to reach an end-of-linked-frame condition (forward link zero), the system will generate a subroutine call to NEXTOVF, which in turn obtains and links up an available frame, and then resumes execution of the interrupted instruction. Note that the instruction "increment register by tally" cannot be so handled. Instructions compatible with NEXTOVF are: MIID, MII and MCI.

Input Interface

IR	R	On last data byte of frame.

Output Interface

IR IRBEG	R S	Points to first data byte of forward linked frame.
IREND	S	Points to last data byte of frame.
RECORD	D	Contains FID of frame to which IR points.
FRMN, NNCF, FRMP, NPCF, R15		As set up by RDREC.

Subroutine Used

RDREC; ATTOVF; uses two additional levels of subroutine linkage.

Error Conditions

None

WORK SPACE INITIALIZATION

WSINIT

Functional Description

Initializes the process work-space pointer dyads: BMSBEG, BMSEND; CSBEG, CSEND; AFBEG, AFEND; IBBEG, IBEND; OBBEG, OBEND; PBUFEND, PBUFBEG. All work-spaces except the last are contained on one frame; PBUFBEG and PBUFEND define a 4-frame linked work-space.

Work-Space	Size (Bytes)
BMSBEG-BMSEND	50
AFBEG-AFEND	50
CSBEG-CSEND	100
IBBEG-IBEND	Contents of IBSIZE; max. 140
OBBEG-OBEND	Contents of OBSIZE; max. 140
PBUFBEG-PBUFEND	2000 (4 linked frames)

Input Interface

VOBIT	B	Set if linking of PBUF-space required.
IBSIZE	T	Size of IB buffer.
OBSIZE	T	Size of OB buffer.

Output Interface

Storage registers set up as in above table; associated address registers BMS, AF, CS, IB, OB, set at beginnings of respective buffers.

Subroutine Usage

LINK; one additional level of subroutine linkage is used.

TSINIT

Functional Description

This routine sets up the pointers to a one-frame scratch space; the pointers set up are TSBEG, TSEND, by TSINT. The associated address register, TS, is set at the beginning of the buffer.

ISINIT

Functional Description

This routine initializes all the system work-space pointers. The link-fields of linked work-spaces (IS, OS, HS, PBUF) are not initialized unless VOBIT is set. As the IS, OS and HS may have additional work-space assigned to them, calling ISINIT with VOBIT set will cause a loss of the additional work-space and a loss of system overflow space.

Output Interface

As for TSINIT and WSINIT above;

ISBEG, IS	Point to PCB + 16
ISEND	= ISBEG + 3000
OSBEG, OS	Point to PCB + 22
OSEND	=OSBEG + 3000
HSBEG	Points to PCB + 10
HSEND	=HSBEG

PERIPHERAL I/O

Tape Control Subroutines

These routines provide for passing control commands to the magnetic tape unit. As in all tape commands, it is assumed that the tape unit is "attached" to the process executing these routines; this is flagged by the bit ATTACH being set. Conventionally, ATTACH should only be set by executing the verb T-ATT from the TCL level.

INIT and TPSTAT

All tape i/o routines use the INIT and TPSTAT subroutines. INIT outputs a function-code of "1" to the tape controller, thereby setting it to an initial condition, and then falls through into the tape status from the controller. It will return only if the tape is in a "ready" state. If the tape is rewinding, the subroutine will wait till it finishes. Otherwise, the status is tested up to one hundred times; if the tape unit is still not ready, an exit is taken to MD99 with error message 95 (NOT ON-LINE).

Input Interface

None

Internal Usage

T6	T	Used as a delay counter

Output Interface

REJCTR	T	Zero

Tape status bits are as below:

EOFBIT	B	Set if an end-of-file mark is reached.
EOTBIT	B	Set if the tape is at load point, or at the end-of-tape marker.
PARITY	B	Set if a parity error is detected.
NORING	B	Set, on a write operation, if the write ring in the tape is not present.

WEOF

Writes and end-of-file mark on the tape.

BCKSP

Back spaces the tape by one record.

REWIND

Rewinds the tape unit.

FRWSP

Forward spaces the tape by one record; this subroutine destroys location X'1FF' in the PCB.

Tape I/O Routines

TPREAD reads one record from the tape to a buffer defined by R15; the read stops either when the inter-record gap in the tape is detected, or at the end of the frame to which R15 points.

TPWRITE writes one record from the buffer defined by R15 to the magnetic tape; the write will always continue until the end of the frame to which R15 points.

A maximum of 512 bytes may be transferred by these routines.

Input Interface

ATTACH	B	Must be set, indicating tape unit is attached.
R15	R	Points to first byte of buffer area.

Output Interface

R15	R	Points to last byte read (TPREAD). In the case of a read where the tape record is shorter than the buffer, R15 points one byte past the last data byte.

Tape status bits set appropriately.

Subroutines Used

INIT, TPSTAT (local); uses two additional levels of subroutines linkage.

Error Conditions

Read parity error: the read is repeated ten times; if the parity error persists, an exit is taken to MD99 with error number 98.

Write parity error: the write is re-tried once; then the sequence - backspace / write end-of-file mark / backspace and repeat write - is tried nine times; if the parity error persists, an exit is taken to MD99 with error number 98.

Also see INIT and TPSTAT.

Blocked Tape I/O Operations

The routines ITPIB, TPIB, OBTP and FOBTP allow reading and writing variable length records, blocked in fixed-length 500-byte records. These routines use the first two frames of the OS workspace to block and de-block; the unblocked data is passed to the write routine (OBTP) in the OB; it is passed from the read routine (IBTP) in the IB.

Reading a blocked tape: An initial call must be made to ITPIB to initialize the de-blocking pointers; subsequently, each call to TPIB will return one tape record.

Writing a blocked tape: The data to be written to the tape is placed in the OB, and OBTP is called to store it in the blocking area. When the output is to be terminated, one call to FOBTP must be made to clear the blocking area and force the data to be written to the tape.

These routines use the delimiter SB (X'FB') as the block delimiter; therefore, SB's in the data to be written to tape are converted to blanks before being output.

Note interface equivalence of these routines with the corresponding terminal i/o routine.

Input Interface

ITPIB	:	OSBEG	S	Points are prior to deblocking buffer.
TPIB	:	IBBEG	S	Points one prior to buffer area where deblocked data is to be copied.
OBTP	:	OBBEG	S	Points one prior to buffer area containing data to be blocked.
		OB	R	Points to last byte of data.
FOBTP	:	NONE		

Internal Usage (All Routines)

OSBEG	S	Points to a linked work-space used by the blocking and de-blocking routines. Must be at least 3 frames.

OS	R	Current pointer in blocked data area; must be maintained.
SC2	C	Scratch (OBTP, FOBTP only).

Output Interface

ITPIB : OS	R	Points to OSBEG; first two tape records have been read into the OS frames.
TPIB : IB	R	Points to IBBEG, one prior to de-blocked data.
IBEND	S	Points to a SM following last byte of data.
OBTP : OB	R	Reset to OBBEG
FOBTP : NONE		

Subroutines Called

TPREAD or TPWRITE; three additional levels of subroutine linkage.

Errors

See preceding documentation.

SEGMNT (3,TAPEIO-II)

This subroutine is used by the File-Restore, Sel-Restore, and Acc-Restore processors to de-block data from a File-Save tape. The built-up "segment" (data between segment marks on the tape) is stored in the IS.

Input Interface

S1	S	Points to location, within IB, of the last SM found. On initial entry, this is setup by the calling mode; it must be maintained between calls to SEGMNT.

Output Interface

ISBEG	S	Buffer where the segment has been copied.
IS	R	=ISBEG
XMODE	T	Zero

Subroutines Used

TPIB

LABELED TAPE i/o ROUTINES

Label format:

(SM)L...label data...(VM) time date (AM) reel # (AM) (SM)

The label is stored in the quadrenary control block (PCB + 3); displacements to various elements are as below:

Byte Displacement	Type	Description
1A5 (Bit Ø)	B	"unlabeled tapes in use" flag
1A6	T	Reel number
1A8	L	Label save buffer (46 bytes)
1D6	L	Label write/read buffer (30 bytes)

Since the tape-i/o routines are non-reentrant, internal storage is utilized when an EOT condition is handled by the tape write or read subroutines.

These routines save R13, R14, R15 in internal save areas (defined in TAPEIO-II), and set up R13 to displacement X'1A6' in the quadrenary control block in order to address elements in that block.

R13, R14 and R15 are restored on exit.

RDLABEL (2,TAPEIO-II)

May be called once by any program to read the label from reel #1; if the tape is labeled, the label is stored in the save area; if not, the "unlabeled tapes in use" flag is set. If the tape is not at the load point, no action is taken. No input interface. On output, the label save area is set up.

RDLABELX (5,TAPEIO-II)

As for RDLABEL, except that no check is made to see if the tape is at the load point. This routine is used by the FILE-RESTORE processor (ABSL), to read a label even though the tape may be positioned past the load point. May be used by user-programs, though the implications of doing so should be kept in mind.

WTLABEL (2,TAPEIO-III)

May be called once by any processor to write a label on reel #1; no action is taken if the tape is not at load point. The label (if any) passed as an input parameter is written to the tape, with the current time and date, and reel number one, added. The label is also stored in the label save buffer.

Input Interface:

IS	R	Points one before the label data, which must be terminated by any standard system delimiter. The label cannot be greater than 16 characters; it will be truncated to 16 if it is. If a null label is submitted, no label is written to the tape, and the "unlabeled tapes in use" flag is set.

Output Interface

IS	R	Points to delimiter terminating label, or to 16 bytes beyond the input position if none is found.

Label save area initialized.

WTLABELX (4,TAPEIO-III)

As for WTLABEL, except that no check is made to see if the tape is not set to load point. This routine is used by the FILE-SAVE processor (ABSD), to write a label at the current position of the tape. May be used by user-programs, though the implications of doing so should be kept in mind.

CREAD

Functional Description

The subroutine either reads a card and returns the card reader status after the read or it just returns the status if it cannot read a card. Cards are read in EBCDIC and are not converted by this routine.

Input Interface

R2	R	Must point to a scratch byte. (Typically R2 will always point to byte zero of the SCB.)
OBBEG	S	Points anywhere within the frame that the card is to be read into. (Typically OBBEG will always point within PCB+4).

Internal Usage

T3	T	Used as a counter for status timeout after a read.

Output Interface

R2	R	Unchanged. The byte that R2 points to contains the status of the card reader.
CBIT	B	Zero if no card was read. Set if an attempt to read a card was made.
R15	R	Points to first byte of card read, 80 bytes from the end of the frame that OBBEG points to.

Errors

None, except card reader errors returned as status. The status bits are as follows:

Bit	Explanation of the set condition.
0-2	Unused by the controller. Will be zero.
3	Card reader mechanical error (e.g., pick failure, card motion error, etc.)
4	EBCDIC error detected, (e.g., an invalid punch combination was detected.) Not an error if CBIT is zero.

5	Input hopper empty. Not an error if CBIT is set.
6	This bit is always zeroed by the routine. It is only used for byte I/O.
7	Card reader ready.
3,5,7	If bits 3, 5, and 7 are all set, this indicates that power is off on the card reader.

MISCELLANEOUS

Some of these subroutines are primarily used by system processors, and, therefore may use elements other than the minimum set used by the general-purpose system subroutines.

TIMDATE, TIME AND DATE

Functional Description

These routines obtain the system time and/or the system date, and store it in the buffer area specified by R15. The time is returned as on a 24-hour clock.

Entry	Buffer Size Required	Format
TIME	8	HH:MM:SS
DATE	11	DD MMM YYYY
TIMDATE	21	HH:MM:SS DD MMM YYY

Input Interface

R15	R	Points one prior to the buffer area.

Output Interface

R15	R	Points to last byte of data stored.

Subroutines Used

Entry TIMDATE uses TIME; requires two additional levels of subroutine linkage; other entries require one level.

Errors

None

ASCII-Character to Binary Conversion

Functional Description

The routines described below will convert a string of ASCII decimal or hexadecimal characters to their binary equivalent; the conversion continues until an illegal (non-decimal or non-hexadecimal) character is encountered.

On entry, the appropriate register (see table) points either to a non-numeric character, one prior, or to the first character of the string, which must be a plus sign, a minus sign or an appropriate numeric (0-9 for the decimal routines, 0-9 and A-F for the hexadecimal routines). On return, the converted binary number is in the accumulator (and in some cases, in CTR1); the register points to the illegal character causing the conversion to terminate. Note that the register will always be incremented by one even in the case of a null string (no legal characters). Arithmetic overflow due to too many digits in the character string cannot be detected.

Entry Name	Register Used	Conversion From: Dec.	Hex.	Value Returned IN: Accumulator	CTR1
CVDR15	R15	X		X	
CVXR15	R15		X	X	
CVTNIS	IS (R4)	X		X	X
CVTHIS	IS (R4)		X	X	X
CVTNOS	OS (R5)	X		X	X
CVTHOS	OS (R5)		X	X	X
CVTNIR	IR (R6)	X		X	X
CVTHIR	IR (R6)		X	X	X
CVTNIB	IB (R10)	X		X	X
CVTHIB	IB (R10)		X	X	X

Subroutine Used

CVDR15 or CVXR15 are called by the other routines; one additional level of linkage is required.

Binary to ASCII-Character Conversion

MBDSUB AND MBDNSUB

Functional Description

These routines will convert a binary number to the equivalent string of decimal ASCII characters. The conversion will store a minimum number of characters (that is, leading zeroes will be padded if needed) if the entry MBDNSUB is used; if MBDSUB is used, only as many characters as are needed to represent the number will be stored. A minus sign will precede the character string if the number to be converted is negative.

These subroutines are implicitly called by the assembler instructions MBD (move binary to decimal) and MBDN.

Input Interface

D0	D	(Accumulator) Contains number to be converted.
R15	R	Points one prior to buffer where converted characters are to be stored (maximum 9 characters).
T4	T	(MBDNSUB entry only) Contains minimum number of characters to be stored.

Output Interface

R15	R	Points to last converted character.

Subroutines Used and Error Conditions:

None.

EBCDIC to ASCII Conversion

Functional Description

EBCDIC to ASCII conversion - ECONV

The register IB points to the EBCDIC character; a call to ECONV converts it to ASCII; characters that cannot be converted are returned as a question mark (?).

String EBCDIC to ASCII with move - R.ETA.M

The registers R15 and R8 point to the first character of the source and destination strings, respectively. CTR1 contains the character count. R13, R14, and T0 are destroyed. All characters are converted. CTR1 should be zero on return. R15 and R8 point to the last character of their respective strings on return.

String ASCII to EBCDIC with move - R.ATE.M

Same as R.ETA.M except ASCII to EBCDIC translation.

File-initialization

DLINIT (6,DLOAD)

Functional Description

Obtains a block of contiguous overflow space for a file; links the frames and sets up initial conditions using the routine DLINIT1 (described below)

Input Interface

MODULO	T	Contains the modulo required for the file.
SEPAR	T	Contains the separation required for the file; if SEPAR is greater than 127, it will be reset to one.

Output Interface

BASE	D	Contains the beginning FID of a contiguous block of size MODULO*SEPAR. If BASE=0, the system does not have sufficient overflow space.

Note: This subroutine automatically enters DLINIT1 if the overflow space is obtained.

Subroutines Used

GETBLK; two additional levels of subroutine linkage.

DLINIT1 (7,DLOAD)

Functional Description

Initializes link fields of a file as specified by its base, modulo and separation parameters; sets each group empty by adding an AM at the beginning.

Input Interface

BASE	D	Contains base-FID, modulo and separation of file.
MODULO	T	
SEPAR	T	

Internal Usage

CTR1	T	
RECORD	D	

Output Interface

R14	R	As returned by subroutine LINK
R15	R	

Subroutines Used

LINK; RDREC; two additional levels of linkage required.

System Accessing Routines

These subroutines may be used to setup pointers to system-files (SYSTEM,ACC), to the PIB's, etc.

GPCB0 (4,ABSL)

Functional Description

Returns the PCB-FID for channel zero in the accumulator.

Input Interfaces

None

Output Interfaces

D0	D	Contains FID of PCB for channel zero. High order 16 bits are zero.

SETPIB (4,LOGON)

Functional Description

Sets up R14 to point to the first byte of the PIB associated with the process. No input interface.

Internal Usage

SR5	S	Scratch

Output Interface

R14	R	As described.

SETPIBF (3,ABSL)

Functional Description

Sets up R14 to point to the first byte of the PIB associated with channel zero. No input interface.

Output Interface

R14	R	As described.

GMMBMS

Functional Description

Sets up pointers to the SYSTEM dictionary (formerly called MM/DICT). No Input Interface.

Output Interface

BASE	D	Contains base-FID, modulo and separation of SYSTEM dictionary.
MODULO	T	
SEPAR	T	

GACBMS (1,LOGOFF)

Functional Description

Sets up pointers to the ACC dictionary. No Input Interface.

Internal Usage

SR1	S	Scratch
T6	T	Scratch

All elements as used by GBMS

Output Interface

BASE	D	Contains base-FID, modulo and separation of the ACC dictionary (actually a file, since ACC as defined in SYSTEM is a Q-entry to the DL/ID of ACCOUNT)
MODULO	T	
SEPAR	T	
REJ1	T	Contains the value 331 if the ACC file is missing.

Subroutines Called

GMMBMS; GBMS; two additional levels of linkage required.

GETOPT (10,SYSTEM-SUBS-II)

Functional Description

This program converts an option string consisting of single alphabetic characters or a numeric specification. Alphabets set the corresponding bit (A sets ABIT, etc.). Multiple options are separated by commas, and the string must be terminated by a ")". ABIT through ZBIT are not zeroed on entry.

Input Interface

IS	R	Points one before the option string.

Output Interface

ABIT through ZBIT		Set as described above.
NOBIT	B	Set if numeric option is found; zero otherwise.
RMBIT	B	Set
D4	D	Contains numeric values if numeric options is found; unchanged otherwise. D4 contains first numeri·, D5 second if found, otherwise the same value as D4.

Error Conditions

Exits to MD99 with error 209 after setting RMODE zero if a format error is encountered.

GETUPD

Functional Description

Sets up the register triad UPDBEG,UPD and UPDEND, in unlinked format, to PCB+28. A convenient way to setup a register to a buffer (also used by RPG). Note that the UPD registers are treated as a scratch register by some system subroutines. No Input Interface.

Output Interface

UPBEG	S	Points to byte zero of PCB+28
UPD	R	=UPDBEG
UPDEND	S	Points to last byte of PCB+28.

XISOS

Exchanges the register triads ISBEG/IS/ISEND and OSBEG/OS/OSEND.

PRIVTST1 (5,SYSTEM-SUBS-III)

Tests if the process has system privileges, level one; exits to MD99 with error 82 after setting RMODE zero, clearing PQFLG and LISTFLAG, and setting the history string null if not.

PRIVTST2 (7,SYSTEM-SUBS-III)

As above, for system privileges, level two.

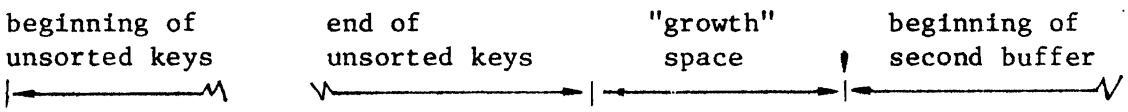
SORT

Functional Description

Sorts an arbitrarily long string of keys in ascending sequence only; the calling program must complement the keys if a descending sort is required. The keys are separated by SM's when presented to SORT; they are returned separated by SB's. Any character, including system delimiters other than the SM and SB may be present within the keys.

An n-way polyphase sort-merge sorting algorithm is used. The original unsorted key string may "grow" by a factor of 10%; and a separate buffer is required for the sorted key string, which is about the same length as the unsorted key string. The "growth" space is contiguous to the end of the original key string; the second buffer may be specified anywhere. The SORT subroutine will automatically obtain additional overflow space and link it if needed.

Due to this, one can follow standard system convention and build the entire unsorted string in an overflow table with OVRFLCTR containing the beginning FID; the setup is then:



The second buffer pointer then is merely set at the end of the "growth" space, and SORT allowed to obtain additional space as required.

Alternately, the entire set of buffers may be in the IS or OS workspace if they are large enough.

Input Interface

SR1	S	Points to SM preceeding first key.
SR2	S	Points to SM terminating last key.
SR3	S	Points to beginning of second buffer area.

Internal Usage

Entire BMS work area.

S1 through S9: Scratch

BMS	R
CS	R
IS	R
OS	R
TS	R
HBIT	B
LBIT	B
SB1	B

Output Interface

SR1	S	Points two bytes before SB preceding first sorted key. The sorted keys are delimited by SB's, and the entire string terminated by an SM.

Subroutine Usage

Internal call to COMP; ATTOVF if end of second buffer is reached. One additional level of linkage required.

BLOCK-LETTERS

Functional Description

This program provides the block letter capability (see BLOCK-TERM and BLOCK-PRINT documentation.) In addition to its use at the verb level, it may be called as a subroutine (DEFM 2,290). It will format a string of words on the terminal of the printer and return to the caller.

Input Interface

ZBIT	If set direct output to the terminal; if not set, direct output to the printer
IS	Points one character prior to the first character to be output; end of data is indicated by the character pair SM,Z, if a segment mark is present in the string not followed by a "Z" the string must be terminated by a switch block (SB) X'FB' (see TCL-I interface).
PAGSIZE	Maximum number of lines per page.
OBSIZE	Maximum number of characters on each output line.

Internal Usage

The following functional elements are used and not restored.

SC2, SCL REJCTR, CO, PAGINATE, BASE, MODULO, SEPAR, CTR16, CTR17, CTR18, SR5, SR6, SR7, SR8, SR9, SR10, SR11, SR12, SR13, SR14, SR15, SR16, SR17, SR18, SR19, SR20, SR21, SR22, AFEND, LPBIT, ZBIT

Subroutines Called:

RETIX, GBMS, NEWPAGE, CVTNIR, WRTLIN

Error Conditions:

See verb write-up on BLOCK-TERM and BLOCK-PRINT.

ENGLISH INTERFACES.

Selection Processor.
LIST Processor.
CONVERSION Processor.
FUNCTION Processor.

ENGLISH Interface

It is possible to interface with the ENGLISH processor at several levels. A typical LIST on SORT statement passes through the Pre-processor and Selection Processor before entering the LIST processor. All statements must pass through the first two stages; but control can be transferred to user-programs from that point onward.

General Conventions

The ENGLISH processors use a compiled string that is stored in the IS work-space. String elements are separated by segment marks; there is one element for each attribute specified in the original statement; one file-defining element, and special elements pertaining to selection criteria, sort-keys, etc.

Formats:

File-Defining Element; at ISBEG+1

(SM)D file-name (AM) base (VM) modulo (VM) separ (AM) conv. (AM) correl. (AM) type (AM) just. (AM) (SM).

Attribute-Defining Element

(SM) c attribute-name (AM) amc (AM) conv. (AM) correl. (AM) type (AM) just. (AM) (SM)

c = A - regular or D2 attribute.

Q - D1 attribute.

BX - SORT-BY, SORT-BY-DSND, etc. "X" is from attribute one of the connective.

End-of-String Element

(SM)Z

Explicit Item-id's

(SM)I item-id (SM)

The Selection Processor

This performs the actual retrieval of items which pass the selection criteria, if specified. Every time an item is retrieved, the processor

at the next level is entered with RMBIT set; a final entry with RMBIT zero is also made after all items have been retrieved. If a sorted retrieval is required, the Selection processor passes items to the GOSORT mode, which builds up the sort-keys preparatory to sorting them. After sorting, GOSORT then retrieves the items again, in the requested sorted sequence.

A user-program may get control directly from the selection processor (or GOSORT if a sorted retrieval is required); the formats of the verbs are:

Line Number	Non-Sorted	Sorted
001	PA	PA
002	35	35
003	xxxx	76
004		xxxx

where "xxxx" represents the mode-id of the user-program. Note that, in this method of interface, only item retrieval has taken place; none of the conversion and correlative processing has been done. For functional element interface, the column headed "Selection Processor" in the table shown later must be used.

Exit convention: On all but the last entry, exit indirectly via RMODE (using ENT* RMODE); on the last entry, exit to one of the WRAPUP entry points. Processing may be aborted at any time by setting RMODE zero and entering WRAPUP. SBO must be set on first entry.

Special Exit From The LIST Processor

A user-program may also gain control in the place of the normal LIST formatter, to perform special formatting. The advantage here is that all conversions, correlatives, etc. have been processed, and the resultant output data has been stored in the history string (HS area). The formats of the verbs that are:

Line Number	Non-Sorted	Sorted
001	PA	PA
002	35	35
003	4D	4E
004	xxxx	xxxx

Where "xxxx" is the mode-id of the user-program.

History-String Format: The output data is stored in HS area; data from each attribute specified is stored in the string, delimited by AM's; multiple values and sub-multiple values are delimited within by VM's and SVM's respectively. Since the HS may contain data other than the retrieved item, the user-program should scan from HSBEG, looking for a segment preceded by an "X"; all segments except the first are preceded by an SM.

X item-id (AM) value one (AM)...(AM) value n(AM)(SM)Z

The program must reset the history string pointer HSEND, as items are taken out of the string. In special cases, data may not be used till, say, four items are retrieved, in which case HSEND is reset on every fourth entry only. HSEND must be reset to point one byte before the next available spot in the HS; normally one before the first "X" code found.

Exit convention: see preceding section.

Example: The following program is an example of one which prints item id's (only) four-at-a-time across the page.

```
001             FRAME   504                 INTERNAL FLAG
002             ZB      SB30                INTERNAL FLAG
003             BBS     SB0,NOTF            NOT FIRST TIME
004  * FIRST TIME SETUP
005             MOV     4,CTR32
006             SB      SB0
007  *
008  NOTF       BBZ     RMBIT,PRINTIT       LAST ENTRY
009             BDNZ    CTR32,RETURN        NOT YET 4 ITEMS OBTAINED
010             MOV     4,CTR32             RESET
011  PRINTIT    MOV     HSBEG,R14
012  LOOP       INC     R14
013             BCE     C'X',R14,STOREIT    FOUND AN ITEM
014             BCE     C'Z',R14,ENDHS      END OF HS STRING
015  SCANSM     SCD     R14,X'C0'           SCAN TO NEXT (SM)
016             B       LOOP
017  STOREIT    BBS     SB30,COPYIT         NO FIRST ID FOUND
018             SB      SB30                FLAG FIRST ID FOUND
019             MOV     R14,SR28            SAVE LOCATION OF FIRST "X"
020  COPYIT     MIID    R14,OB,X'A0'        COPY ITEM-ID TO OB
021             MCC     C' ',OB             OVERWRITE (AM)
022             INC     OB,5                INDEX
023             B       SCANSM
024  ENDHS      BSL     WRTLIN              PRINT A LINE
025             MOV     SR28,HSEND          RESTORE HSEND TO FIRST "X" CODE
026             DEC     HSEND               BACKUP ONE BYTE
027             BBZ     RMBIT,QUIT
028  RETURN     ENT*    RMODE               RETURN TO SELECTION PROCESSOR
029  QUIT       ENT     MD999               TERMINATE PROCESSING
030             END
```

Functional Element Usage

The following table summarizes the functional element usage by the Selection and LIST processors. Only the most important usage is described; elements that have various usages are labeled "scratch". A " " indicates that the processor does not use the element. Since the LIST processor is called by the Selection processor, any element used for "memory" purposes (not to be used by others) in the former is indicated by a blank usage in the latter column.

In general, user-programs may freely use the following elements:

```
Bits   :   SB20 upwards
Tallys :   CRT30 upwards ; D3-D8
S/R's  :   SR20 upwards
```

SB0 and SB1 have a special connotation; they are zeroed by the Selection processor when it is first entered, and not altered thereafter. They are conventionally used as first-time switches for the next two levels of processing. SB0 is set by the LIST processor when it is first entered, and user-programs that gain control directly from Selection should do the same. SB0 may be used as a first-entry switch by user-programs that gain control from the LIST processor.

An ENGLISH verb is considered an "update" type of verb of the SCP character (from line one of the verb definition) is A, B, C, D, E, G, H, I or J. SCP characters of B, C, D and E are reserved for future ENGLISH update verbs.

BITS	Selection Processor	LIST Processor
ABIT	unused	non-columnar list
BBIT	first entry flag	
CBIT	scratch	scratch
DBIT	scratch	dummy control-break
EBIT	reserved	control-break flag
FBIT	reserved	scratch
GBIT	reserved	scratch
HBIT	reserved	scratch
IBIT	explicit item-ids specified	
JBIT	reserved	D2 attribute in process
KBIT	scratch	scratch
LBIT	scratch	left-justified field
MBIT	conversion interface; zero	zero
NBIT	scratch	scratch
OBIT	selection test on item-id	
PBIT	scratch	scratch
QBIT	scratch	scratch
RBIT	full-file retrieval flag	
SBIT	selection on values (WITH)	
TBIT	scratch	print limiter flag

BITS	Selection Processor	LIST Processor
UBIT	scratch	reserved
VBIT	reserved	scratch
WBIT	scratch	reserved
XBIT	scratch	reserved
YBIT	left-justified value being tested	left-justified print-limiter test
ZBIT	left-justified item-id	
SB0	Unavailable	first entry flag, level one
SB1	Unavailable	first entry flag, level two
SB2	reserved; zero	
SB4 through SB16	scratch or reserved	scratch or reserved
VOBIT	set for WRAPUP interface	
COLHDRSUPP DBLSPC HDRSUPP IDSUPP LPBIT CBBIT PAGINATE	set if corresponding connective was found in input statement	
RMBIT	set on exit if an item was retrieved; zero on final exit.	
SMBIT	FUNC interface	FUNC interface
GMBIT	FUNC interface	FUNC interface
BKBIT	scratch	scratch
DAF1	set if SCP = B,C,D,E,G, H,I or J	
DAF8	set if accessing a dictionary	

Tallys	Selection Processor	LIST Processor
C1;C3-C9	Scratch	Scratch
C2	contents of MODEID2	
CTR1-CTR4	Scratch	Scratch
CTR5	Scratch	AMC of current element in IS
CTR6	reserved	Scratch
CTR7	reserved	AMC corresponding to IR
CTR8	reserved	Scratch
CTR9	reserved	Scratch
CTR10	reserved	Scratch

Tallys	Selection Processor	LIST Processor
CTR11	reserved	Scratch
CTR12	FUNC interface	current-sub-value count
CTR13	FUNC interface	current value count
CTR14	reserved	Scratch
CTR15	reserved	Scratch
CTR16	reserved	Contents of item count file
CTR17	reserved	reserved
CTR18	reserved	Scratch
CTR19	reserved	Scratch
CTR20	CONV interface	CONV interface
CTR21	CONV interface	CONV interface
CTR22	CONV interface	CONV interface
CTR23	CONV interface	CONV interface
CTR24	reserved	Scratch
CTR25	reserved	Scratch
CTR26	reserved	Scratch
CTR27	reserved	current max-length
CTR28	reserved	Scratch
D9	count of retrieved items	
FP1-FP3,D7	FUNC interface	FUNC interface
RMODE	return mode-id (MD30)	
SIZE	item-size	Scratch
SBASE SMOD SSEP	b,m,s of file	
DBASE DMOD DSEP	b,m,s of dictionary	

S/R's	Selection Processor	LIST Processor
S1	Points to next explicit item-id	
S2-S9	Scratch	Scratch
SR0	Points one before count field of item	
SR1	Points to correlative field	Current correlative field
SR2	Scratch	Scratch
SR3	reserved	Scratch
SR4	Points to last AM of item	
SR5	reserved	Next segment in IS
SR6	Points to conversion field	Current conversion field
SR7	reserved	Scratch

S/R's	Selection Processor	LIST Processor
SR8	reserved	reserved
SR9	reserved	reserved
SR10	reserved	Scratch
SR11	reserved	reserved
SR12	reserved	reserved
SR13	GOSORT only: next sort-key	reserved
SR14	reserved	reserved
SR15	reserved	reserved
SR16	reserved	reserved
SR17	reserved	reserved
SR18	reserved	reserved
SR19	reserved	reserved
PAGHEAD	Heading in HS if HEADING was specified	generated heading in HS
Workspace pointers	See section on work-space usage	

A/R's	Selection Processor	LIST Processor
AF	Scratch	Scratch
BMS	within BMS area	Scratch
CS		Scratch
IB		Scratch
OB		Output data line
IS	compiled string	compiled string
OS		Scratch
TS	within TS area	within TS area
UPD		within HS area
IR	within item	within item

Work-Space Usage	Selection Processor	LIST Processor
AF	Scratch	
BMS	Contains item-id	
CS		
IB		
OB		Output line.
IS	Compiled string	
OS		Scratch

Work-Space Usage	Selection Processor	LIST Processor
HS	Heading data	Heading data, attribute data for special exits.
TS	Scratch	Current value in process.

Additional Notes

1. If a full-file retrieval is specified, the additional internal elements as used by GETITM will be used. If explicit item-id's are specified, RETIX is used for retrieval of each item.

2. Elements as used by the FUNC and CONV processors have been shown in the table; both may be called either by the Selection processor or the LIST processor.

3. Since the ISTAT and SUM/STAT processors are independently driven by the Selection processor, the element usage of these processors are not shown.

4. The section of the IS and OS used by the Selection and LIST processors is delimited by ISEND and OSEND respectively. The buffer space beyond these pointers is available for use by other programs.

BATCH Processor Interface

The BATCH processor uses a BATCH- string which defines the method of updating one or more items in one or more files using a single line of input data. The updated item(s) is(are) built as disc-update string(s) in the history string area (see WRAPUP for format).

A user-exit can be defined in the BATCH-string; the functional elements used by BATCH are described in the following tables; the column headed "Level" has the following entries:

O - Element is used in the described fashion throughout the BATCH processing.

F - Element is redefined every time a file-defining element is found.

A - The element is redefined for every attribute.

blank - Scratch element, on reserved for future usage.

As far as user-exit programs are concerned, therefore, all elements defined at the "A" level can also be considered scratch.

Exit Convention: The user-exit must return to the BATCH processor by executing the external transfer to the mode BATCH5 (DEFM 0,84).

Bits	Level	Description
ABIT	0	First-time switch for BATCH process.
BBIT		reserved
CBIT		Scratch
DBIT	A	D2 attribute in process.
EBIT	F	Updates to be merged with item on file.
FBIT	0	Set when a BV or BC Arb-element is found.
GBIT		reserved
HBIT	A	D1 attribute in process.
IBIT	0	Set when a "secondary" file.
JBIT		reserved
KBIT	F	Item to be verified as existing on file.
LBIT	F	Item to be verified as not existing on file.
MBIT	A	Set; CONV interface.
NBIT		reserved
OBIT		reserved
PBIT	F	Flag indicating that a multi-valued field referenced by BC/BV element.
QBIT		reserved
RBIT		reserved
SBIT		scratch
TBIT		scratch
UBIT	0	Item is being deleted (X element in file-definition)
VBIT		scratch
WBIT		scratch
XBIT		reserved
YBIT	0	Primary item being deleted.
ZBIT		scratch
SB1 through SB9		scratch
DAF10	0	Set if SELECT/SSELECT is driving BATCH.

Tallys	Level	Description
C1		scratch
C2		scratch
C3-C9		reserved
CTR1		scratch
CTR2		scratch
CTR3		scratch
CTR4	A	D1-D2 set number (follows D1 or D2 element)
CTR5		reserved
CTR6		reserved
CTR7		scratch
CTR8	F	Current amc in process.
CTR9		reserved
CTR10		reserved
CTR11	F	Value no. of "D1;1" attribute; 0 if unspecified.
CTR12	F	Value no. of "D1;2" attribute; 0 if unspecified.
CTR13	F	Value no. of "D1;3" attribute; 0 of unspecified.
CTR14-CTR19		reserved
FP1-FP3		scratch
BASE		
MODULO		scratch
SEPAR		scratch
SBASE		scratch
SMOD		scratch
SSEP		scratch
D7		scratch
D9		scratch
RMODE	0	Return mode-id for WRAPUP

Work-Spaces & A/R's	Level	Description
BMS	A	Work-space contains current value
CS		Scratch; work-space reserved.
AF		Unused
IB	0	Input data line
OB		Unused
TS	0	Used for reading input lines.
IS	0	Contains BATCH string; IS points to AM before next element.
OS		Scratch work-space
UPD	0	Points to history-string

S/R's	Level	Description
S1-S9		Scratch
SR0	0	One before count field of primary item on file.
SR1	0	End of primary item on file
SR2		scratch
SR3		reserved
SR4	F	End of current item on file.
SR5		reserved
SR6		reserved
SR7	0	End of OS deletion table
SR8		reserved
SR9	A	Last byte of value in BMS area.
SR10	F	
SR11	0	End of primary update string if FBIT set.
SR12	0	Points one before "DU" of history string for primary item update.
SR13		reserved
SR14	F	Location of file-defining element in IS
SR15	F	Location of IB when current file-defining element was found.
SR16-SR19		reserved.

Characters	Level	Description
SCP	0	Contains a "D" for B/DEL; "A" for B/ADD
SC0	0	Contains a blank.
SC1	0	Scratch
SC2	0	Contains a comma.

Also note elements used by CONV processor.

CONVERSION Processor Interface

The Conversion processor is called as a subroutine (CONV DEFM 0, 90) and may be used to perform the Translate, Date, or Mask Conversions. More than one conversion can be performed at once if the conversion string is set up to do so; multiple conversion codes are separated by VM's. Conversion is called by the ENGLISH pre-processor to perform conversions on "input" data (in selection criteria), and by the LIST/SORT processor to perform "output" conversion.

Input Interface

MBIT	B	Set if an "input" conversion is to be performed; zero for an "output" conversion.
TSBEG	S	Points one before the value to be converted; the value is converted "in place", and the buffer is used for scratch spare; therefore it must be large enough to contain the converted value. The value to be converted is terminated by any of the standard system delimiters; SM, AM, VM, SVM.
IS	R	Points to the first character of the conversion code specification string; the code(s) must be terminated by an AM.
BMSBEG	S	Used for item-id copy on Translate conversions.

Internal Usage

SB10	B
SB11	B
SB12	B
SC2	C
CTR20	T
CTR21	T
CTR22	T
CTR23	T

S4	S	Scratch; used to save and restore various elements.
S5	S	
S6	S	
S7	S	

Output Interface

IS	R	Points to AM terminating the conversion code(s).
TSBEG	S	Points one before converted value.
TS	R	Points to the last character of the converted
TSEND	S	value; a SM is also placed one past the value. If a null value is returned, TS=TSEND=TSBEG.

If a Translate conversion is used, subroutines GBMS, GDLID, and RETIX are used. Thus all elements used by those subroutines will be destroyed, with the exception of IR, SR4 and SIZE, which are restored before exit to their values or entry.

Subroutines Used: GBMS, GDLID, RETIX (T-conversion only); MBDSUB, CVDR15

Error Exits:

CONV will exit to WRAPUP after setting RMODE zero under the following conditions:

705	Illegal conversion code
706	Illegal T-conversion: format incorrect, filename cannot be found, etc.
707	DL/ID cannot be found for T-conversion file.

WRAPUP is also entered, without setting RMODE zero, under the following error conditions:

708	Value cannot be converted by the T-conversion.
339	Invalid format for input data conversion.

User Conversion Processing

The Conversion Processor will pass control to a user-written routine if a "Uxxxx" code is found in the conversion string, where "xxxx" is the hexadecimal mode-ID of the user-routine. This routine can then perform special conversion before returning. The input interface at the user-routine will be identical to that described in the preceding section; after performing the conversion the user-routine should setup the

output interface elements described under CONVERSION, and exit via an external branch to 1,CONV, which will continue the conversion process if multiple conversions are specified; a RTN may be executed if this is not needed, or to prevent further conversions being performed. Elements used by the regular conversion processors may be safely used by user-routines; however, if additional elements are needed, a complete knowledge of the processor that called CONVERSION (LIST, SELECTION, etc.) will be necessary.

FUNCTION Processor Interface

The FUNCTION processor is used by the ENGLISH LIST/SORT processors to compute values which have an "F" correlative specified. It may also be called, as a subroutine, by user routines. Each call to FUNC (DEFM 0,101) returns one value.

Input Interface

SR1	S	Points to the "F" of the Function string.
SR0	S	Points one before the count-field of the item.
SR4	S	Points to the last AM of the item.
CTR13	T	Contains the "value number" currently being processed (one on initial entry).
CTR12	T	Contains the "sub-value number" (D2 sub-value) currently being processed.
TSBEG	S	Points to a buffer area-350 where the value is to be stored on exit.

Internal Usage

SMBIT	B
GMBIT	B
CTR1	T
CTR12	T
CTR20	T
CTR21	T
IR	R
IS	R
D7	D
FP1-FP3	D

Output Interface

IR	R	Points one before the value copied (TSBEG+350); the value is delimited by an AM if none of the referenced fields contained multiple or sub-multiple values; by a VM if at least one of the referenced fields contained a VM on this entry by A SVM if at least one of the referenced fields contained a SVM on this entry.
R15	R	Points to a blank following the terminal delimiter of the value.
IS	R	Points to the AM or one past a VM, terminating the Function string.
D0,FP1	D	Contains final computed result.

Programming Note

On the first call to FUNC, CTR12 and CTR13 are both set to one; when FUNC returns a value, the terminal delimiter determines what action to take or subsequent calls--a VM indicates increment of CTR13 before the next call; a SVM indicates increment of CTR12; an AM indicates end of processing:

```
          .
          ONE   CTR13        SET VALUE NUMBER TO ONE
FC1       ONE   CTR12        SET SUB-VALUE NUMBER TO ONE
FC2       BSL   FUNC
          .
          .
          store value from IR
          .
          .
          DEC   R15
          BCE   AM,R15,END   END OF PROCESSING
          INC   CTR12        INCREMENT SUB-VALUE COUNT
          BCE   SVN,R15,FC2  GET NEXT SUB-VALUE
          INC   CTR13        INCREMENT VALUE COUNT
          B     FC1          GET NEXT VALUE; RESET SUB-VALUE COUNT
END       EQU   *            CONTINUE
```