REALITY®

by Microdata

Assembly Language
Programming Manual

REALITY®
(3.0 SERIES)

Assembly Language
Programming Manual

771049

Price: $20.00

Microdata Corporation
17481 Red Hill Avenue, Irvine, California 92714
Post Office Box 19501, Irvine, California 92713
Telephone: 714/540-6730 · TWX: 910-595-1764

TABLE OF CONTENTS

LIST OF FIGURES

HOW TO USE THE REALITY® MANUALS

The Reality® manuals are written in modular format with each pair of facing pages presenting a single topic.

This and other Reality manuals differ substantially from the typical reference manual format. The left-hand page of each topic is devoted to text, while the right-hand page presents figures referred to by the text. At the head of each text page are a pair of titles, the first title naming the section, the second the topic. Immediately below these titles is a brief summary (boxed) of the material covered in the topic.

The advantage of this format will become readily apparent to the reader as he uses this manual. First, the figures referred to in the text are always conveniently right in front of the reader at the point where the reference is made. Secondly, the reader knows that when he turns the page, he has completed one idea and is ready to encounter a new one.

Documentation for the Reality system includes the following:

Reality Programmer's Reference Manual, #1048

Reality EDITOR Operator's Guide, #1052

Reality ENGLISH™ Programming Manual, #1038

Reality DATA/BASIC™ Programming Manual, #1051

Reality PROC and BATCH Programming Manual, #1044

Reality Assembly Language Programming Manual, #1049

Reality Bisync Operator's Guide, #1043

The examples throughout this manual use certain conventions as defined in Figure A.

CONVENTION	MEANING
TEXT	*Shaded text represents the user's input.*
TEXT	*Standard text represents computer output printed by the system.*
TEXT	*Italicized text is used for comments and notes which help explain or describe the example.*
(CR)	*This symbol represents a carriage return.*
ß	*This symbol represents a space (blank).*

Figure A. Conventions Used Throughout This Manual

SECTION 1

INTRODUCTION

1.1 THE REALITY® CPU AND HARDWARE

The Reality system runs on a Microdata 1600 CPU. Although small in size, it has the architecture of a medium scale computer. Its main memory is core, and is expandable to 131,072 bytes. The CPU cycle time is 200 nanoseconds, and the main memory full cycle time is 1 microsecond. The CPU is microprogrammed, meaning that the assembly language instructions are executed by many small micro-instructions which are "close" to the machine. These micro-instructions (firmware) are in read-only, fixed memory, as this affords higher execution speeds. Taking this approach permits proven hardware to be used for Reality systems while allowing the flexibility of a custom instruction set.

The mass memory is disc, which is organized into 512 byte blocks called frames. Over 300 megabytes of disc storage can be configured on a Reality system. There is a large list of supporting hardware which can be interfaced to the 1600 Computer, including tapes, communication devices, terminals, etc.

1.2 THE REALITY SYSTEM ARCHITECTURE

Figure 1-1 shows an overall view of the software on the Reality system. The firmware is burned onto integrated circuit chips and placed on a firmware board. The monitor serves to allocate disc activity and to schedule processes for activation. It uses assembly language instructions which are executed by the firmware. There is a large volume of available system software, also written in assembly language, with instructions executed by the firmware. This system software includes compilers, utilities, the assembler, and a large number of subroutines to which the user may interface.

Reality will support up to 32 separate asynchronous processes (terminals plus the work they are doing). Because Reality code is reentrant, each may be running the same or different tasks.

Reality assembly language operates through its own set of registers, stacks, accumulators, and other data structures. Each process is assigned a control block which contains 16 addressing registers, an accumulator, a return stack which will hold 11 entries, an accumulator extension, and a large number of other registers, counters, pointers, and flags which make the assembly language very powerful. The 16 address registers in a control block can access any byte in disc space. Relative addressing is also possible using an offset displacement plus one of the registers to any bit, byte, word (16 bits), double word (32 bytes), or triple word (48 bits) in the entire virtual memory.

Input and output to the terminals is handled automatically by the firmware, which makes these operations fast. Input and output to the discs are handled automatically by the monitor and firmware, a feature which greatly simplifies the programming task.

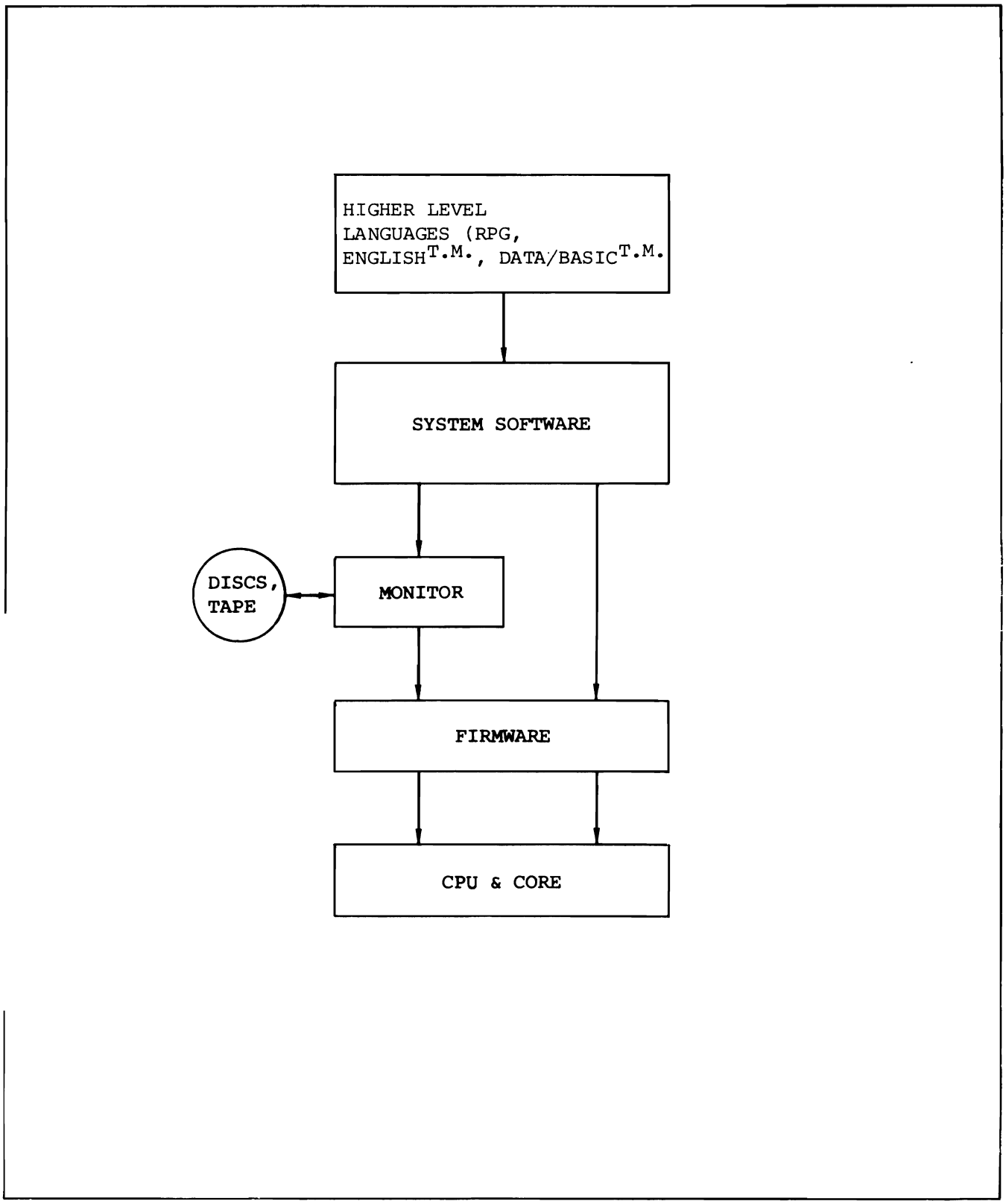


Figure 1-1. Hierarchy of System Architecture

1.3 THE REALITY INSTRUCTION SET

The Reality Computer System has an extensive instruction set. The main features include:

- Bit, byte, word, double-word, and triple-word operations.
- Memory-to-memory operation using relative addressing on bits, bytes, words, double-words, and triple-words.
- 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.
- 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.

1.4 RESTRICTIONS ON USE OF ASSEMBLY LANGUAGE ON REALITY

While the use of assembly language on Reality is supported, certain restrictions are placed on this usage to insure compatibility from one release to another, and to insure that the systems are both hardware and software supportable. The following are not supported.

1) Any change to the assembly code system software supplied by Microdata.

2) Any interface to routines not documented in the chapter SYSTEM SUBROUTINES in this manual.

3) Any interface to P.I.B's or other data or code in non-virtual core.

4) Any code written in monitor mode.

5) Any modifications, deletions or additions to the tables PSYM and OSYM as supplied on SYS-GEN tapes.

6) Any interface to a device or peripheral not supplied by Microdata.

1.5 MANUAL ORGANIZATION AND CONVENTIONS

This manual is organized as follows:

- Section 2 is essentially a "reference manual" for the Reality CPU. It describes the system structure and the machine instructions.
- Section 3 describes the Reality Assembly Language (REAL).
- Section 4 describes the Interactive Debugger (DEBUG), which may be used to monitor and control program execution.
- Section 5 documents Microdata-supplied system subroutines (and their interfaces) which users may call.

In presenting general command formats throughout this manual, the following conventions apply.

Convention	Meaning
UPPER CASE	Characters or words printed in upper case are required and must appear exactly as shown.
lower case	Characters or words printed in lower case are parameters to be supplied by the user (e.g., file name, item-ID, data, etc.).
{}	Braces surrounding a word and/or parameter indicate that the word and/or parameter is optional and may be included or omitted at the user's option.
{}...	If an ellipsis (i.e., three dots) follows the terminating bracket, then the enclosed word and/or parameter may be omitted or repeated an arbitrary number of times.

In presenting examples, the following conventions apply:

Convention	Meaning
TEXT	Shaded text represents the user's input.
TEXT	Standard text represents output printed by the system.
CR	This symbol represents a carriage return.
LF	This symbol represents a line feed.

SECTION 2

REALITY CPU REFERENCE INFORMATION

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 separate documents.

2.1 SYSTEM STRUCTURE

2.1.1 SYSTEM COMPONENTS

The Reality system consists of a core storage unit, a tape drive, a printer, a disc storage device used as a virtual storage unit, a central processing unit (CPU), and from one to 32 input/output terminals. There is a one-to-one correspondence between a terminal attached to the system and a process. Additional input/output devices such as magnetic tape units, disc units, card readers, and printers may be attached to the system. Input/output devices, other than a process's terminal, may be accessed by any process. The disc unit containing the virtual store, however, cannot be accessed as an input/output unit, except by the monitor.

2.1.2 INFORMATION FORMATS

The system CPU processes information 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 leftmost 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. Figure 2-1 shows the information formats.

2.2 VIRTUAL STORAGE

All information in the Reality system, other than the monitor program and certain data used by the monitor, is stored in virtual space. "Virtual" means that the physical location of this space moves from disc to core and from core to disc automatically, without attention from the user.

2.2.1 VIRTUAL STORAGE ORGANIZATION

Virtual storage is organized into blocks of 512 bytes each. Each block is called a frame. Frames are numbered from one to some maximum number which depends on the system configuration. When frames are needed for processing, they are moved into core for access by the CPU. Frames which are not used often are moved into disc storage to make room for other frames in core. This movement of frames is handled

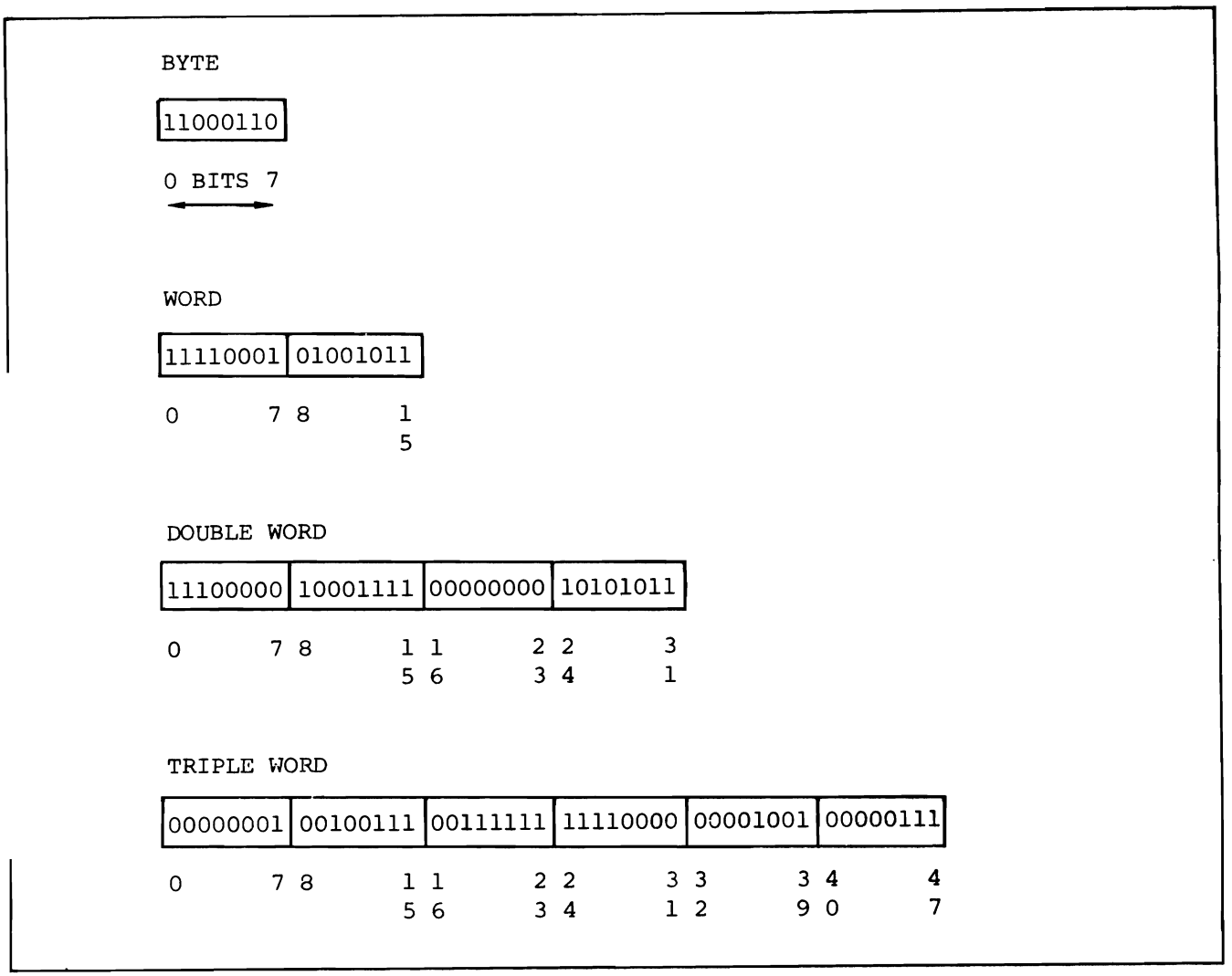


Figure 2-1. Information Formats

automatically by the Reality firmware and monitor, so as far as the user is concerned, available space can be viewed as a linear set of frames. Whether data is actually in core or on disc at any given time is of little importance, and is considered only when optimizing programs.

2.2.2 ADDRESSING VIRTUAL STORAGE

All program references to information are references to virtual storage. Fields in virtual storage are referenced via a frame number (frame-ID, or FID) 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 4096 frames.

2.3 CORE STORAGE

2.3.1 CORE STORAGE ORGANIZATION

Core storage is also organized into 512-byte blocks, called buffers. A few buffers are reserved for the monitor, tables, and status indicators which are required to operate the Reality system. The remaining buffers are available for storing data read in from disc; one buffer can hold exactly one frame of virtual storage.

Any core buffer can hold any frame, and at any given time the buffers in core will have a scattering of frames. Frames are read into core as buffers become available, without regard to which buffers they are. Frames are written back to disc as they fall into disuse.

2.3.2 ADDRESSING CORE STORAGE

Byte locations in core storage are consecutively numbered starting with zero. 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 particular Reality instruction. The addressing mechanism uses a one-bit bank select register and a sixteen-bit binary address register, giving a maximum of 131,072 addressable bytes.

2.4 VIRTUAL STORAGE MANAGEMENT

The Reality monitor uses several tables to manage the movement of frames between core and disc.

The Buffer Status Table contains the status of each buffer in core storage--whether it is I/O-busy, corelocked (not to be read into) or, write-required (data in frame changed).

The Buffer Map (or FID Table) is a table containing the virtual storage addresses of all frames currently in core buffers.

The Buffer Queue (or Links Table) is a linked list of buffer numbers used to schedule buffers for disc I/O efficiently.

The Hash Address Table (HAT) and the Hash Link Table (HLT) are used to locate frames in core storage. To determine whether a given frame is in core or not, the frame number is transformed ("hashed") into a HAT entry number, which points to a list of HLT entries. This list contains the numbers of all frames in core which have the same HAT entry number. If the given frame number is not in the list, the frame is not in core, and a frame fault is then generated in order to read the frame in from disc.

Figure 2-2 shows the interaction of the memory management tables.

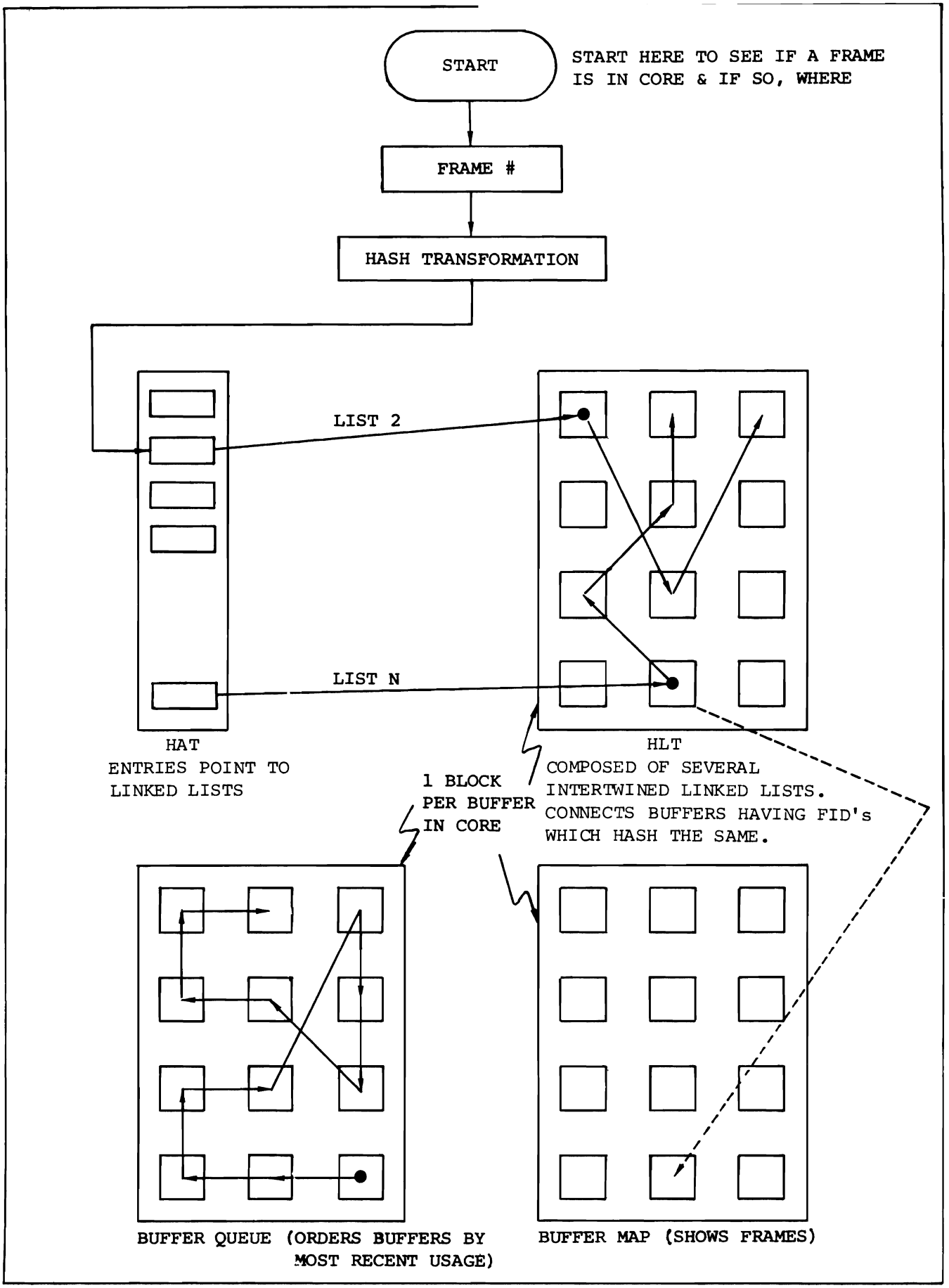


Figure 2-2. Memory Management Tables

2.4.1 FRAME FAULTS

If a program (process) attempts to reference data which is not in core, it is deactivated and marked disc-roadblocked by the firmware, and the monitor is activated to search for an available buffer for the frame. If an available buffer exists and the required disc drive is not busy, the monitor sets that buffer's status to I/O-busy and corelocked, zeroes the frame number in the Buffer Map, and commands the disc to read the requested frame into that buffer. Then the monitor selects another process for activation. When the disc interrupt occurs, indicating completion of the read, the monitor stores the requested frame number in the Buffer Map, clears the I/O-busy and corelocked buffer status, and clears the process's disc-roadblocked flag. The monitor then starts another disc read, if possible, for another process, and selects another process for activation.

If a process needs a frame read into core and no buffer is available, the monitor finds the least recently used buffer which is not corelocked, writes it out to disc if its write-required flag is set, and marks it available. By the time a buffer is written out, however, the disc drive required to read in the desired frame may be busy, or another process may have already read this frame into core. But the buffer freed by a write is always marked available in the Buffer Map (by zeroing the frame number), and this table is always searched before using the available buffer.

2.4.2 AUTOMATIC FRAME WRITES

Whenever the monitor completes a search for disc roadblocks for an available disc drive and fails to find any, it next looks for a buffer with write-required, non-corelocked status. It searches the Buffer Map beginning with the least recently used buffer and starts a write/verify operation for the first buffer found for an available disc drive. The monitor sets the buffer's corelocked flag and clears the write-required flag, but does not set the I/O-busy flag in this case. This means that processes may read and write data into and out of the buffer after the write/verify operation is initiated, but if the data in the buffer changes, the firmware will set the write-required flag again. When the write/verify operation is complete, the monitor clears the buffer's corelocked status and searches for another process requiring disc I/O. The monitor thus ensures system efficiency by continually providing available buffers.

2.5 PROCESSES

The Reality CPU is designed as an interactive system capable of communicating with several users simultaneously. A user communicates with the system via a terminal, and associated with each terminal is a process. A process can be defined as a continuing operation on a set of functional elements (areas of virtual space). The number of processes in a Reality system is a function of its configuration. Each process, except the monitor process, is associated with a Process Identification Block (PIB) in core, and a Primary Control Block (PCB) and other elements in virtual space.

2.5.1 PROCESS IDENTIFICATION BLOCK

Process Identification Blocks (PIBs) are used in handling the I/O operations associated with each process, and in scheduling activation and deactivation of the processes. Each PIB is 64 bytes long and is formatted as shown in Figure 2-3.

NOTE

The information in Figure 2-3 is intended to give a better understanding of the operation of Reality systems. It is not intended to be used as an interface specification.

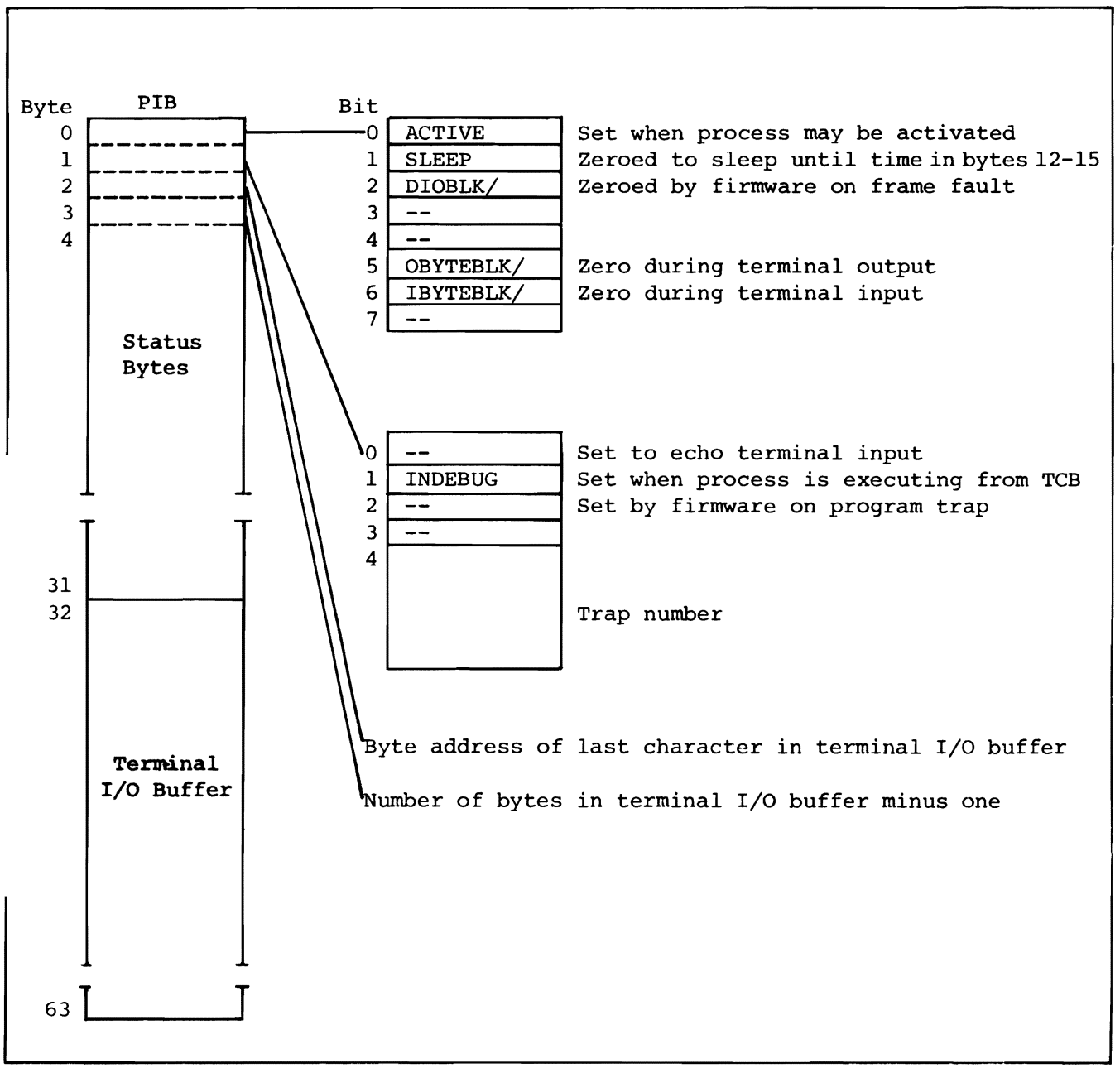


Figure 2-3. PIB Format

2.5.2 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. Figure 2-4 describes elements of the PCB which are accessed by the firmware. Figure 2-5 shows the entire PCB layout. Figure 2-6 shows the Secondary Control Block (SCB) layout.

BYTES	DESCRIPTION
0	Reserved.
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'182'-X'183'	These bytes contain the pointer to the current top of the subroutine stack.
X'184'-X'1AF'	These bytes contain the subroutine return stack. Each entry is four bytes long: the first two bytes contain the FID and the last two contain the displacement of the address one before that where program execution is to resume upon returning from the subroutine.

Figure 2-4. PCB Elements Accessed by Firmware

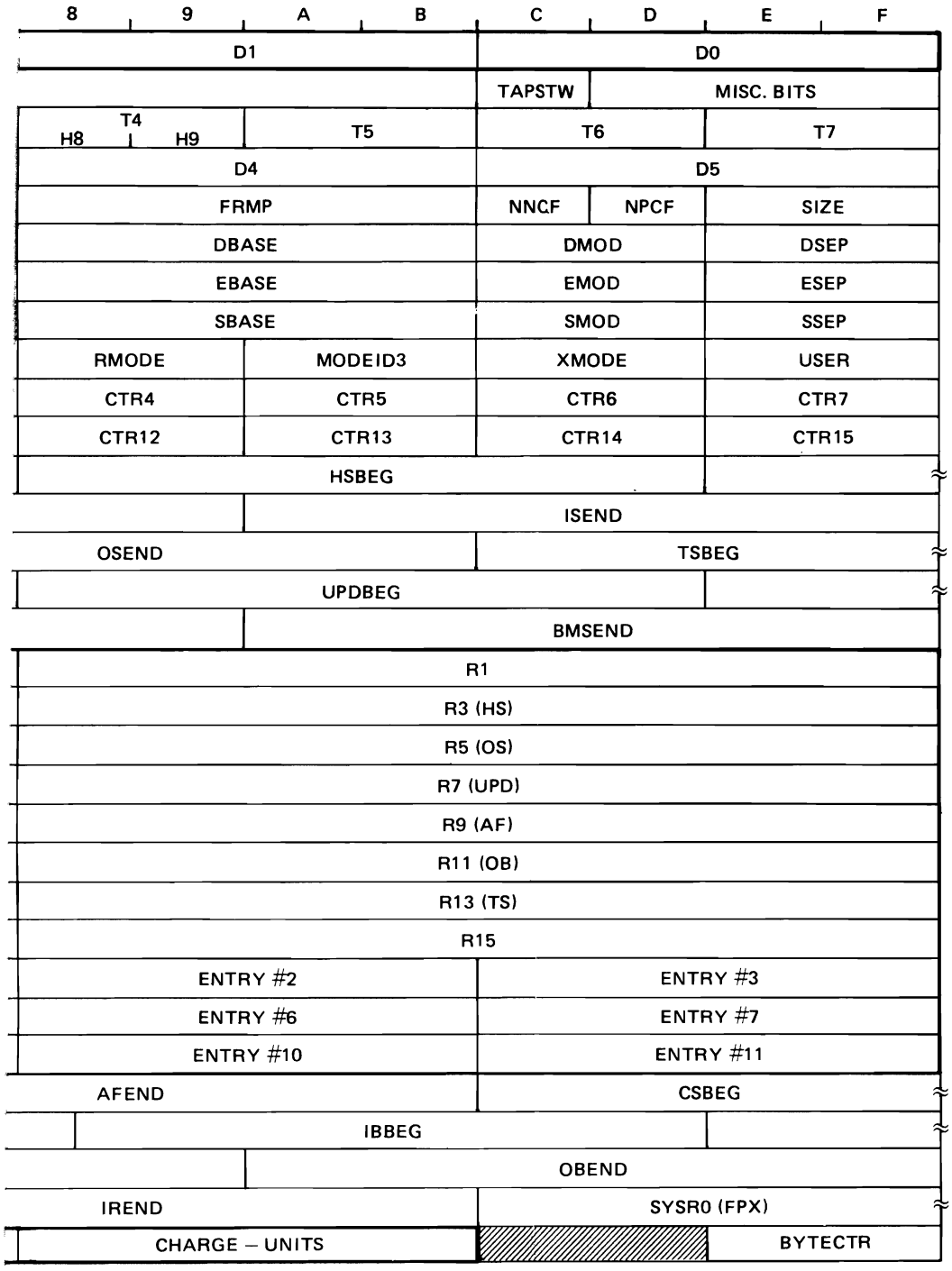
PRIMARY CONTROL BLOCK

Addressing register R0 set to PCB. Areas bordered by hea
reserved for future system software use.

	0	1	2	3	4	5	6	7
000		ACF	PRMPC	SC0	SC1	SC2	DEBUGBYTE	RNICTR
010	AFLG-ZFLG, SB0-SB35, DAF0-DAF9, MISC. BITS							
020	CH0	CH1	CH2	CH3	CH4	CH8	CH9	SCP
030	D2				D3			
040	RECORD				FRMN			
050	BASE				MODULO		SEPAR	
060	MBASE				MMOD		MSEP	
070	OVRFLW				CMODE			
080	INHIBITH	ACFSAV	RCDCTR		MODEID2		WMODE	
090	CTR0		CTR1		CTR2		CTR3	
0A0	CTR8		CTR9		CTR10		CTR11	
0B0	REJCTR		REJ0		IBSIZE		OBSIZE	
0C0	≈ HSEND				ISBEG			
0D0	OSBEG							
0E0	≈		TSEND					
0F0	≈ UPDEND				BMSBEG			
100	R0WA		R0DSP		R0FLGS	R0FID		
110	R2 (SCB)							
120	R4 (IS)							
130	R6 (IR)							
140	R8 (BMS)							
150	R10 (IB)							
160	R12 (CS)							
170	R14							
180		RSCWA			RTN STACK ENTRY #1 FID		DSP	
190	ENTRY #4				ENTRY #5			
1A0	ENTRY #8				ENTRY #9			
1B0	AFBEG							
1C0	≈		CSEND					
1D0	≈ IBEND				OBBEG			
1E0	IRBEG							
1F0	≈		SYSR1 (FPY)					

Figure 2-5. Primary Control Block

vy lines are accessed by hardware. Shaded areas are



SECONDARY CONTROL BLOCK

Addressing register R2 set to SCB. SCB = PCB+1.

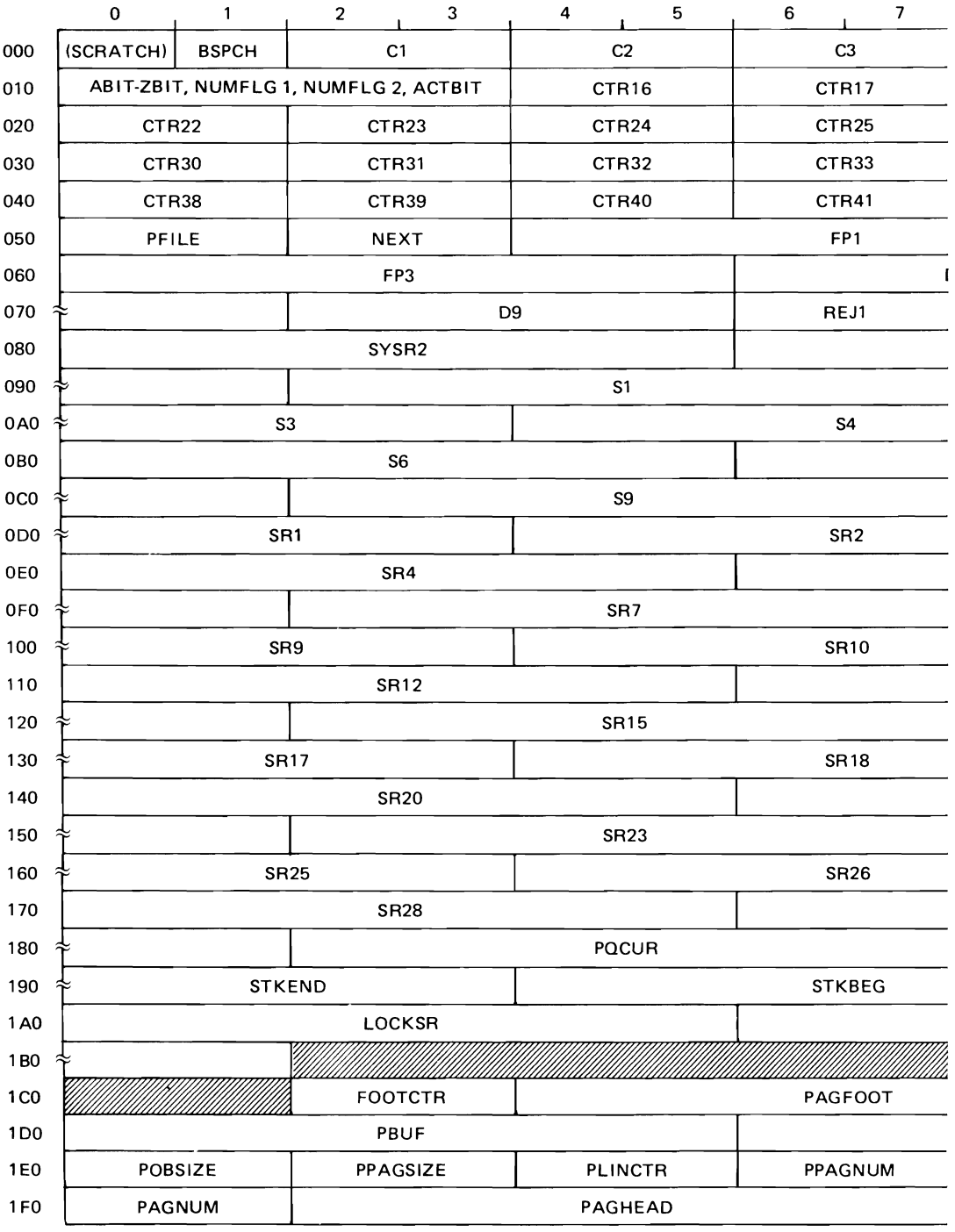
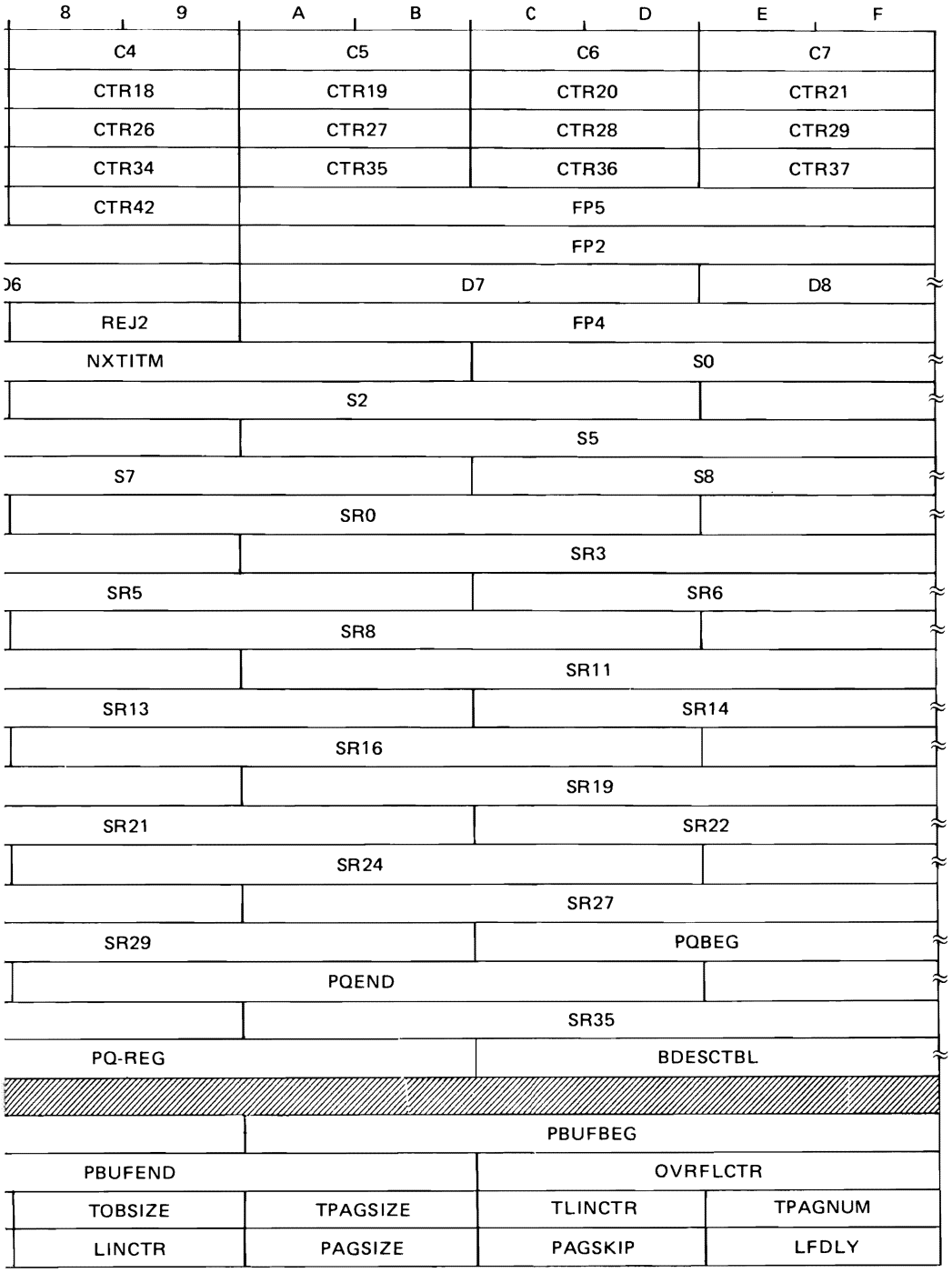


Figure 2-6. Secondary Control Block


8 9 A B C D E F
C4 C5 C6 C7
CTR18 CTR19 CTR20 CTR21
CTR26 CTR27 CTR28 CTR29
CTR34 CTR35 CTR36 CTR37
CTR42 FP5
FP2
D6 D7 D8
REJ2 FP4
NXTITM S0
S2
S5
S7 S8
SR0
SR3
SR5 SR6
SR8
SR11
SR13 SR14
SR16
SR19
SR21 SR22
SR24
SR27
SR29 PQBEG
PQEND
SR35
PQ-REG BDESCTBL
PBUFBEG
PBUFEND OVRFLCTR
TOBSIZE TPAGSIZE TLINCTR TPAGNUM
LINCTR PAGSIZE PAGSKIP LFDLY

2.6 FRAME FORMATS AND LINKAGES

2.6.1 FRAME FORMATS

The Reality system provides two types of frame formats: linked and unlinked.

Unlinked formats contain 512 data bytes (see Figure 2-7). For unlinked frames, the displacement portion of an address is relative to byte 0 of the frame. Displacements outside the range 0 through 511 are not valid for frames in the unlinked format.

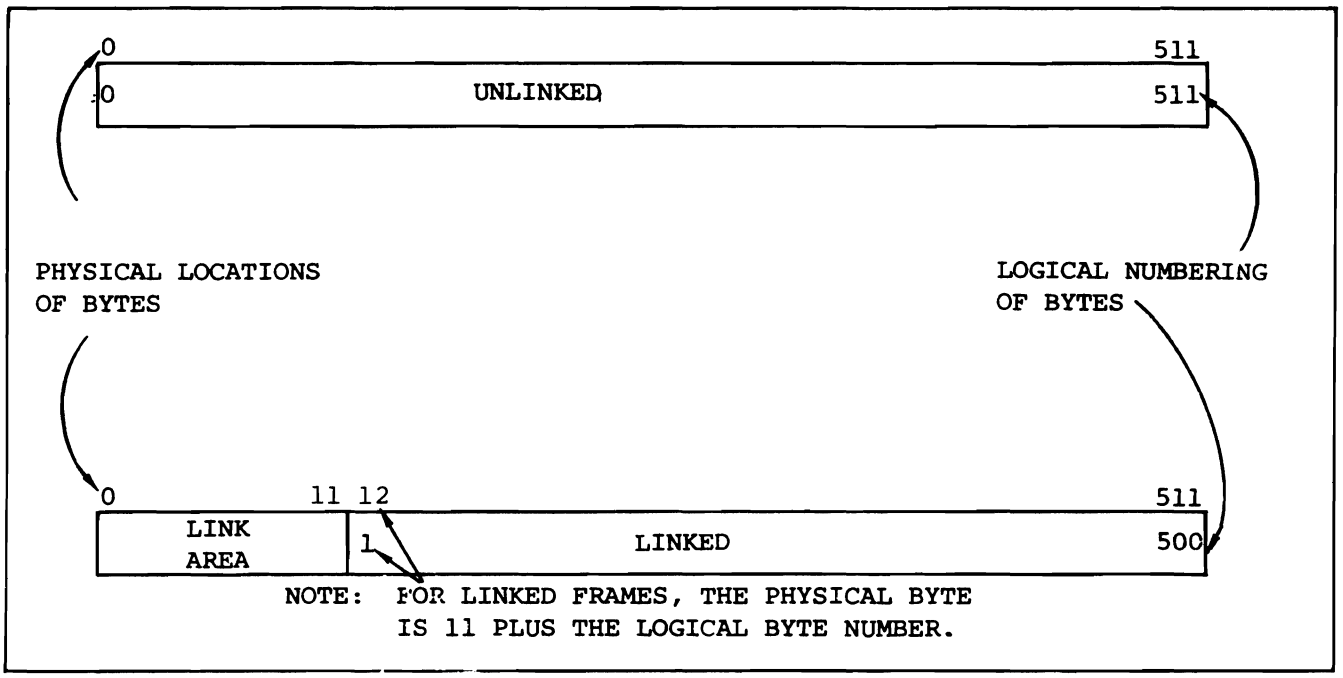
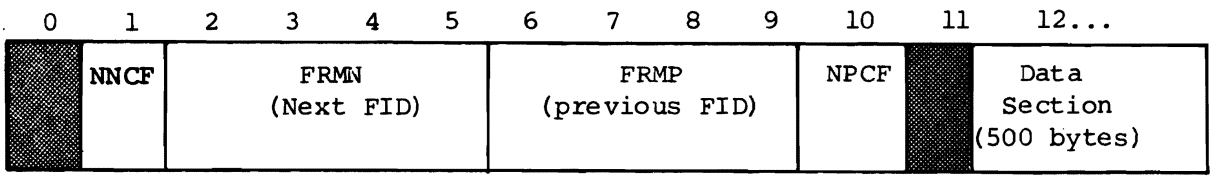


Figure 2-7. Unlinked Vs. Linked Frame Formats

Linked frames contain 500 data bytes, numbered 1 to 500. For linked frames, the displacement of an address is relative to byte 11 of the frame. However, a displacement of zero is a reference to byte 511 of the frame previous to the current frame. Displacements for linked frames may be positive or negative so long as the displacement references a logically linked frame. The link field is described in Figure 2-8.



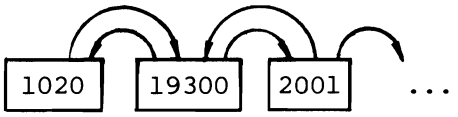
LINKED FRAME FORMAT

BYTES	NAME	DESCRIPTION
0	--	Unused and reserved.
1	NNCF	This byte contains a count of the number of next contiguous frames which follow this frame. A zero in this byte indicates that this frame is the last frame in a contiguously linked set of frames.
2-5	FRMN FORWARD LINK	This field contains the frame number of the frame that logically follows this frame. If byte 1 contains other than zero, this will be the next higher numbered frame. If byte 1 contains a zero this may be any frame number. A zero in this field indicates that this is the last frame of a linked set.
6-9	FRMP BACKWARD LINK	This field is similar to bytes 2 through 5 except that it contains the number of the frame preceding this frame.
10	NPCF	This byte is similar to byte 1 except that it contains a count of the number of previous contiguous frames preceding this frame.
11	--	Unused and reserved.
12-511		Data section.

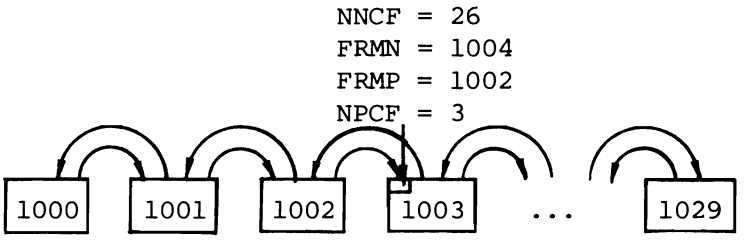
Figure 2-8. Linked Frame Format

2.6.2 LINKED SETS OF FRAMES

A series of frames may be linked together to hold data structures that will not fit in a single frame. Such a linked set may contain contiguous frames, singly linked frames, or combinations. Figure 2-9 shows some examples.



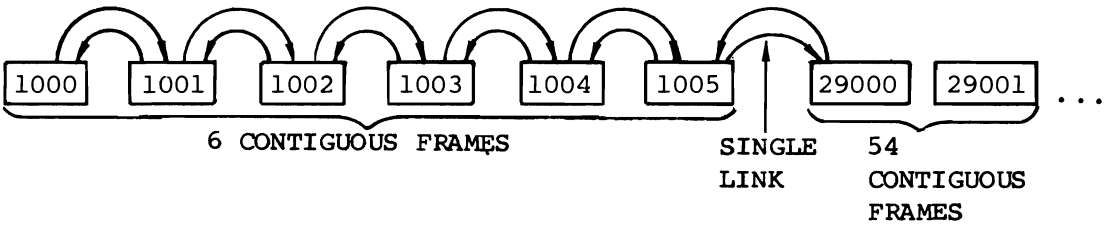
A. A SERIES OF SINGLY LINKED FRAMES



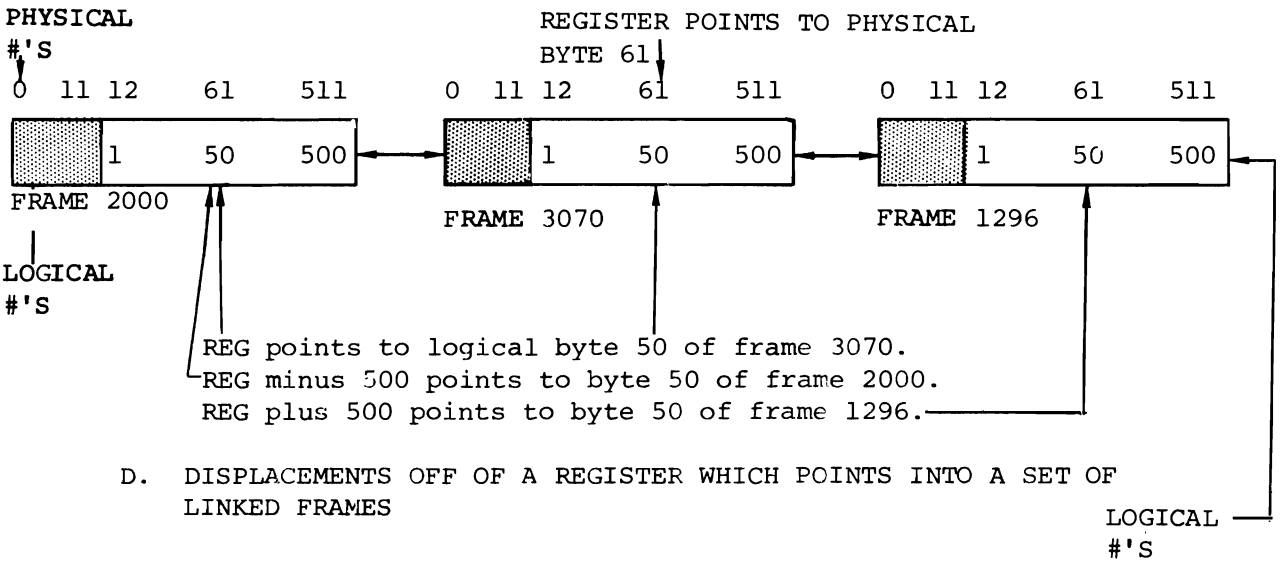
B. A SET OF 30 CONTINGUOUSLY LINKED FRAMES

NOTE

If all NNCF and NPCF fields in these frames were zero, this would be a singly linked list of frames which happened to have consecutive FID's.



C. TWO CONTIGUOUS LINKED SETS THEMSELVES LINKED WITH A SINGLE LINK. THIS IS TYPICAL OF 'LOGON WORKSPACE.'



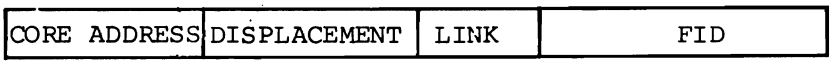
D. DISPLACEMENTS OFF OF A REGISTER WHICH POINTS INTO A SET OF LINKED FRAMES

Figure 2-9. Examples of Linked Sets of Frames

2.7 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 contain 8 bytes as described in Figure 2-10.

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 address register is "detached."
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 twos complement form. These bytes are meaningful only when the register is detached. (See the section ADDRESS REGISTER ATTACHMENT.)
4	Zero in bit zero of this byte indicates that the register references data in linked format. If bit zero is a one, the register references the data in 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. Bit 7 is used as an extension of the word address to indicate a main memory bank.
5-7	These bytes contain the virtual storage frame number of the byte being referenced.

Figure 2-10. Address Register Format

ADDRESS REGISTER ZERO

Register zero is used in a special way. This register always points to the PCB. Register zero is attached when the process is activated. The displacement field of this register is always effectively 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 byte zero of the program frame. This allows register one to be used to reference data in 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.

2.7.1 ADDRESS REGISTER ATTACHMENT

When setting up an address register, the first two bytes of the register must be set to zero. Bytes 2 through 7 are set to 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 procedes as follows. The buffer map is searched 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 road-blocking 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.

Figure 2-11 summarizes the attachment/detachment process.

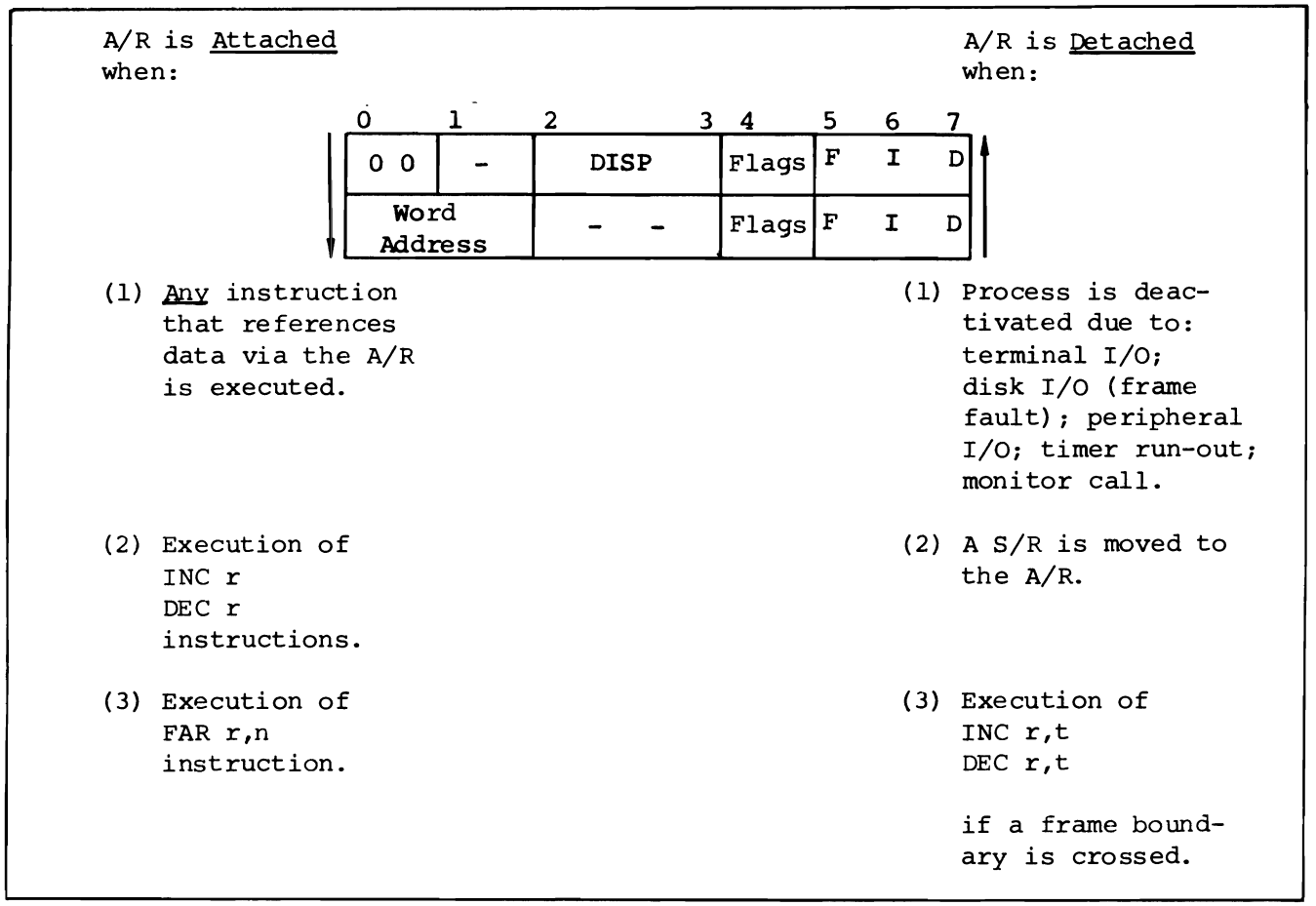


Figure 2-11. Attachment & Detachment of Address Registers

2.7.2 CAUTIONS INVOLVING REGISTER ATTACHMENT

Address registers can be set up explicitly by setting their fields appropriately; a more conventional way is to move a S/R into it. Consider the following:

```
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.

2.7.3 ATTACHMENT AND DETACHMENT OF ADDRESS REGISTERS

All instructions that reference data force "attachment" of the A/R(s) used in the reference. Other instructions do not do this; 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 continguously linked (see Section 3.14). Therefore, it is best to force "attachement" 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.

2.8 THE MONITOR

The monitor is a program that is an integral part of the Reality system. Its function is to initiate the transmission of information between core storage and virtual storage and to schedule each of the processes.

2.8.1 MONITOR FUNCTIONAL ELEMENTS

The monitor process is the only one not associated with a PIB. The PCB for the monitor is located in low core.

Besides the functional elements described in Section 2.5.2, the monitor PCB contains such information as the system time and date, pointers for peripheral devices zero through fifteen, and the bootstrap software.

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.

2.8.2 PROCESS SCHEDULING

The monitor maintains a queue of processes currently in the system, arranged in increasing order of expected total processing time. The position of a process in the queue determines its priority for activation--the process at the head of the queue has the highest priority. The process with the highest priority without any roadblocks is always the next one to be activated.

Expected total processing time for a process, at any given instant, is based on the amount of CPU processing and number of disc reads already done by that process. Interactive processes are favored by increasing their priority. As a process outputs characters to the terminal, it migrates up the priority queue. When a process receives terminal input, it is moved to the head of the queue, for immediate activation. As a process consumes system resources (CPU time and disc reads), it migrates down the priority queue.

The effect of each system resource on a process's priority is controlled by a weighting factor. When the number of units of a resource consumed reaches the weighting factor, the process is moved up or down in the priority queue one position and the resource unit count is reset to zero. See 'priority scheduling' in The Programmer's Reference Manual.

One process in the Reality system may be designated the Super High Priority Process (SHPP) in order to receive special handling in the process scheduling mechanism. The SHPP has top priority to all system resources, allowing it to run without interference from other process. This is implemented for BISYNC communictions. The following rules are applied to the SHPP:

- The SHPP is reactivated when its CPU processing time quantum is used up.
- Frame faults for the SHPP are processed as soon as the necessary disc is available.

- Disc interrupts which signal completion of a disc read for the SHPP cause the SHPP to be activated immediately.

- Interrupts from devices with addresses in the range X'10'-X'13' which are for the SHPP cause the SHPP to be activated immediately.

- Voluntary RQM's by the SHPP allow two other processes to run before the SHPP is activated again.

2.9 PERIPHERAL I/O

Communication between the CPU and peripheral devices is made through controllers. Each controller has a unique device address in the Reality system. Device addresses 0 through 15 are used for non-virtual storage devices such as tape drives and line printers, addresses 16 through 23 are used for virtual storage devices (disc drives), and addresses 24 through 27 are used for the process terminals.

I/O instructions other than those for terminal I/O must specify a device address and an order code. The meaning of each of the eight possible order codes is explained in Figure 2-12. External interrupts cause the monitor to perform certain processing; during this time, further external interrupts are inhibited.

2.10 PROGRAM TRAPS

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:

Message	Description
ILLEGAL OPCODE	An illegal (undefined) operation code has been executed.
RTN STACK EMPTY	A RTN (return) instruction was executed when the return-stack was empty (RSCWA equals X'0184').
RTN STACK FULL	A BSL or BSLI (subroutine call) instruction was executed when the return-stack was full (RSCWS equals X'01B0'); the return-stack has been reset to an "empty" condition before the trap.
REFERENCING FRAME ZERO	An address register has a FID of zero.
CROSSING FRAME 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.

Message	Description
FORWARD LINK ZERO	An address register in the "linked" format has benn incremented past the last frame in the linked frame set.
BACKWARD LINK ZERO	An address register in the "linked" format has been decremented prior to the first frame in the linked frame set.
PRIVILEGED OPCODE	A Privileged operation code (one executable only in the Monitor mode of operation), has been found while executing in the Virtual mode.
REFERENCING ILLEGAL FRAME	An address register has a FID outside the allowable disc configuration.
RTN STACK FORMAT ERR	The Return-stack pointer is illegal either less than X'0184', or greater than X'01B0'. The return-stack has been reset to an "empty" condition.
DIVIDE OVERFLOW	An overflow condition occurred on a divide operation.
REFERENCING ILLEGAL DEVICE	A device has been referenced outside the allowable system configuration.
UNNORMALIZED	A storage register with an unnormalized displacement was referenced.

*A register number will be printed out.

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 stopped and an 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.

Figure 2-12. Order Codes

SECTION 3

REALITY ASSEMBLY LANGUAGE (REAL)

The Reality Assembler Language (REAL) translates source statements into Reality CPU machine language equivalents. The source program, or "mode", is an item in any file defined on the data base. 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).

3.1 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 opcode) field, an operand field, and a comment field.

3.1.1 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.

3.1.2 OPERATION FIELD

The operation field begins following the label field and consists of a legal REAL opcode. Opcodes are pre-defined in the permanent opcode symbol file OSYM and consist of one or more alpha characters. Opcodes may be mnemonics for Reality machine language instructions (e.g., B for BRANCH); macros, which may assemble into several Reality machine language instructions (e.g., MBD for MOVE BINARY to DECIMAL); or assembler pseudo-ops (e.g., ORG for ORIGIN).

3.1.3 OPERAND FIELD

Operand field entries are optional, and vary in number according to the needs of the associated REAL opcode. 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.

3.1.4 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

```
001                             FRAME 501
    001 7FF001F5                +FRM: 501
    001                         +ORG  1
002                     *TEST
003                     *11MAR77
004                     *03
005                     *00
006                     *SMITH
007                             B     !DISPLAY
    001 1C3F                    +B:   !DISPLAY
008                             B     !COMPARE
    003 1CD8                    +B:   !COMPARE
009                     *
010     M 2 2C          BACK    DEFM  2,PROC-I
011     T AC 2          SR24DSP DEFT  2,SR24
012     D AD 2          SR24FID DEFD  2,X'AD'
013                     *
014                     *THIS SUBROUTINE DISPLAYS ALL FOUR PROC BUFFERS, AND
015                     *INDICATES CURRENT BUFFER POINTER POSITIONS
016                     *
017 005 E2E5EC          INIT    MOV   PBUFBEG,R12     P.I.B FROM PBUFBEG TO SM, FOLLOWED BY S.I.B.
018 008 6C08C0                  SCD   R12,X'C0'       SCAN TO SM AT END OF P.I.B.
019 00B E2E5CC                  LAD   R12,PBUFBEG
020 00E A01459                  STORE T4              STORE THE LENGTH OF PRIMARY INPUT BUFFER.
021 011 E2E5CA                  LAD   PBUFBEG,IB      IB IS CURRENT POINTER IN ACTIVE INPUT BUFFER
022 014 A05059                  STORE CTR3            INIT SIZE OF P.I.B.
023 017 A05159                  STORE CTR9            INIT SIZE OF S.I.B.
024                             BLE   T0,T4,S0500     BRIF IB IS IN P.I.B.
    01A F0075014                +BT:  T0,T4,S0500,3
    01E 5C0C
025 020 F0517014                DEC   CTR9,T4
026 024 A05142                  DEC   CTR9
027 027 A05040                  ZERO  CTR8            INDICATE THAT POINTER IS NOT IN P.I.B.
028                             B     S0800
    02A 1C03                    +B:   S0800
029 02C A05140          S0500   ZERO  CTR9            INDICATE NO POINTER IN S.I.B.
030                     *
031 02F 90920802        S0800   BBZ   SFLG,S1000      IF STACK ACTIVE, EXCHANGE IS, UPD.
032                             CMNT  *               IS AND UPD WILL BE EXCHANGED AGAIN LATER
033 033 1747                    XRR   IS,UPD
034 035 E062C4          S1000   LAD   ISBEG,IS        CALCULATE POSITION OF POINTER INTO P.O.B.
035 038 A05259                  STORE CTR10
036 03B E2CAC7                  LAD   STKBEG,UPD      DITTO FOR 'P.O.B.
```

FRAME STATEMENT

SOURCE LINE NUMBER

LABEL FIELD

OPCODE FIELD

LOCATION COUNTER

COMMENT FIELD

OBJECT CODE GENERATED

OPERAND FIELDS

MACRO EXPANSION OF ABOVE LINE

COMMENT LINE

Figure 3-1. Sample Assembly Listing

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.

The "N" option may be used with the MLOAD verb to load code into the TS workspace but suppress the normal copy from there to the specified frame. This may be helpful in checking the size or checksum.

3.5 VERIFYING A LOADED PROGRAM MODE

After assembling and loading a program, the TCL-II verb MVERIFY is used to check the assembled program against the loaded program.

Examples:

```
:[illegible]

 [217] MODE 'EXAMPL1' VERIFIED FRAME = 34 SIZE = 477 CHECKSUM = C3A2

:[illegible]

 014 OC 18

 [218] MODE 'EXAMPL2'  FRAME = 35  HAS 1 MIS-MATCHES
```

The first example verifies, but the second does not. In example two, the system informs the user that one byte at byte address 14 should have a value of OC, not 18.

An "A" option is available, and will cause a columnar listing of all bytes which mismatch. Each value in the source program which mismatches will be listed, followed by the value in the executable frame.

Example:

```
:[illegible]
LOC  SB  AB   LOC  SB  AB   LOC  SB  AB   LOC  SB  AB
014  0C  18   015  13  17   016  0E  0D   017  3A  3C
--   --        --             --             --

[218] MODE 'EXAMPLE3' PRAME = 35 HAS 78 MIS-MATCHES
```

The "E" option, useful when verifying several programs (items) with the same MVERIFY command, causes a message to be printed only if a program does not verify, and suppresses output otherwise.

```
:[illegible]
```

The "P" option causes all messages from MVERIFY to be routed to the line printer (spooler).

3.6 TCL-II CROSS REFERENCE CAPABILITY

3.6.1 CROSS-INDEX VERB

The TCL-II CROSS-INDEX verb first clears the CSYM file then updates it item by item with the external references of each item. Each attribute in the CSYM item records references of a particular type, such as bit, character, half-tally, etc. The CROSS-INDEX verb requires the following format:

CROSS-INDEX file-name item-list {(options)}

Example:

:[redacted]

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 is shown in Figure 3-2. Notice that the item called SYSTEM-SUBS-1 has one external reference to B14, two external references to BKBIT, etc.

3.6.2 X-REF VERB

The TCL-II X-REF verb uses the CSYM file as updated by the CROSS-INDEX verb for input. X-REF then updates the XSYM file in the opposite order of the CSYM file. The X-REF verb requires the following format:

X-REF file-name item-list {(options)}

Example:

:[redacted]

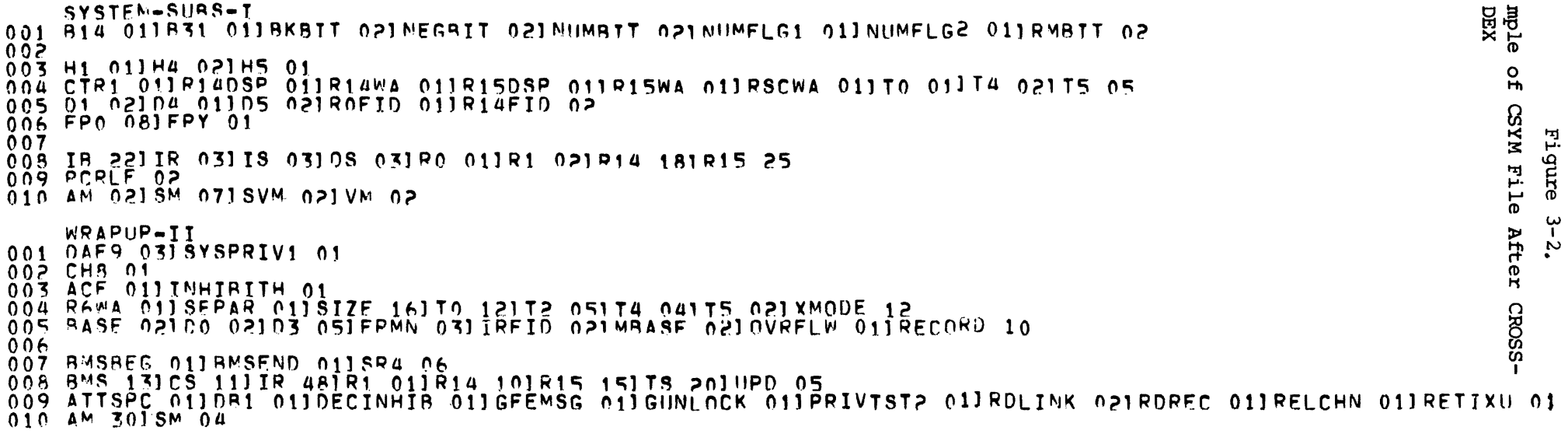
Would cross reference all items of the CSYM file. An example of what a portion of the XSYM file might look like after using the X-REF verb is shown in Figure 3-3. Notice that the item called T5 was externally referenced by WP3, WRAPUP-II, etc.

The SORT verb may be used after performing X-REF to produce a sorted output.

Example:

:[redacted]

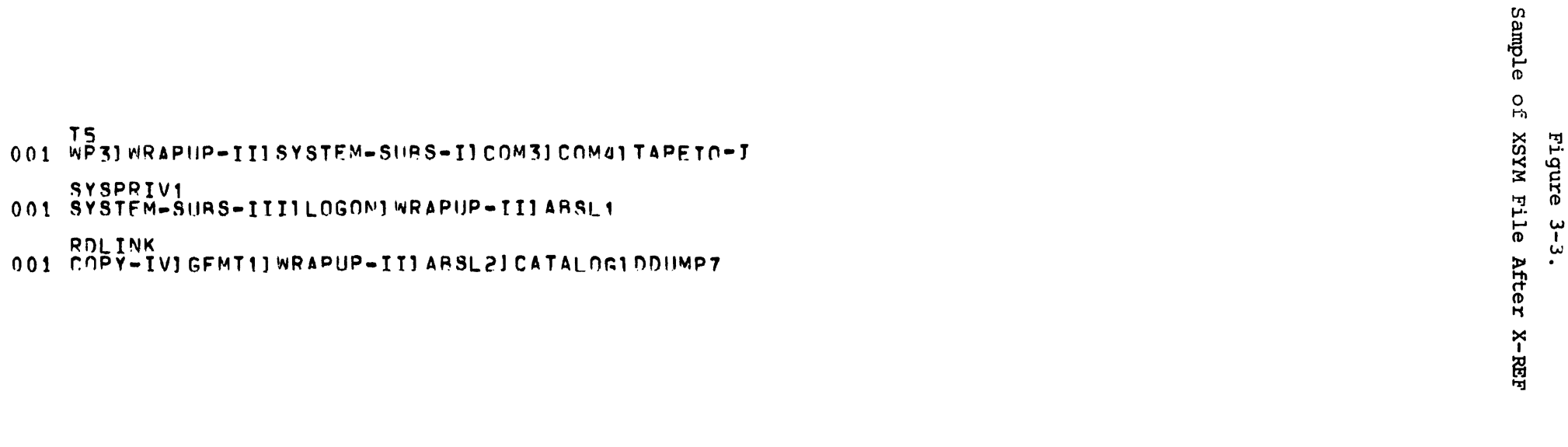
Would produce an alphabetical non-columnar listing on the line printer. REFERENCES and NONCOL are attribute definitions in the XSYM dictionary.



```
    SYSTEM-SUBS-I
001 B14 01]B31 01]BKBIT 02]NEGBIT 02]NUMBIT 02]NUMFLG1 01]NUMFLG2 01]RMBIT 02
002
003 H1 01]H4 02]H5 01
004 CTR1 01]R14DSP 01]R14WA 01]R15DSP 01]R15WA 01]RSCWA 01]T0 01]T4 02]T5 05
005 01 02]D4 01]D5 02]R0FID 01]R14FID 02
006 FP0 08]FPY 01
007
008 IB 22]IR 03]IS 03]OS 03]R0 01]R1 02]R14 18]R15 25
009 PCRLF 02
010 AM 02]SM 07]SVM 02]VM 02

    WRAPUP-II
001 0AF9 03]SYSPRIV1 01
002 CH8 01
003 ACF 01]INHIBITH 01
004 R6WA 01]SEPAR 01]SIZE 16]T0 12]T2 05]T4 04]T5 02]XMODE 12
005 BASE 02]C0 02]D3 05]FPMN 03]IRFID 02]MBASE 02]OVRFLW 01]RECORD 10
006
007 BMSREG 01]BMSEND 01]SR4 06
008 BMS 13]CS 11]IR 48]R1 01]R14 10]R15 15]TS 20]UPD 05
009 ATTSPC 01]DR1 01]DECINHIB 01]GFEMSG 01]GUNLOCK 01]PRIVTST2 01]RDLINK 02]RDREC 01]RELCHN 01]RETIXU 01
010 AM 30]SM 04
```

Figure 3-2. Sample of CSYM File After CROSS-DEX



```
    T5
001 WP3]WRAPUP-II]SYSTEM-SUBS-I]COM3]COM4]TAPETO-I

    SYSPRIV1
001 SYSTEM-SUBS-III]LOGON]WRAPUP-II]ABSL1

    RDLINK
001 COPY-IV]GFMT1]WRAPUP-II]ABSL2]CATALOG]DDUMP7
```

Figure 3-3. Sample of XSYM File After X-REF

3.6.3 XREF PROC

The XREF Proc will perform the following functions:

1. Clear the XSYM file.
2. Use the X-REF 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:

:[redacted]

would cross reference all items of the CSYM file and would list the results in alphabetical order on the line printer.

3.7 THE REAL INSTRUCTION REPERTOIRE

In defining the REAL opcodes, the following set of symbolic operands are used.

Symbol	Operand	Description
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.

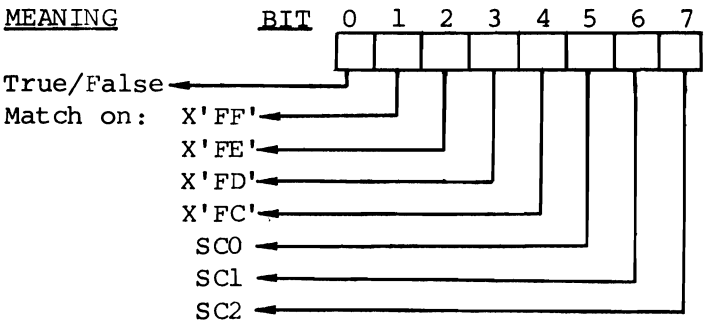
n	LITERAL	A literal or immediate value. The size of the assembled literal or value is dependent on the instruction in which the "n" is used.
r	ADDRESS-REGISTER	One of the sixteen Reality address registers (A/R's).
s	STORAGE-REGISTER	A 6-byte field (usually a storage-register, or S/R) relatively addressed 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.

In the following subsections, the first number in the comment field of each instruction is the length in bytes of that instruction. The parenthesized footnotes are defined in Section 3.7.12.

3.7.1 CHARACTER INSTRUCTIONS (MOVES)

MCC	n,c	6	(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	3		
	n,s	6	(1)	
	c,c	4		
	c,r	3		
	c,s	6	(1)	
	r,c	3		
	r,r	2		
	r,s	5	(1)	
	s,c	8	(2)	
	s,r	5	(3)	
	s,s	8	(2)	
MCI	n,r	3		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	9	(1)	
	c,r	4		
	c,s	10	(1)	
	r,r	2		
	r,s	8		
	s,r	5	(3)	
	s,s	11	(2)	

MCI	n,r,n	10	(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	13	(4)	
	n,r,t	13	(4)	
	n,r,d	13	(4)	
MIC	r,c	4		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	2		
	r,s	5	(1)	
	s,c	11	(2)	
	s,r	8	(3)	
	s,s	11	(2)	
MII	r,r	2		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	8	(1)	
	s,r	8	(3)	
	s,s	14	(2)	
MII	r,r,n	5	(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	(5)	
	r,r,t	5	(5)	
	r,r,d	5	(5)	
MII	r,r,r	4	(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	5	(3)	
MIID	r,r,n	3		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 0 is a true/false flag; if 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'. None of the scan characters may contain a system delimiter.

SCD r,n 3

Scan characters to delimiter(s). The addressing-register is incremented until a "match" condition (see MIID instruction) as defined by the 8-bit mask field "n" is found.

MIIT r,r 2

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 opcode). 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 2

This instruction assumes that address register R15 is set up to a location equal to or greater than that of addressing-register-1. (See MII opcode). 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	8	(2)	Exchange Character with Character; the byte
	c,r	5	(3)	(addressed) by operand-1 is interchanged
	c,s	8	(2)	with the byte defined by operand-2.
	r,c	5	(1)	
	r,r	2		
	r,s	5	(1)	
	s,c	8	(2)	
	s,r	5	(3)	
	s,s	8	(2)	
OR	c,n	6	(3)	OR character; the byte (character) addressed
	r,n	3		by operand-1 is logically or'd with the
	s,n	6	(3)	8-bit immediate operand-2.
XOR	c,n	6	(3)	Exclusive OR character; the byte (character)
	r,n	3		addressed by operand-1 is exclusively or'd
	s,n	6	(3)	with the 8-bit immediate operand-2.
AND	c,n	6	(3)	AND character; the byte (character)
	r,n	3		addressed by operand-1 is logically and'd
	s,n	6	(3)	with the 8-bit immediate operand-2.

3.7.2 CHARACTER INSTRUCTIONS (TESTS)

BCE	n,c,l	7	(1)	Branch Character Equal; the byte (character)
	n,r,l	4		defined or addressed by operand-1 is com-
	c,n,l	7	(3)	pared to the byte defined or addressed by
	c,c,l	6		operand-2. If the two bytes are equal,
	c,r,l	4		instruction execution branches to the loca-
	r,n,l	4		tion as defined by operand-3. Neither
	r,c,l	4		operand-1 nor operand-2 are altered. The
	r,r,l	3		arithmetic condition flag (ACF) is set on c,c,l 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,l	5		Branch if Character is Numeric; branch if the character addressed by the first operand is in the range 0-9, inclusive.
BCX	r,l	5		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,l	5		Branch if Character is Alphabetic; branch if the character addressed by the first operand is in the range A-Z, inclusive.

3.7.3 BIT INSTRUCTIONS

SB	b	2		Set Bit; the bit addressed by the operand is set to an on condition (one).
ZB	b	2		Zero Bit; the bit addressed by the operand is set to an off condition (zero).
BBS	b,l	4		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,l	4		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.

3.7.4 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	(6)	MOVe word to word; integer defined or addressed by integer-1 is moved to the location addressed by operand-2.
	n,t	4		
	n,d	4		
	h,h	4		
	h,t	6	(6)	
	h,d	6	(6)	
	t,h	6	(6)	
	t,t	4		
	t,d	6	(6)	
	d,h	6	(6)	

	d,t	6	(6)	
	d,d	4		
	b,b	4		
TST	h	3		Test the contents of the operand and set
	t	3		the arithmetic condition flags.
	d	3		
INC	h	3		INCrement by one; the integer defined by
	t	3		the operand is incremented by one.
	d	3		
	h,n	9	(6)	INCrement word by word; the integer de-
	h,h	4		fined or addressed by operand-2 is added
	h,t	9	(6)	to the integer stored in the location
	h,d	9	(6)	addressed by operand-1 and the result is
	t,n	4		stored in the latter location.
	t,t	4	(6)	
	t,d	7	(6)	
	d,n	4		
	d,h	7	(6)	
	d,t	7	(6)	
	d,d	4		
DEC	h	3		DECrement by one; the integer defined by
	t	3		the operand is decremented by one.
	d	3		
	h,n	9	(6)	DECrement word by word; the integer
	h,h	4		defined or addressed by operand-2 is
	h,t	9	(6)	subtracted from the integer stored in
	h,d	9	(6)	the location addressed by operand-1 and
	t,n	4		the result is stored in the latter loca-
	t,h	7	(6)	tion.
	t,t	4		
	t,d	7	(6)	
	d,n	4		
	d,h	7	(6)	
	d,t	7	(6)	
	d,d	4		
ZERO	h	3		ZERO word; a zero is moved to the operand
	t	3		location defined by operand-1.
	d	3		
ONE	h	3		Set word ONE; an integer value of one is
	t	3		moved to the operand location defined by
	d	3		operand-1.
NEG	h	3		NEGate word; the integer defined by
	t	3		operand-1 is negated (two's complement).
	d	3		
LOAD	n	3		LOAD to accumulator; the integer addressed
	h	3		by operand-1 is loaded into the 32-bit
	t	3		accumulator (D0). For half-word and word
	d	3		operands, the sign bit is extended.

LOADX	n	3		LOAD to accumulator; the integer addressed by operand-1 is loaded into the 48-bit accumulator (FP0), and the sign bit is extended.
	h	3		
	t	3		
	d	3		
STORE	h	3		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.
	t	3		
	d	3		
ADD	n	3		ADD to accumulator; the integer addressed by operand-1 is added to the 32-bit accumulator (D0) with sign extension.
	h	3		
	t	3		
	d	3		
SUB	n	3		SUB from accumulator; the integer addressed by operand-1 is subtracted from the 32-bit accumulator (D0) with sign extension.
	h	3		
	t	3		
	d	3		
MUL	n	3		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.
	h	3		
	t	3		
	d	3		
DIV	n	3		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).
	h	3		
	t	3		
	d	3		

3.7.5 REGISTER INSTRUCTIONS

MOV	r,r	2		MOVe register to register; the address or storage register operand-1 is moved into the address or storage register operand-2.
	r,s	3		
	s,r	3		
	s,s	4		
XRR	r,r	2		eXchange Register with Register; the address or storage register operand-1 is exchanged with the address or storage register operand-2.
	r,s	8	(1)	
	s,r	8	(2)	
	s,s	10	(1)	
INC	r	1		INCrement register; the address or storage register operand-1 is incremented by one.
	s	3		
INC	r,n	3		INCrement register by count; the address or storage register operand-1 is incremented by the integer stored at the location addressed by operand-2.
	r,h	6		
	r,t	3		
	r,d	6		
	s,n	4		

	s,h	7		
	s,t	4		
	s,d	7		
DEC	r	1		DECrement register; the address or storage register operand-1 is decremented by <u>one</u>.
	s	3		
DEC	r,n	3		DECrement register by count; the address or storage register operand-1 is decremented by the integer stored at the location addressed by operand-2.
	r,h	6		
	t,t	3		
	r,d	6		
	s,n	4		
	s,h	7		
	s,t	4		
	s,d	7		
LAD	r,r	6	(7)	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.
	r,s	3		
	s,r	3		
	s,s	6	(1)	
SRA	r,c	3		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.
	r,h	3		
	r,t	3		
	r,d	3		
	r,s	3		
	r,l	3		
FAR	r,n	3		Flag and Attach Register; the address-register operand-1 is attached. Normally n=0. If n=4 (or any value with bit 5 set), R15 is set to the first byte (unlinked format) of the frame.
BE	r,r,l	7	(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. NOTE: 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; therefore the instruction execution may prove incorrect if one of the registers is in an unlinked format, and the other is not. An abort will occur displacement if a linked format SR is greater than 500. (This can be remedied by moving it to a register, forcing attachment, and moving it back.) This is unnecessary if the SR and register point into the same contiguous block.
BU	r,s,l	4		
	s,r,l	4		

BE BU	s,s,l	6		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. If the displacement fields are not normalized, this may fail. See the forms above.

3.7.6 DATA COMPARISON INSTRUCTIONS

BE	n,h,l	7	(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.
	n,t,l	6		
	n,d,l	6		
	h,n,l	9	(6)	
	h,h,l	6		
	h,t,l	9	(6)	
	h,d,l	9	(6)	
	t,n,l	6		
	t,h,l	9	(6)	
	t,t,l	6		
	t,d,l	8	(6)	
	d,n,l	6		
	d,h,l	9	(6)	
	d,t,l	9	(6)	
	d,d,l	6		
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.
BE3	d,d,l	6		These forms of the compare instructions compare 3 byte fields starting 1 byte after each register. These allow register FIDS to be compared without including the flag byte in the compare.
BU3	d,d,l	6		
BL3	d,d,l	6		
BLE3	d,d,l	6		
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,l	6		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.
	t,n,l	6		
	t,t,l	6		
	d,n,l	6		
	d,d,l	6		

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,l	6	Branch on Decrementing word Zero; same as
	d,l	6	BDZ above but decrement by one.
BDNZ	t,l	6	Branch on Decrementing word not Zero; same
	d,l	6	as BDNZ above but decrement by one.
BDLZ	t,l	6	Branch on Decrementing word Less than Zero;
	d,l	6	same as BDLZ above but decrement by one.
BDLEZ	t,l	6	Branch on Decrementing word Less than or
	d,l	6	Equal to Zero; same as BDLEZ above but decrement by one.

All of the above data comparison instructions set the arithmetic condition flags.

3.7.7 TRANSLATE INSTRUCTIONS

MBD	h,r	10	Move Binary word to Decimal characters;
	t,r	10	This macro generates a call to the sub-
	d,r	10	routine MBDSUB (if "n" is not specified) or
	n,h,r	14	MBDNSUB (if "n" is specified), which con-
	n,t,r	14	verts from a binary integer at the location
	n,d,r	14	addressed by operand-1 to a string of deci- mal ASCII characters, stored beginning from the location addressed by the byte-pointer operand-2 plus one.
			The following elements are used by the sub- routine and macro: D0; D1; D2; T4; T5; R14; R15. A minus sign will precede the con- verted value if it was negative; at the conclusion of the instruction, the byte pointer operand-2 addresses the last convert- ed byte. MBDSUB deletes leading zeros, but converts at least one character; MBDNSUB converts at least "n" characters, padded with leading zeros if necessary.
MDB	r,t	3	Move Decimal character to Binary word;
	r,d	3	ASCII decimal to binary conversion. The word at the location addressed by operand-2 is multiplied by 10, and a value (as defined for the MXB instruction) from the byte

				addressed by the addressing register is added to it. The arithmetic condition flags are not reset, and arithmetic overflow cannot be detected.
MBX	h,r	3		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). The four low order bits 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.
	t,r	3		
	d,r	3		
MBX	n,h,r	6	(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 a 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.
MBXN	n,t,r	6		
	n,d,r	6		
MXB	r,h	3		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 6. The arithmetic condition flags are not reset by this instruction, and arithmetic overflow cannot be detected.
	r,t	3		
	r,d	3		

3.7.8 EXECUTION TRANSFER INSTRUCTIONS

B	l	2	Branch; branch to location defined, in the current frame, defined by label "l".

BSL	l	2		Branch and Stack Location; Subroutine call to mode defined by mode-ID "m" or to local label "l". The location of the instruction following the BSL, minus one, 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.
	m	3		
BSLI		1		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		1		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.
ENT	m	3		ExterNal Transfer; Branch to location defined by mode-ID"m".
ENTI		1		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	4	(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.
	t	4		
	d	4		
ENT*	h	4	(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.
	t	4		
	d	4		

3.7.9 I/O AND CONTROL INSTRUCTION

IOI	r,n_1,n_2	3	I/O Instruction Input; this instruction is used to set up block transfer starting and ending addresses and start input for 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 Section 2.7 for details.
IOO	r,n_1,n_2	3	I/O instruction Output; as above this instruction controls output to peripheral devices.
READ	r	2	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	2	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	3	Some standard calls are provided for functions which can only be performed in monitor code. These include: MCAL r,5,11 (corelock) MCAL r,6,11 (unlock)
RQM		3	Process releases the remainder of its time quantum to the monitor. Equivalent to: MCAL 0,0,9.

			register, or output (OB) from the location addressed by the addressing register. I/O pointers must be set up initially with an IOO instruction.
NOP		l	No OPeration is performed by this instruction.

3.7.10 ASSEMBLER DIRECTIVES

l	ADDR	n,n	Defines the local symbol "l" as a storage register in unlinked format. The displacement is defined by the first operand. The FID is defined by the second operand.
l	AR	l r n	Defines the local symbol "l" as an address register with a value defined by the operand.
l	CHR HTLY TLY DTLY SR	l n	Defines the local symbol "l" (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 opcode, 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.
l	DEFM	r,l r,n n,l n,n	Defines the local symbol "l" to be of type m; a mode-ID with entry point defined by the first operand and FID defined by the second operand.
l	DEFk	r,l r,n n,l n,n	Defines the local symbol "l" 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 l 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           DEFk    l
```

Defines local symbol "1" to be of type "k" (where k=b,c,d,h,l,s,t) with base register and displacement defined by the operand.

```
1           DEFTU   d
                    s
```

Defines local symbol "1" to be of type "t" with base register and displacement defined by the upper (left-most) tally of the operand.

```
1           DEFTL   d
                    s
```

Defines local symbol "1" to be of type "t" with base register and displacement defined by the lower (right-most) tally of the operand.

```
1           DEFDL   s
```

Defines local symbol "1" to be of type "t" with base register and displacement defined by the lower double tally of the operand.

l	EQU	c h t d s l n	Equates the local label "l" 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	l n	Resets the location counter to value defined by the operand. This statement may have a label field entry.
	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.

3.7.11 ADDRESS REGISTER USAGE

In some, a displacement is added to the contents of the address register to form an effective address. The length of the operand(s) is (are) encoded in the instruction.

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	m bytes	1 byte
Rn,Hm	m bytes	
Rn;Tm	2*m bytes	2 bytes
Rn;Dm	4*m bytes	4 bytes
Rm;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.
```

3.7.12 REAL INSTRUCTION SIDE EFFECTS

Many of the REAL opcodes use functional elements not specified as operands for execution. Those instructions are so footnoted in the previous listing; 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.

3.8 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 user's Master Dictionary (M/DICT) 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 initialized before the assembly starts. TSYM is a permanently defined file on a user's account. It can be examined at the conclusion of the assembly. Although TSYM has a lock to limit its use to one person on an account during an assembly, entries from one assembly disappear when another starts. TSYM is also used by the FIX-FILE-ERRORS verb.

3.8.1 TSYM/PSYM TABLE ENTRY FORMATS

The item format of the entries in the PSYM and 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.

Symbol-Code	Description - Symbol Type	Unit of Displacement
B	Bit	Bits
C	Character Register	Bytes
D	Double-Word (4-byte)	Words
H	Half-Word (1-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

3.8.2 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 opcode), 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 opcode first, and, if it is not found, a second attempt will be made with the basic opcode suffixed as described above.

3.8.3 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 opcode. If no entry is found in the TSYM and 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.

3.9 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) location object-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.

3.10 ASSEMBLER ERROR MESSAGES

*UNDEF: $Symbol_1$ $Symbol_2$....	Undefined symbols at end of pass 1 (Message at end-of-mode).
*MULT-IDEF	Label-field entry was previously defined.
*REF: UNDEF	Reference to undefined symbol.
LBL REQD	Required label-field missing.
*OPCD?	Opcode-field entry missing.
*OPRWD REQD	Required operand-field entry missing.
*ILGL OPCD:opcode	Either the opcode was illegal, or the operand types were illegal for the opcode.
*OPRWD RNG	The range of the operand-field entry is illegal.
*TRUNC.	Object code truncation may be due to: branch out-of-range; TSYM/PSYM table entry error; specification error in the GEN primitive.
*OPRND DEF	The operand-field entry is improperly defined, e.g.: non-hexadecimal character in a hexadecimal string.

The following are errors in the OSYM-table entry specifications.

*FRMT. A-FIELD *FRMT. B-FIELD	Error in A- or B-field specification.
*OPCD TYP!	Opcode type not a P/Q/M, or primitive type was illegal.
*MACRO DEF!	Error in the macro specification.

3.11 REAL INSTRUCTION SUMMARY

3.12 PROGRAMMING CONSIDERATIONS AND CONVENTIONS

3.12.1 REENTRANCY

In practically all cases, the system software is reentrant, that is, the same copy of the object code may by 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 Quaternary 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 and should not be used by any other programs. The Quaternary Control Block has no register addressing it; it is used by some system software (magnetic tape routines, for example, which temporarily set up a register pointing to it); its used is reserved for future software extensions.

A user program may utilize storage internal to the program if it is to be non-reentrant. Often it will be found that the functional elements defined in the PSYM will be sufficient.

In some cases it may be required to set up 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 (NOTE USAGE
          CMNT                     OF STORAGE INTERNAL TO 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 LOCKED
          RQM                      WAIT (RELEASE QUANTUM)
          B     LOCK               TRY AGAIN
          .
          .
          .
   UNLOCK MCC   X'01',R1           UNLOCK PROGRAM
```

3.12.2 WORK-SPACES OR BUFFERS

There is a set of work-spaces, or buffer areas, that is predefined 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 (CPB+28 through PCB+31) that are unused by the system, subroutines, through they are used by some of the processors.

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 "moving 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, although the AF A/R is used as a scratch register.
CS	4 (disp.=100)	100	No	As above.
IB	4 (disp.=200)	$\leq$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

Work-Space	Location (Offset From PCB)	Size	Linked?	Remarks
				freely usable except when explicitly or implicitly calling a terminal input routine.
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	Used for conversions.
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).
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 and OSBEG respectively (not at the true end if additional work-space is assigned at LOGON time). IS and OS A/R's are freely usable except when calling system subroutines that use them.

Work-Space	Location (Offset From PCB)	Size	Linked?	Remarks
	28-31			Used for compilation and execution of the RPG programs, and by the DATA/BASIC Debugger.

3.12.3 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. 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 set up R3 only once in the above manner. However, exit to TCL via WRAPUP will reset R3 to PCB+3.

3.12.4 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 set up 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 that can be handled by this scheme are: INC register; MCI; MIC; MII; MIID; MIIT; SCD; MIIR. Care should be taken in the case of MIIR to save register R15 in the subroutine.

Example:

```
      .
      .
      .
MOV     XXX,XMODE       SET UP XMODE FOR NEXT INSTRUCTION
MII     R12,R13,SR4     COPY FROM R12 TO R13, TILL R12=SR4
ZERO    XMODE
      .
      .
      .
```

Example: (continued)

```
!XXX  EQU     *               ENTRY POINT FOR SUBROUTINE
      MOV     R15, SR20       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   SET UP INTERFACE FOR ATTOVF
      BSL     ATTOVF          GET ANOTHER FRAME FROM OVERFLOW

      MOV     SR20,R15        RESTORE R15
      RTN                     RETURN TO CONTINUE EXECUTION OF MII
                              INSTRUCTION.
```

3.12.5 INITIAL CONDITIONS

At any level in the system, the following elements are assumed to be set up; 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	

3.12.6 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.

Functional Element	Description
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'.
SYSPRIV1	If set indicates system privileges, level one.
SYSPRIV2	If set in addition to SYSPRIV1, indicates system privileges, level two.
T0 through T3	Overlays accumulator and extension.
XMODE	This tally may be set up to a mode-ID of a subroutine that is to gain control when a "forward link zero" condition occurs.
WMODE	If WMODE is non-zero on any entry to WRAPUP, a BSL* through WMODE will be executed at the termination of history-string processing, before 1) the print-spool-files are closed, and 2) the overflow chain is released. The BSL* instruction will be executed whether RMODE is zero or not. This feature may be used by processors that require special WRAPUP processing.
USER	Tally 'USER' in the PCB has global significance: Tally=0 Indicates not logged on. Tally=-1 Indicates the spooler process. Tally=1 Indicates the file restore process. Tally=2 Indicates a process which must go to LOGOFF after WRAPUP processing. Other values indicate normal logged on processes.

SECTION 4

THE INTERACTIVE DEBUGGER (DEBUG)

The Interactive Debugger (DEBUG) provides a means for monitoring and controlling program execution. For all Reality users, DEBUG has the ability to turn the print off at the terminal, and to terminate program execution.

The use of the extended facilities of DEBUG (other than turning the terminal printing on and off, and terminating program execution) require system privileges level two. If the user has such privileges, he may control program execution by the insertion of break-points in the program, and by executing specific DEBUG commands. The user may also trace execution by displaying data at specific locations. DEBUG additionally allows the user to display data throughout the virtual memory of the system.

Thus (for users with system privileges level two) DEBUG is ideally suited for the checkout phase of assembly language programming.

4.1 ENTERING DEBUG

DEBUG is entered voluntarily by depressing the BREAK key on the terminal (INT key on some terminals). DEBUG will then display the location of the execution interruption point, followed by the DEBUG prompt character; the DEBUG prompt character is the exclamation mark (!).

DEBUG is entered involuntarily when a hardware trap condition occurs. In this case, DEBUG will display a message indicating the nature of the error causing the trap (see Section 4.6), followed by the location at which the trap occurred, followed by the DEBUG prompt character (!).

When the DEBUG prompt character is displayed, the user enters an appropriate DEBUG Control Command or DEBUG Data Display Command.

4.2 THE DEBUG CONTROL COMMANDS

4.2.1 CONTROL COMMAND SYNTAX

Prior to describing the actual DEBUG Control Commands, it is necessary to define the terms "address" and "indirect-address".

ADDRESS

An "address" references a byte in virtual memory. An "address" consists of a frame-ID (FID) and an offset byte displacement within the frame. The FID and/or displacement may be either in decimal or hexadecimal. The general forms of an "address" are shown below ("f" represents the FID value, and "d" represents the displacement value).

Address	Description
f,d	FID in decimal; displacement in decimal.

Address	Description
f.d	FID in decimal; displacement in hexadicimal.
.f,d	FID in hexadecimal; displacement in decimal.
.f.d	FID in hexadecimal; displacement in hexadecimal.
.d	Displacement in hexadecimal.
,d	Displacement in decimal.

If the FID value is omitted, then the PCB FID is used as a default value. The displacement must be in the range $0 \leq d < 512$.

As a general example, the following "addresses" are equivalent:

```
12.3C
12,60
.C.3C
.C,60
```

INDIRECT-ADDRESS

An "indirect-address" 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 following forms.

Indirect Address	Description
Rr	Specifies Address Register "r" (where "r" is a decimal value in the range $0 \leq r \leq 15$).
R.r	Specifies Address Register "r" (where "r" is a hexadecimal value in the range $0 \leq r \leq F$).

Note that "indirect-addresses" have an implied displacement within the FID that the Address Register is pointing to; this displacement depends on whether the register is in the "linked" or the "unlinked" format (see Section 2).

4.2.2 DEBUG CONTROL TABLES

DEBUG maintains three tables which may be manipulated by the DEBUG commands: 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, then the data pointed at by the entries are printed whenever a break message is printed (see Section 4.4). Up to four entries can be placed in each of these tables.

4.2.3 CONTROL COMMANDS

The following is a list of the DEBUG Control Commands. Users *without* system privileges level two may only use the P, G, END, and OFF commands.

General Form	Description
A address	Displays the address of an element.
B address	This command adds the "address" to the Break Table.
D	This command displays the Break Table and Trace Table.
En	This command sets the Execution Counter to "n", where "n" is a positive integer < 250. Setting the Execution Counter causes a break to occur after the execution of every "n" instruction. The command "E 0" or simply "E" turns off the Execution Counter.
END	This command terminates execution and returns to TCL. "END (carriage-return)" re-initializes the break and trace tables, whereas "END (line-feed)" preserves the tables.
G or line-feed	This command causes resumption of process execution from the point of interruption. G cannot be used if a process ABORT condition caused the entry to DEBUG.
G address	This command causes resumption of execution at the specified "address".
H	"HUSHES" terminal output (this is an on/off toggle).
K address	This command "kills" the break-point (i.e., deletes "address" from Break Table). "K" alone kills all break-points.
L	Display frame links.
M	Each entry of an M command switches (toggles) "Modal-Break" status ON and OFF. When "Modal-Break" status is ON, a break in execution will occur upon all intermodal transfers (i.e., BSL or ENT instructions; see Section 3.7.8). The message "ON" is displayed when the M command switches "Model-Break" on; the message "OFF" is displayed when "Modal-Break" is switched off.
Nn	This command sets the Break-Point Counter to "n" (i.e., inhibits traps until "n" breaks have occurred). "N" is equivalent to "N 0".
OFF	This command logs the user off of the system.
P	Each entry of a P command switches (toggles) from print suppression to print non-suppression. The message OFF is displayed if output is currently suppressed. The message ON is displayed if output is resumed.

General Form	Description
T	Each entry of a T command switches (toggles) suppression of display of entries in the trace tables.
T format address; window	This command adds the "address" to the Trace Table with the given display format and window, if present. Default display is hexadecimal, 4 bytes. No negative displacement for windows is allowed.
T format/symbol; window	This command adds the address referenced by the "symbol" to the Trace Table with the specified or default format and window. Default format and window depends on "symbol" type.
T format indirect-address; window	This command adds the "indirect-address" to the Indirect Trace Table with the specified or default format and window.
T format * symbol; window	This command adds the address referenced indirectly by the "symbol" (A/R or S/R) to the Indirect Trace Table with the specified or default format and window.
U address	This command deletes the "address" from the Trace Table.
U indirect-address	This command deletes the "indirect-address" from the Indirect Trace Table.
U	This commands deletes all addresses and indirect-addresses from the trace tables.
/,¢	Symbolic displays of elements.

4.3 THE DEBUG DATA DISPLAY COMMANDS

4.3.1 WINDOWS

Before describing the Data Display commands, it is necessary to define the concept known as a "window."

A "window" 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 general forms of the "window" are shown below.

Window	Description
;m	Number of bytes in decimal.

Window	Description
;.m	Number of bytes in hexadecimal.
;n,m	Displacement in decimal; number of bytes in decimal.
;n.m	Displacement in decimal; number of bytes in hexadecimal.
;.n,m	Displacement in hexadecimal; number of bytes in decimal.
;.n.m	Displacement in hexadecimal; number of bytes in hexadecimal.

The default "window" is 0,4 (no negative displacement, display four bytes).

4.3.2 DATA DISPLAY COMMANDS

The following is a list of the DEBUG Data Display commands.

General Form	Description
Caddress;window Cindirect-address;window	These commands display specified data in character format.
Xaddress;window Xindirect-address;window	These commands display specified data in hexadecimal format.
Iaddress;window Iindirect-address;window	These commands display specified data in integer format. ("window" must be ≤6)
Format/symbol;window	This command displays data referenced by "symbol" in given or default format and window size.
Format*symbol;window	This command displays data referenced indirectly by "symbol" in given or default format and window size.
A	This command displays the address at which program execution was interrupted.
A/symbol	This command displays the address referenced by "symbol".
A*symbol	This command displays the address referenced indirectly by "symbol".
L fid	This command displays the link fields of frame "fid".

General Form	Description
L*symbol	This command displays the link fields of the frame referenced indirectly by "symbol".

Immediately after the data at the specified address has been displayed, DEBUG prompts with an equal sign (=). The user then enters either a Data Replacement Specification or a Special Control Character.

4.3.3 DATA REPLACEMENT SPECIFICATIONS

Displayed data may be altered (replaced) by entering the new data in one of the following forms (after DEBUG prompts with an equal sign).

General Form	Description
.xxxxxx...	Replaces data with hexadecimal string "xxxxxx". The string should contain an even number of hexadecimal digits, and may be up to 80 digits in length.
'cccccc...	Replaces data with character string "cccccc". The string may be up to 80 characters in length.
n	Replaces data with integer value "n".

In the case of a hexadecimal or character string replacement, the data actually replaced may extend beyond the currently defined "window".

A Special Control Character (see Section 4.3.4) *must* be entered immediately following a Data Replacement Specification.

4.3.4 SPECIAL CONTROL CHARACTERS

The user may enter a Special Control Character in response to the DEBUG equal sign prompt character. In addition, the user must terminate a Data Replacement Specification (see Section 4.3.3) with a Special Control Character.

The Special Control Characters are listed below.

Control Character	Description
Carriage Return	Terminates display mode; DEBUG will prompt with an exclamation mark (!).
Line Feed	Displays data in the next "window" (i.e., the previously specified "address" or "indirect-address" is updated according to the currently specified "window"). The data is displayed on the same line.
Control-N	Displays data in the next "window", preceded by the address being displayed (in the format "f.d", where f is in decimal and d is in hexadecimal).

Control Character	Description
Control-P	Displays data in the previous "window" preceded by the address being displayed (in the format "f.d").

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.

Generally speaking, Control-N displays the set of bytes the same size as and immediately following the current display, and Control-P displays the immediately preceding set, with each skipping first to the next line and preceding the display of these bytes with their address (Line-Feed functions the same as Control-N, without skipping a line or displaying an address). Exceptions occur only in the case of the specification in the initial display of a negative displacement window, i.e., a window of the form:

```
:Windowl, Window2
```

Where windowl is positive.

In these cases, the address of the beginning of the next byte-set display is determined by the formulas:

For Control-N and Line-Feed:

```
ADDR OF DISPLAY = ADDR OF CURRENT DISPLAY + SIZE WINDOW - DSPLC WINDOW
```

For Control-P:

```
ADDR OF DISPLAY = ADDR OF CURRENT DISPLAY - SIZE WINDOW - DSPLC WINDOW
```

The user may describe a sequence by careful specification of size and displacement windows. A few examples follow.

Display a data list of DTLYS from right to left, i.e., by diminishing addresses, first displaying the DTLY at address 200.100. The easiest way is to simply use Control-P with a non-negative displacement window:

```
!X200.100;D0 .C1F1043F=     (Control-P)
 200.FC  .07510254=         (Control-P)
 200.F8  .A10551F0=         (etc.)
```

Another way of reading right to left, using Control-N, is accomplished by specifying the value of the displacement window (:window) to be twice that of the value of the size window (.window2 (= 4 for DTLYS)):

```
!200.108;8.4 .C1F1043F=     (Control-N)        (display DTLY at 200.100)
 200.FC  .07510254=         (Control-N)
 200.F8  .A10551F0=         (etc.)
```

To display an address over and over, as when monitoring changes at a certain address, the Line-Feed function may be used, specifying a displacement window equal in value to the size window. For example:

```
!I510.102;2,2 5000=  5000=  5000=  5001=  5001=  5002=
                        (Line-Feed display of tally at 510.100)
```

A somewhat more tricky example: suppose one has sorted a list of five-letter words beginning at the 100th data byte of linked frame 510 and wishes to check it for correct order by comparing items 0 and 1, 1 and 2, 2 and 3, and so forth. This may be done, using Control-N, by specifying a size window twice the value of the displacement window:

```
!C510.106;6,12 APPLECHAIR=     (Control-N)
 +510.111 CHAIRCHOIR=          (Control-N)
 +510.117 CHOIRFUNNY=          (Control-N)
 +510.11D FUNNYHELLO=          (etc.)
```

4.4 THE FORMATTED TRACE

The TRACE facility also allows formatting. This enables the user to specify a format and one window only (the size window) for each item traced. The display of each item will then reflect its specified format and byte size. Forward or backward displacements will be ignored. Note, however, that the default format and window of an indirect trace is hex display of 4 bytes, not the preceding window.

Examples:

```
!TX200.100;4+
```
(Will trace location 200.100 with 4 bytes displayed in hex - the '+' prompt from DEBUG indicates entry into the table.)

```
!T/CH1+ 736.21
```
(Trace of symbol CH1 - format = C, display size = 1 character - prompt 736.21 = display address of CH1.)

```
!T*R15;T0+
```
(Indirect trace R15 - format = I, window size = 2 bytes.)

```
!T*SR4+ 737.E0
```
(Indirect trace SR4 - format = X, display size = 4 bytes - default trace format and window is hex with 4 bytes, not previous format and window.)

```
!D
 BRK TBL: 0.  0.  0.  0.
 TRC TBL: 200.100  736.21  0.  0.
*TRC TBL: R 15.  * 737.E0  0.  0.
```
(Display of above entries in trace tables - 736.21 = display address of CH1, * 737.E0 means the address pointed to by the S/R at 737.E0 (i.e., SR4) will be displayed.)

4.5 SYMBOLIC REFERENCES

Symbolic reference to system-defined or user-defined data items is possible with the use of the SET-SYM and SET-SYM2 verbs. These TCL-II verbs are issued to specify tables for symbolic operands to be referenced by DEBUG. Entries in these tables must be in the format used in the Assembler PSYM and TSYM files.

SET-SYM assigns one symbol table; SET-SYM2 assigns another. Typically, SET-SYM is

used to reference standard system-defined elements, and SET-SYM2 is used to reference user-defined elements. For example:

DEBUG always looks for a symbolic operand first in the table set up by SET-SYM2. If this table in not assigned, or if the symbol is not found, it then looks in the table set up by SET-SYM.

4.5.1 SYMBOLIC OPERATORS

The symbolic operators '/' and '*' respectively indicate that a symbolic or indirect symbolic operand is to follow. They may be preceded by any format specification (X.I.C) or followed by a window specification (:window1.window2 or :symbolic window) which will override the listed default display values. DEBUG will display only those symbols from the Symbol Table which would be accepted by the Assembler as legal in a normal assembly.

4.5.2 DISPLAY FEATURES

Symbolic operands for display may be any properly defined bit, character, half-word, word, double-word, triple-word, storage register, or address register within the assigned Symbol Table. Normal display features are as follows:

Type of Symbol	Format of Display	# of Bytes Displayed
HTLY	Integer (I)	1
TLY	Integer	2
CHR	Character (C)	1
DTLY	Integer	4
FTLY	Hex (X)	6
S/R	Hex	6
A/R	Hex	8
S/R (INDIRECT)	Previous format	Previous window
A/R (INDIRECT)	Previous format	Previous window

These values are default values and are superceded whenever a specific format or window size is entered as part of a command.

Examples:

```
!/CTR5  31=              (Symbol = CTR5, format = I, display
                         size = 2 bytes.)

!/R15 708.CA     .000000CA800002C4=

                         (708.CA is the address pointed to by R15-
                         see 'the address function' - format = X,
                         window size = 8 bytes.  These are the con-
                         tents of R15.)

!X/D0 008C008C=          (Actual stored contents of accumulator -
                         format = X (as specified), display size = 4
                         bytes.)
```

Examples: (Continued)

```
!*R15 708.CA      .2D2F2A2F=    (Indirect display - contents at 708.CA -
                                format = X (prev. format). Window size =
                                4 bytes (prev. window).)

!C*R15;0,4 708.CA  -/*/-        (Format = C, window = 4 bytes with no
                                negative displacement)
```

A '+' indicates an address of a symbolic operand defined within a linked frame where 11 (hex 'B') has been added to the displacement to produce a display address starting from byte 1 of the frame.

4.5.3 SYMBOLIC WINDOWS

The symbolic window provides a useful means of referencing data pointed to by an A/R or S/R. It also enables the user to specify a forward reference from the address pointed to and carries an implicit default format specification.

Examples:

```
!*R9;D0 708.32  17301644=     (Specifies the double-tally pointed to
                              by R9)

!*R3;T2 705.4   12593=        (Specifies second tally after the tally
                              pointed to by R3)

!*SR6;C1 +32075.13A  ,=       (Gives the character HTLY immediately after
                              the one pointed to by SR6 - implicit format = C)

!X/SR4;T0  .012F=             (Gives the displacement (in hex) of SR4)

!/CTR4;S2  .009900010035=     (Implicit format = X, size = 6 bytes)
```

4.6 THE ADDRESS FUNCTION

The address function is evoked by preceding a symbolic operator by the command 'A'. An indirect symbolic operator preceded by the command 'A' yields the address pointed to by the specified A/R or S/R.

The command 'A' alone, not followed by any operators, will yield the interrupt address from which execution was halted when the DEBUGGER was invoked. If the DEBUGGER was not entered due to an error trap condition, this address is also the address from which execution will continue if a 'G' command without a specified address is given. Some examples of the use of the 'A' function follow:

```
!A/CTR5 512.9A                (Display address of symbol CTR5)

!A*SR1 +534.2F                (Adjusted byte address on linked frame pointed
                              to by S/R SR1)

!A*SR1;4.0 +534.2B            (Address of DTLY preceding address pointed to
                              by S/R SR1)
```

!A 6.94	(Address from which execution interrupted when DEBUG entered. Execution will continue from this address also.)

4.7 THE LINKS FUNCTION

This facility enables the user to display the forward and backward links of a specified frame as well as the number of next contiguous frames (NNCF) and number of previous contiguous frames (NPCF). The links of a frame pointed to be an A/R or S/R may also be obtained by an indirect symbolic links specification. The format of display is:

NNCF : FORWARD LINK BACKWARD LINK : NPCF

Examples:

!L727 4 : 728 726 : 1	(Links for frame 727 - 4 contiguous linked frames follow beginning at frame 728. 1 contiguous linked frame precedes frame 726.)
!L*IRBEG 14891. 0 : 14893	0 : 0 (S/R IRBEG points to frame 14891 - frame 14891 has no immediately contiguous links. Forward link is 14891. No backward link.)
!L.1F 28 : 68944000	-179407469 : -112 (These are the 'links' of frame 31 which is not a linked frame. No test is made to determine if a frame is linked or not before play. If NNCF = 28 or 29 then the frame is probably not linked.)

4.8 BIT DATA

4.8.1 SYMBOLIC BITS

Symbolically defined bits may also be displayed, providing they are defined within a 32-byte displacement range of their reference base register. Among the display functions are:

Control-N	Skip to next line, display bit address and value of next bit.
Line-Feed	Display bit value on same line.
Control-P	Skip to next line, display bit address and value of previous bit.

4.8.2 BIT ADDRESSES

The address function may also be used for bit operands. A bit address has the form:

(+)FID.DSP:BIT

where BIT is the bit displacement of the byte display address.

Examples:

```
!/ABIT 0=                       (ABIT is not set.)

!/RMBIT 1=      (Control-N)     (RMBIT is set.)

512.13:7  0=    (Control-P)
512.13:6  1=                    (Display RMBIT again.)

!A/LPBIT 512.19:5               (LPBIT is the 5th bit off address .19 of the
                                PCB.)
```

4.8.3 REPLACING BIT DATA

Bit data values are changed by placing the desired value for the bit (0 or 1) after the '=' prompt. Up to 10 values in succession may be altered by placing a string of 1's and 0's after the prompt.

Examples:

```
!/OVFBIT 1=0                    (Reset Overflow flag.)
!/OVFBIT 0=                     (Display of new OVFBIT value.)

!/DBIT 0=111111                 (Set bits DBIT = IBIT.)
!/DBIT 1= 1= 1= 1= 1= 1=        (Display new values of DBIT - IBIT using
                                Line-Feed function.)
```

4.8.4 BIT WINDOWS

An alternative means of bit display and modification is the bit window. This is a symbolic window using the character 'B', followed by a decimal bit displacement, as follows:

```
!*R15;B6 516.CA:6 1=            (6th bit off address pointed to by 15)

!200.100;B0  200.100:0  1=      (Leading bit of address 200.100)

!A*R15;B100 516.D6:4
```

4.9 BREAK MESSAGES

DEBUG causes an execution break to occur when the BREAK key on the terminal is depressed. DEBUG also has the facility to break on intermodal transfers (i.e., BSL or ENT instructions; see Section 3.7.8); the M command acts as an alternate action switch, to change this feature from ON to OFF. A break can also be initiated with the E command, causing a break after the execution of a specified number of instructions. The following 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 instruction encountered).
R f.d	Return (RTN) instruction encountered.

where "f" is the decimal FID and "d" the hexadecimal displacement, representing the location of the execution interruption point.

The Execution Break and Address Break facilities are mutually exclusive. When the Execution Counter is positive, Break Table entries are ignored. However, the Break Table of the Execution Break facility can be used with the Modal Break facility.

4.10 EXAMPLES

4.10.1 SIMPLE EXAMPLE

The following example illustrates a simple DEBUG interaction. The features illustrated here may be used by all Reality users.

```
:[illegible]                          ENGLISH LIST statement

SYSTEM-MODES..... FRAME.....

WSPACES           FRAME 172           Listing output from
EDIT-I            FRAME 013           system.
PQUEUE/1200       FRAME 164
WRAPUP-II                             BREAK key depressed.
I 6.1A3                               Interrupt message.
!P CR OFF                             Turns Print off.
!G CR                                 Go (resumes execution without
                                      printing).
                                      BREAK key depressed.
I 3.FB                                Interrupt message.
!P CR ON                              Turn Print back on.
!G CR                                 Go (resumes execution with
                                      printing).
DB3               FRAME 018
DB4               FRAME 019           Listing output resumes.
TAPEIO-II         FRAME 036
DB5                                   BREAK key depressed.
I 6.137                               Interrupt message.
!END CR                               Terminates LIST execution.

:                                     TCL prompt.
```

4.10.2 EXTENDED EXAMPLE

The following example illustrates use of the extended DEBUG facilities. These facilities can be used only by users with system privileges level two.

```
:                                                    BREAK key depressed.
I 6.87
!X200.12;6 CR  .1E1327101881=012345678ABCDEF N      Display and change data.
 200.18  .CDEF34567890=N                             Display next window
 200.1E  .012C00000064=XSTMR P                       (no change).
 200.18  .CDEF34567890= CR .4B4A4D4E0064=M           Change data in
 200.24  .0004000A0000= CR                           character form.
!M CR ON                                             Set Modal Trace on.
!N 5 CR                                              Set delay counter.
!T .40 CR                                            Trace location .40 in PCB.
!TR4 CR                                              Trace Register four.
!G CR                                                Go.

:HHMM CR                                             TCL statement.

R 5.49                                               RTN instruction encountered.
 512.40 = .000002060000                              Data from direct trace.
 R 4. : 528. = .004154545249                         Data from indirect trace.
M 7.3                                                Modal break.
 512.40 = .000002060000
 R 4. : 528. = .004154545249
R 5.78
 512.40 = .000020920000
 R 4. : 528. = .004154545249
M 10.1
 512.40 = .000020920000
 R 4. : 528. = .004154545249
M 8.1
 512.40 = .000020920000
 R 4. : 528. = .004154545249
R 10.32
 512.40 = .000020920000
 R 4. : 528. = .004154545249

!B CR                                                Display Break and Trace Table
 BRK TBL: 0.  0.  0.  0.                             entries.
 TRC TBL: 512.40  0.  0.  0.                         Break Table entries.
*TRC TBL: R 4. 0.  0.  0.                            Trace Table entries.
!END CR                                              Indirect Trace Table entries.
                                                     Terminate Execution.
:                                                    Back to TCL.
```

SECTION 5
SYSTEM SUBROUTINES

The following subroutines are from a computer printout. The subroutines are listed alphabetically.

DOCUMENTATION CONVENTIONS

IN THE SYSTEM SOFTWARE DOCUMENTATION, EACH ROUTINE IS LISTED ALONG WITH ITS ENTRY POINT (AS WOULD BE USED IN A DEFM STATEMENT); IF THE ENTRY POINT IS INCLUDED IN THE STANDARD PSYM FILE, IT IS FOLLOWED BY AN ASTERISK (*). UNLESS OTHERWISE SPECIFIED, ROUTINES ARE MEANT TO BE CALLED AS SUBROUTINES, USING A BSL INSTRUCTION, AND THEY RETURN TO THE CALLING PROGRAM VIA A RTN INSTRUCTION.

THE FUNCTIONAL DESCRIPTION SECTION FOR EACH ROUTINE BRIEFLY DESCRIBES THE ACTION TAKEN. THE INPUT INTERFACE, OUTPUT INTERFACE, AND ELEMENT USAGE SECTIONS DESCRIBE THE FUNCTIONAL ELEMENTS USED BY THE ROUTINE. THE SINGLE LETTER FOLLOWING AN ELEMENT NAME DESCRIBES ITS TYPE: B=BIT, C=CHARACTER, H=HALF TALLY, T=TALLY (WORD), D=DOUBLE TALLY, F=TRIPLE TALLY, R=ADDRESS REGISTER, S=STORAGE REGISTER. EVEN IF NOT SPECIFIED, THE FOLLOWING ELEMENTS MAY BE DESTROYED BY ANY ROUTINE; THE ONLY WAY TO BE SURE IS TO INSPECT THE CODE:

BITS	:	ARITHMETIC CONDITION FLAGS, SB60, SB61
TALLIES	:	T4, T5
DOUBLE TALLIES	:	ACCUMULATOR AND EXTENSION (D0, D1), D2
REGISTERS	:	R14, R15
STORAGE REGISTERS	:	SYSR0, SYSR1, SYSR2

IF NO DESCRIPTION FOLLOWS AN ELEMENT NAME, IT INDICATES THAT THE ELEMENT IS USED AS A SCRATCH ELEMENT.

THE SYSTEM DELIMITERS ARE SYMOLICALLY REFERRED TO AS FOLLOWS:

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

BCKSP (10,TAPEIO-I)

FUNCTIONAL DESCRIPTION

THIS ROUTINE BACK-SPACES THE TAPE BY ONE RECORD. IT CALLS INIT AND TPSTAT, AND REQUIRES FOUR ADDITIONAL LEVELS OF SUBROUTINE LINKAGE.

BLOCK-SUB

BLOCK-SUB (2,BLOCK-LETTERS)

FUNCTIONAL DESCRIPTION

THIS ROUTINE PRINTS BLOCK LETTERS ON THE TERMINAL OR LINE PRINTER. IT IS USED, FOR INSTANCE, BY THE TCL VERBS "BLOCK-TERM" AND "BLOCK-PRINT"; FOR MORE INFORMATION, SEE THE DISCUSSION OF THESE VERBS IN THE SYSTEM COMMANDS DOCUMENTATION.

INPUT INTERFACE

IS	R	POINTS ONE BEFORE THE FIRST CHARACTER TO BE OUTPUT; THE END OF DATA IS MARKED BY THE CHARACTER PAIR SM Z (NO SPACE AFTER THE SM); IF ANY ELEMENT IN THE DATA STRING CONTAINS A SM, IT MUST BE TERMINATED BY A SB (SEE MD18 DOCUMENTATION, "EDITING FEATURES")
ZFLG	B	IF SET, OUTPUT IS DIRECTED TO THE TERMINAL, OTHERWISE OUTPUT IS PASSED TO THE SPOOLER FOR LINE PRINTER LISTING OR OTHER USE
OBSIZE	T	CONTAINS THE MAXIMUM NUMBER OF CHARACTERS ON EACH OUTPUT LINE
OB	R	=OBBEG
SBO	B	IF SET, NO TEST FOR TERMINAL OR PRINTER OUTPUT IS MADE, TERMINAL OR PRINTER CHARACTERISTICS ARE NOT INITIALIZED, THE OUTPUT DEVICE IS NOT ADVANCED TO TOP-OF-FORM, AND THE HEADING IS NOT SET NULL; ALL THESE ACTIONS TAKE PLACE IF SBO IS RESET
AFBEG	S +	
BMSBEG	S +	POINT TO SCRATCH AREAS
HSEND	S +	
LISTFLAG	B +	

```
      FRMTFLG     B  +
      NOBLNK      B  +
      LFDLY       T  + AS REQUIRED BY WRTLIN
      PAGSIZE     T  +
      PAGSKIP     T  +
      PAGFRMT     B  +

OUTPUT INTERFACE

      OB          R     =OBBEG

      PAGINATE    B     =1

      PAGHEAD     S     POINTS  TO A NULL PAGE  HEADING (SM)  AT
                        HSEND IF SB0=0

ELEMENT USAGE

      BITS        C  +
      SC0         C  +
      SC1         C  +
      SC2         C  +
      REJCTR      T  +
      C1          T  +
      CTR16       T  +
      CTR17       T  +
      CTR18       T  +
      CTR19       T  +
      D0          D  +
      D1          D  +
      BASE        D  +
      MODULO      T  + UTILITY
      SEPAR       T  +
      IR          R  +
      UPD         R  +
      BMS         R  +
      AF          R  +
      OB          R  +
      CS          R  +
      TS          R  +
      R15         R  +
      SR4         S  +
      .
      SR22        S  +

      CTR1        T     USED BY CVDIR

      R14         R     USED BY RETIX

      T7          T  + USED BY WRTLIN
      SYSR1       S  +
```

SUBROUTINE USAGE

RETIX; GBMS IF THE SYSTEM FILE "BLOCK-CONVERT" IS FOUND; CVDIR; WRTLIN; NEWPAGE IF REQUIRED; PRNTHDR IF SB0=0; PCLOSEALL AND SETLPTR IF SB0=0 AND ZFLG=0; SETTERM IF SB0=1 OR ZFLG=1

SIX ADDITIONAL LEVELS OF SUBROUTINE LINKAGE REQUIRED IF "BLOCK-CONVERT" IS A "Q"-CODE ITEM IN THE MASTER DICTIONARY, OTHERWISE FIVE LEVELS REQUIRED

ERROR CONDITIONS

BLOCK-SUB EXITS TO WRAPUP (MD995 OR MD99) UNDER THE FOLLOWING CONDITIONS:

ERROR NUMBER	ERROR TYPE
520	NULL INPUT DATA
521	TOO MANY CHARACTERS (MORE THAN NINE) IN A WORD TO BLOCK
522	BLOCK-CONVERT FILE MISSING OR IMPROPERLY DEFINED IN THE MASTER DICTIONARY
523	BLOCK OUTPUT WOULD EXCEED PAGE WIDTH
524	AN INPUT CHARACTER IS NOT IN THE BLOCK-CONVERT FILE
525	AN INPUT CHARACTER IS IMPROPERLY FORMATTED IN THE BLOCK-CONVERT FILE

CONV (0,CONV)*
CONVEXIT (1,CONV)

FUNCTIONAL DESCRIPTION

THESE ENTRY POINTS ARE USED TO CALL THE ENTIRE CONVERSION PROCESSOR AS A SUBROUTINE, WHICH WILL PERFORM ANY AND ALL VALID CONVERSIONS SPECIFIED IN THE CONVERSION STRING. OTHER ENTRY POINTS MAY BE USED TO PERFORM CERTAIN SPECIFIC CONVERSIONS. MULTILPLE CONVERSION CODES ARE SEPARATED BY VM'S IN THE CONVERSION STRING. 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.

CONV IS THE USUAL MODE-ID USED TO INVOKE CONVERSION PROCESSING. CONVEXIT IS THE ENTRY POINT TO WHICH ANY PART OF THE CONVERSION PROCESSOR RETURNS IN ORDER TO CHECK IF MORE CONVERSION IS REQUIRED (FURTHER VM'S AND CONVERSION CODES IN THE CONVERSION STRING).

INPUT INTERFACE

TSBEG	S	POINTS ONE BEFORE THE VALUE TO BE CONVERTED; THE VALUE IS CONVERTED "IN PLACE", AND THE BUFFER IS USED FOR SCRATCH SPACE; THEREFORE IT MUST BE LARGE ENOUGHTO CONTAIN THE CONVERTED VALUE; THE VALUE TO BE CONVERTED IS TERMINATED BY ANY OF THE STANDARD SYSTEM DELIMITERS (SM, AM, VM, OR SVM)
IS	R	POINTS TO THE FIRST CHARACTER OF THE CONVERSION CODE SPECIFICATION STRING FOR CONV; FOR CONVEXIT, POINTS AT LEAST ONE BEFORE THE NEXT CONVERSION CODE (AFTER A VM) OR AM AT THE END OF THE STRING, OR TO THE AM; THE CODE STRING MUST END WITH AN AM; INITIAL SEMICOLONS (;) ARE IGNORED
MFLG	B	SET IF "INPUT" CONVERSION IS TO BE PERFORMED; RESET FOR "OUTPUT" CONVERSION

CONV, CONVEXIT

DFLG DAF1	B B	+ AS REQUIRED BY TRANSLATE (SEE TRANSLATE + DOCUMENTATION)
XFLG	B	AS REQUIRED BY CFUNC (SEE CFUNC DOCUMENTATION)

OUTPUT INTERFACE

TSBEG	S	POINTS ONE BEFORE THE CONVERTED VALUE
TS TSEND	R S	+ POINT TO THE LAST CHARACTER OF THE + CONVERTED VALUE; A SM IS ALSO PLACED ONE PAST THIS LOCATION; TS=TSEND=TSBEG IF A NULL VALUE IS RETURNED
IS	R	POINTS TO THE AM TERMINATING THE CONVERSION CODE(S)

ELEMENT USAGE

ELEMENT		CONVERSIONS WHERE USED
DFLG	B	F,T
XFLG	B	F
GMBIT	B	F
WMBIT	B	F
SB10	B	ALL
SB12	B	ALL
DAF1	B	T
DAF9	B	T
SC2	C	C,D,F,T
T3	T	F,MD
T4	T	D,F,MD,MT
T5	T	D,F,MD,MT
T6	T	D,F,M
T7	T	F,MD
CTR1	T	C,F,G,T
CTR12	T	F
CTR13	T	F
CTR20	T	ALL
CTR21	T	D,MD,T

CONV, CONVEXIT

CTR22	T	D
CTR23	T	D,MD
CTR28	T	T
D1	D	C,F,MT,T
D2	D	D,F,MD,MT
D3	D	MT
D7	D	F
D8	D	F
D9	D	F
FP0	F	F,MD
FP1	F	F,MD
FP2	F	F,MD
FP3	F	F
FP4	F	F
FP5	F	F
FPX (SYSR0)	F	F,MD,T
FPY	F	F,MD
BASE	D	T
MODULO	T	T
SEPAR	T	T
RECORD	D	T
SIZE	T	T
NNCF	H	T
FRMN	D	T
FRMP	D	T
NPCF	H	T

XMODE	T	C,F,MT,T
IR	R	T
BMS	R	T
R14	R	D,F,MD,MP,MT,MX,T
R15	R	ALL
SYSR1	S	T
SYSR2	S	T
S4	S	T
S5	S	F
S6	S	C,T
S7	S	ALL
SR0	S	C,F
SR1	S	F
SR4	S	C,T

SUBROUTINE USAGE

CONV, CONVEXIT

CVXIS FOR "U" CONVERSIONS; GCORR FOR "G" CONVERSIONS; TRANSLATE FOR "T" CONVERSIONS; PACKUN FOR "MP" CONVERSIONS; CONCATENATE FOR "C" CONVERSIONS; ADDITIONAL SUBROUTINES AS USED BY ROUTINES LISTED UNDER "EXITS" BELOW, AND BY USER-WRITTEN ROUTINES

THE NUMBER OF ADDITONAL LEVELS OF SUBROUTINE LINKAGE REQUIRED DEPENDS ON THE CONVERSIONS PERFORMED - SEE THE DOCUMENTATION FOR THE VARIOUS CONVERSION ROUTINES FOR MORE SPECIFIC INFORMATION; NOTE THAT FOR "F" CONVERSIONS, CFUNC MAY CALL CONV RECURSIVELY

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 FOR THE USER ROUTINE WILL BE IDENTICAL TO THAT DESCRIBED IN THE PRECEDING SECTION; AFTER PERFORMING THE CONVERSION THE USER ROUTINE SHOULD SET UP THE OUTPUT INTERFACE ELEMENTS TO BE COMPATIBLE WITH CONVEXIT, AND THEN EXIT VIA AN EXTERNAL BRANCH TO THAT POINT TO CONTINUE THE CONVERSION PROCESS IF MULTIPLE CONVERSIONS ARE SPECIFIED. ALTERNATELY, A RTN MAY BE EXECUTED IF THIS IS NOT NEEDED, OR TO PREVENT FURTHER CONVERSIONS FROM BEING PERFORMED. ELEMENTS USED BY THE REGULAR CONVERSION ROUTINES MAY SAFELY BE USED BY USER ROUTINES; HOWEVER, IF ADDITIONAL ELEMENTS ARE NEEDED, A COMPLETE KNOWLEDGE OF THE PROCESSOR THAT CALLED CONV (LIST, SELECTION, ETC.) WILL BE NECESSARY.

EXITS

TO IDATE FOR "D" CONVERSIONS ON INPUT (MFLG=1); TO ODATE FOR "D" CONVERSIONS ON OUTPUT; TO ICONVMD OR OCONVMD FOR "MD" CONVERSION ON INPUT OR OUTPUT; TO CFUNC FOR "F" CONVERSIONS; TO TIMECONV FOR "MT" CONVERSIONS; TO HEXCONV FOR "MX" CONVERSIONS; ALL THESE ROUTINES, HOWEVER, RETURN TO CONVEXIT

FOR OUTPUT CONVERSION, A NULL VALUE RETURNED CAUSES AN

CONV, CONVEXIT

IMMEDIATE END OF CONVERSION PROCESSING.

ERROR CONDITIONS

CONV EXITS TO WRAPUP AFTER SETTING RMODE TO 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 A "T" CONVERSION FILE

WRAPUP IS ALSO ENTERED WITHOUT SETTING RMODE TO ZERO UNDER THE FOLLOWING ERROR CONDITIONS:

708	VALUE CANNOT BE CONVERTED BY A "T" CONVERSION
339	INVALID FORMAT FOR INPUT DATA CONVERSION

CREAD

CREAD (2,CARDIO)

FUNCTIONAL DESCRIPTION

THIS ROUTINE 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	POINTS TO A SCRATCH BYTE; NORMALLY R2 ALWAYS POINTS TO BYTE ZERO OF THE PROCESS'S SCB
OBBEG	S	POINTS ANYWHERE WITHIN THE FRAME THAT THE CARD IS TO BE READ INTO, NORMALLY PCB+4

OUTPUT INTERFACE

CFLG	B	SET IF AN ATTEMPT WAS MADE TO READ A CARD; RESET IF NO CARD WAS READ
R2	R	UNCHANGED, BUT THE BYTE ADDRESSED CONTAINS THE STATUS OF THE CARD READER
R15	R	POINTS TO THE FIRST BYTE OF THE CARD READ, 80 BYTES FROM THE END OF THE FRAME POINTED TO BY OBBEG

ELEMENT USAGE

T3	T	USED AS A COUNTER FOR STATUS TIMEOUT AFTER A READ

SUBROUTINE USAGE

NONE

ERROR CONDITIONS

NONE, EXCEPT CARD READER ERRORS RETURNED AS STATUS;

CREAD

THE MEANING OF THE STATUS BITS IS AS FOLLOWS:

BIT	EXPLANATION OF SET CONDITION
0-2	UNUSED BY THE CONTROLLER, AND ALWAYS ZERO
3	CARD READER MECHANICAL ERROR (PICK FAILURE, CARD MOTION ERROR, ETC.)
4	EBCDIC ERROR DETECTED (E.G., AN INVALID PUNCH COMBINATION); THIS IS NOT AN ERROR IF CFLG IS ZERO, HOWEVER
5	INPUT HOPPER EMPTY

6	(ALWAYS RESET BY CREAD, AND USED ONLY FOR BYTE I/O)
7	CARD READER READY
3,5,7	IF THESE BITS ARE ALL SET, CARD READER POWER IS OFF

CVSUBS

STRING TO SIX-BYTE BINARY CONVERSION

FUNCTIONAL DESCRIPTION

THESE ROUTINES CONVERT A STRING OF ASCII DECIMAL OR HEXADECIMAL CHARACTERS TO THEIR BINARY EQUIVALENT; CONVERSION CONTINUES UNTIL AN ILLEGAL (NON-DECIMAL OR NON-HEXADECIMAL) CHARACTER IS ENCOUNTERED.

ON ENTRY, THE APPROPRIATE REGISTER (SEE TABLE) POINTS ONE PRIOR TO THE FIRST CHARACTER OF THE STRING; THIS CHARACTER MUST BE A PLUS SIGN, MINUS SIGN, OR NUMERIC (0-9 FOR DECIMAL ROUTINES, 0-9 AND A-F FOR 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). ALSO, ARITHMETIC OVERFLOW DUE TO TOO MANY DIGITS IN THE CHARACTER STRING CANNOT BE DETECTED.

INPUT INTERFACE

ENTRY POINT	REGISTER	CONVERSION
CVDR15 (4,SYSTEM-SUBS-I)*	R15	DECIMAL
CVDIS (5,SYSTEM-SUBS-I)*	IS (R4)	DECIMAL
CVDOS (6,SYSTEM-SUBS-I)*	OS (R5)	DECIMAL
CVDIR (9,SYSTEM-SUBS-I)*	IR (R6)	DECIMAL
CVDIB (11,SYSTEM-SUBS-I)*	IB (R10)	DECIMAL
CVXR15 (3,SYSTEM-SUBS-I)*	R15	HEXADECIMAL
CVXIS (7,SYSTEM-SUBS-I)*	IS (R4)	HEXADECIMAL
CVXOS (8,SYSTEM-SUBS-I)*	OS (R5)	HEXADECIMAL
CVXIR (10,SYSTEM-SUBS-I)*	IR (R6)	HEXADECIMAL
CVXIB (12,SYSTEM-SUBS-I)*	IB (R10)	HEXADECIMAL

OUTPUT INTERFACE

FP0	F	CONTAINS THE CONVERTED VALUE OF THE STRING IF LEGAL CHARACTERS ARE FOUND, OTHERWISE ZERO
CTR1	T	=D0 (EXCEPT FOR CVDR15 AND CVXR15, WHICH

DO NOT USE THIS ELEMENT)

NUMBIT	B	SET IF CONVERSION COMPLETED AND THE STRING IS TERMINATED BY A SYSTEM DELIMITER

ELEMENT USAGE

T3	T

SUBROUTINE USAGE

CVDR15 OR CVXR15 USED BY THE OTHER ROUTINES

ONE ADDITIONAL LEVEL OF SUBROUTINE LINKAGE REQUIRED, EXCEPT FOR CVDR15 AND CVXR15

DLINIT

DLINIT (2,WSPACES-II)*

FUNCTIONAL DESCRIPTION

DLINIT IS USED TO OBTAIN A BLOCK OF CONTIGUOUS OVERFLOW SPACE FOR A FILE. AFTER CHECKING THE INPUT PARAMETERS AND OBTAINING THE NECESSARY NUMBER OF FRAMES, IF AVAILABLE, IT ENTERS DLINIT1 TO INITIALIZE THE FRAMES (SEE DLINIT1 DOCUMENTATION). IF NOT ENOUGH SPACE IS AVAILABLE FOR THE FILE, DLINIT CALLS NOSPACE TO FIND OUT IF PROCESSING SHOULD BE ABORTED (SEE NOSPACE DOCUMENTATION).

INPUT INTERFACE

MODULO	T	* CONTAIN THE MODULO AND SEPARATION
SEPAR	T	* PARAMETERS FOR THE FILE; IF MODULO IS INITIALLY LESS THAN OR EQUAL TO ZERO, IT IS SET TO ELEVEN; IF SEPAR IS INITIALLY LESS THAN OR EQUAL TO ZERO, IT IS SET TO ONE, AND IF INITIALLY GREATER THAN 127 IT IS SET TO 127

OUTPUT INTERFACE

BASE	D	CONTAINS THE BEGINNING FID OF A CONTIGUOUS BLOCK OF SIZE MODULO*SEPAR IF THE SPACE IS AVAILABLE, OTHERWISE UNCHANGED

OVRFLW	D	=BASE IF THE REQUESTED SPACE IS AVAILABLE, OTHERWISE =0
RMBIT	B	SET IF THE REQUESTED SPACE IS OBTAINED, OTHERWISE UNCHANGED

ELEMENT USAGE

R14	R	+
R15	R	+ USED BY GETBLK
D0	D	+

SUBROUTINE USAGE

DLINIT

GETBLK; NOSPACE IF THE REQUESTED SPACE IS UNAVAILABLE

THREE ADDITIONAL LEVELS OF SUBROUTINE LINKAGE REQUIRED

EXITS

TO DLINIT1 IF THE REQUESTED SPACE IS OBTAINED; TO NSPCQ (WRAPUP) FROM NOSPACE IF THE SPACE IS UNAVAILABLE AND PROCESSING IS ABORTED BY THE USER

DLINIT1

DLINIT1 (0,WSPACES=II)

FUNCTIONAL DESCRIPTION

DLINIT1 INITIALIZES THE LINK FIELDS OF A FILE AS SPECIFIED BY ITS BASE, MODULO, AND SEPARATION PARAMETERS, AND SETS EACH GROUP EMPTY BY ADDING AN AM AT THE BEGINNING (IN THE FIRST DATA BYTE).

INPUT INTERFACE

BASE	D	+ CONTAIN THE BASE, MODULO, AND SEPARATIO
MODULO	T	+ OF THE FILE; NOTE = ONE FRAME IS LINKED
SEPAR	T	+ EVEN IF MODULO IS LESS THAN OR EQUAL TO ZERO

OUTPUT INTERFACE

R14	R	POINTS TO THE FIRST DATA BYTE IN THE FIRST FRAME OF THE LAST GROUP IN THE FILE (SET BY LINK)
R15	R	POINTS TO THE LAST BYTE OF THE LAST FRAME OF THE LAST GROUP IN THE FILE (SET BY LINK)
RECORD	D	=ONE GREATER THAN THE FID OF THE LAST FRAME OF THE LAST GROUP IN THE FILE
NNCF	H	=SEPAR-1

FRAMES ARE INITIALIZED AS DESCRIBED ABOVE

ELEMENT USAGE

CTR1	T	UTILITY
FRMN	D	+
FRMP	D	+ USED BY LINK
NPCF	H	+

SUBROUTINE USAGE

DLINIT1

LINK

ONE ADDITIONAL LEVEL OF SUBROUTINE LINKAGE REQUIRED

ECONV, ACONV

ECONV (0,EBCDIC)*
ACONV (2,EBCDIC)

FUNCTIONAL DESCRIPTION

THESE ROUTINES TRANSLATE A CHARACTER FROM EBCDIC OR ASCII TO ASCII OR EBCDIC. ECONV OPERATES ON EBCDIC INPUT, AND DOES NOT TRANSLATE CHARACTERS WITHOUT ASCII EQUIVALENTS - THESE CHARACTERS ARE RETURNED UNTRANSLATED. ACONV OPERATES ON ASCII INPUT, AND ALWAYS ZEROES THE HIGH-ORDER BIT OF THE CHARACTER BEFORE TRANSLATION.

INPUT INTERFACE

IB	R	POINTS TO THE CHARACTER TO BE TRANSLATED

OUTPUT INTERFACE

IB	R	UNCHANGED, BUT POINTS TO THE TRANSLATED VALUE

ELEMENT USAGE

D0	D	+ UTILITY
R15	R	+

SUBROUTINE USAGE

NONE

ENGLISH INTERFACE

ENGLISH INTFRFACE

SUMMARY

IT IS POSSIBLE TO INTERFACE WITH THE ENGLISH PROCESSOR AT SEVERAL LEVELS. A TYPICAL LIST OR SORT STATEMENT PASSES THROUGH THE PREPROCESSOR AND SELECTION PROCESSOR BEFORE ENTERING THE LIST PROCESSOR. ALL STATEMENTS MUST PASS THROUGH THE FIRST TWO STAGES, BUT CONTROL CAN BE TRANSFERRED TO USER-WRITTEN 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 SM'S. THERE IS ONE FILE-DEFINING ELEMENT IN EACH STRING, ONE ELEMENT FOR EACH ATTRIBUTE SPECIFIED IN THE ORIGINAL STATEMENT, AND SPECIAL ELEMENTS PERTAINING TO SELECTION CRITERIA, SORT-KEYS, ETC. THE FORMATS OF VARIOUS STRING ELEMENTS ARE AS FOLLOWS:

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

EXPLICIT ITEM-ID'S:

SM I ITEM-ID SM

END-OF-STRING ELEMENT:

SM Z

ENGLISH INTERFACE

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 BIT 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
1	PA	PA
2	35	35
3	XXXX	76
4		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, THE USER ROUTINE SHOULD EXIT INDIRECTLY VIA RMODE (USING AN ENT* RMODE INSTRUCTION); ON THE LAST ENTRY, THE ROUTINE SHOULD EXIT TO ONE OF THE WRAPUP ENTRY POINTS. PROCESSING MAY BE ABORTED AT ANY TIME BY SETTING RMODE TO ZERO AND ENTERING WRAPUP. BIT SB0 MUST ALSO BE SET ON THE FIRST ENTRY.

SPECIAL EXIT FROM THE LIST PROCESSOR

A USER PROGRAM MAY ALSO GAIN CONTROL IN 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

ENGLISH INTERFACE

STORED IN THE HISTORY STRING (HS AREA). THE FORMATS OF THE VERBS THEN ARE:

LINE NUMBER	NON-SORTED	SORTED
1	PA	PA
2	35	35
3	4D	4E
4	XXXX	XXXX

WHERE "XXXX" IS THE MODE-ID OF THE USER PROGRAM.

OUTPUT DATA IS STORED IN THE HS AREA; DATA FROM EACH ATTRIBUTE IS STORED IN THE STRING, DELIMITED BY AM'S; MULTIPLE VALUES AND SUB-MULTIPLE-VALUES ARE DELIMITED WITHIN AN ELEMENT 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 A SM. THE FORMAT IS:

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 UNTIL, 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 WORK SPACE, NORMALLY ONE BEFORE THE FIRST "X" CODE FOUND.

THE EXIT CONVENTION FOR THE LIST PROCESSOR IS THE SAME AS FOR THE SELECTION PROCESSOR (SEE ABOVE).

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
002             ZB      SB30            INTERNAL FLAG
003             BBS     SB1,NOTF        NOT FIRST TIME
004 * FIRST TIME SETUP
005             MOV     4,CTR32
006             SB      SB1
```

ENGLISH INTERFACE

```
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
020           CMNT    *              "X"
021 COPYIT    MIID    R14,OB,X'A0'   COPY ITEM-ID TO OB
022           MCC     C' ',OB        OVERWRITE AM
023           INC     OB,5           INDEX
024           B       SCANSM
025 ENDHS     BSL     WRTLIN         PRINT A LINE
026           MOV     SR28,HSEND     RESTORE HS TO FIRST
027           CMNT    *              "X" CODE
028           DEC     HSEND          BACK UP ONE BYTE
029           BBZ     RMBIT,QUIT
030 RETURN    ENT*    RMODE          RETURN TO SELECTION
031           CMNT    *              PROCESSOR
032 QUIT      ENT     MD999          TERMINATE PROCESSING
033           END
```

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 " " (BLANK) 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 ROUTINES MAY FREELY USE THE FOLLOWING ELEMENTS:

```
BITS            : SB20 UPWARDS
TALLIES         : CTR30 UPWARDS
DOUBLE TALLIES: 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. SB1 MAY BE USED AS A FIRST-ENTRY SWITCH BY USER PROGRAMS THAT GAIN CONTROL FROM THE LIST PROCESSOR.

BITS	SELECTION PROCESSOR	LIST PROCESSOR
AFLG	SCRATCH	NON-COLUMNAR LIST FLAG
BFLG	FIRST ENTRY FLAG	
CFLG	SCRATCH	SCRATCH
DFLG	SCRATCH	DUMMY CONTROL-BREAK
EFLG	RESERVED	CONTROL-BREAK FLAG
FFLG	RESERVED	SCRATCH
GFLG	RESERVED	RESERVED
HFLG	RESERVED	RESERVED
IFLG	EXPLICIT ITEM-ID'S SPECIFIED	
JFLG	RESERVED	D2 ATTRIBUTE IN PROCESS
KFLG	BY-EXP FLAG	BY-EXP FLAG
LFLG	SCRATCH	LEFT-JUSTIFIED FIELD
MFLG	CONV INTERFACE; ZERO	ZERO
NFLG	SCRATCH	SCRATCH
OFLG	SELECTION TEST ON ITEM-ID	
PFLG	SCRATCH	SCRATCH
QFLG	SCRATCH	SCRATCH
RFLG	FULL-FILE-RETRIEVAL FLAG	
SFLG	SELECTION ON VALUES (WITH)	
TFLG	SCRATCH	PRINT LIMITER FLAG
UFLG	SCRATCH	RESERVED
VFLG	RESERVED	SCRATCH

ENGLISH INTERFACE

WFLG	SCRATCH	RESERVED
XFLG	SCRATCH	RESERVED
YFLG	RESERVED	RESERVED
ZFLG	LEFT-JUSTIFIED ITEM-ID	
SB0	UNAVAILABLE	FIRST ENTRY FLAG, LEVEL ONE
SB1	UNAVAILABLE	FIRST ENTRY FLAG, LEVEL TWO

SB2 THROUGH SB17	SCRATCH OR RESERVED	SCRATCH OR RESEVED
VOBIT	SET FOR WRAPUP INTERFACE	
COLHDRSUPP DBLSPC HDRSUPP IDSUPP DETSUPP LPBIT TAPEFLG CBBIT PAGFRMT	SET IF THE CORRE-SPONDING CONNECTIVE WAS FOUND IN THE INPUT STATEMENT	
RMBIT	SET ON EXIT IF AN ITEM WAS RETRIEVED; ZERO ON FINAL EXIT	
WMBIT	FUNC INTERFACE	FUNC INTERFACE
GMBIT	FUNC INTERFACE	FUNC INTERFACE
BKBIT	SCRATCH	SCRATCH
DAF1	RESERVED	
DAF8	SET IF ACCESSING A DICTIONARY	
TALLIES	SELECTION PROCESSOR	LIST PROCESSOR
C1;C3-C7	SCRATCH	SCRATCH
C2	CONTENTS OF MODEID2	
CTR1-CTR4	SCRATCH	SCRATCH
CTR5	SCRATCH	AMC OF THE CURRENT ELEMENT IN THE IS
CTR6	RESERVED	SCRATCH
CTR7	RESERVED	AMC CORRESPONDING TO IR
CTR8	RESERVED	SCRATCH
CTR9	RESERVED	SCRATCH
CTR10	RESERVED	SCRATCH
CTR11	RESERVED	SCRATCH
CTR12	FUNC INTERFACE	CURRENT SUB-VALUE COUNT
CTR13	FUNC INTERFACE	CURRENT VALUE COUNT
CTR14	RESERVED	SCRATCH
CTR15	RESERVED	ITEM SIZE
CTR16	RESERVED	SCRATCH
CTR17	RESERVED	RESERVED
CTR18	RESERVED	SCRATCH
CTR19	RESERVED	SEQUENCE NO FOR BY-EXP
CTR20-CTR23	CONV INTERFACE	CONV INTERFACE
CTR24	RESERVED	SCRATCH
CTR25	RESERVED	SCRATCH
CTR26	RESERVED	SCRATCH
CTR27	RESERVED	CURRENT MAX-LENGTH
CTR28	RESERVED	SCRATCH
CTR29	RESERVED	RESERVED

OTHER STORAGE	SELECTION PROCESSOR	LIST PROCESSOR
D9	COUNT OF RETRIEVED ITEMS	
D7	FUNC INTERFACE	FUNC INTERFACE
FP1-FP5	FUNC INTERFACE	FUNC INTERFACE
RMODE	RETURN MODE-ID (MD3)	
SIZE	ITEM-SIZE	SCRATCH
SBASE	FILE BASE, MODULO,	
SMOD	AND SEPARATION	
SSEP		
DBASE	DICTIONARY BASE,	
DMOD	MODULO, AND	
DSEP	SEPARATION	

S/R'S	SELECTION PROCESSOR	LIST PROCESSOR
S1	POINTS TO THE NEXT EXPLICIT ITEM-ID	
S2-S9	SCRATCH	SCRATCH
SR0	POINTS ONE BEFORE THE ITEM COUNT FIELD	
SR1	POINTS TO THE CORRELATIVE FIELD	CURRENT CORRELATIVE FIELD
SR2	SCRATCH	SCRATCH
SR3	RESERVED	SCRATCH
SR4	POINTS TO THE LAST AM OF THE ITEM	
SR5	RESERVED	POINTS TO THE NEXT SEGMENT IN THE IS
SR6	POINTS TO THE CONVERSION FIELD	CURRENT CONVERSION FIELD
SR7	RESERVED	SCRATCH
SR8-SR12	RESERVED	RESERVED
SR13	GOSORT ONLY: NEXT SORT-KEY	RESERVED
SR14-SR19	RESERVED	RESERVED
PAGHEAD	HEADING IN THE HS IF HEADING WAS SPECIFIED	GENERATED HEADING IN THE HS
PAGFOOT	FOOTING IN THE HS IF FOOTING WAS SPECIFIED	GENERATED FOOTING IN THE HS, IF PRESENT

ENGLISH INTERFACE

A/R'S	SELECTION PROCESSOR	LIST PROCESSOR
AF	SCRATCH	SCRATCH
BMS	WITHIN THE BMS AREA	SCRATCH

CS		SCRATCH
IB		SCRATCH
OB		OUTPUT DATA LINE
IS	COMPILED STRING	COMPILED STRING
OS		SCRATCH
TS	WITHIN THE TS AREA	WITHIN THE TS AREA
UPD		WITHIN THE HS AREA
IR	WITHIN THE ITEM	WITHIN THE ITEM

WORK SPACE USAGE	SELECTION PROCESSOR	LIST PROCESSOR
AF	SCRATCH	
BMS	CONTAINS THE ITEM-ID	
CS		
IB		
OB		OUTPUT LINE
IS	COMPILED STRING	
OS		SCRATCH
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. MOST ELEMENTS USED BY THE CONV AND FUNC 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 IS 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 ITEMS ARE BUILT AS DISC-UPDATE STRINGS IN THE HISTORY STRING AREA (SEE WRAPUP DOCUMENTATION FOR FORMAT).

A USER ROUTINE CAN BE DEFINED IN THE BATCH-STRING; THE FUNCTIONAL ELEMENTS USED BY BATCH ARE DESCRIBED IN THE

ENGLISH INTERFACE

FOLLOWING TABLES; THE COLUMN HEADED "LEVEL" HAS THE FOLLOWING ENTRIES:

0	THE ELEMENT IS USED IN THE DESCRIBED FASHION THROUGHOUT THE BATCH PROCESSING
F	THE ELEMENT IS REDEFINED EVERY TIME A FILE-DEFINING ELEMENT IS FOUND
A	THE ELEMENT IS REDEFINED FOR EVERY ATTRIBUTE
BLANK	THE ELEMENT IS USED AS SCRATCH, OR IS RESERVED FOR FUTURE USAGE

AS FAR AS USER PROGRAMS ARE CONCERNED, THEREFORE, ALL ELEMENTS DEFINED AT THE "A" LEVEL CAN ALSO BE CONSIDERED SCRATCH.

EXIT CONVENTION: THE USER ROUTINE MUST RETURN TO THE BATCH PROCESSOR VIA A BRANCH INSTRUCTION TO 0,BATCH5.

BITS	LEVEL	DESCRIPTION
AFLG	0	FIRST-TIME SWITCH FOR BATCH
BFLG		RESERVED
CFLG		SCRATCH
DFLG	A	D2 ATTRIBUTE IN PROCESS
EFLG	F	UPDATES TO BE MERGED WITH THE ITEM
FFLG	0	SET WHEN A BV OR BC SUB-ELEMENT IS FOUND
GFLG		RESERVED
HFLG	A	D1 ATTRIBUTE IN PROCESS
IFLG	0	SET WHEN A "SECONDARY" FILE
JFLG		RESERVED
KFLG	F	ITEM IS TO BE VERIFIED AS EXISTING
LFLG	F	ITEM IS TO BE VERIFIED AS NOT EXISTING
MFLG	A	SET; CONV INTERFACE
NFLG		RESERVED
OFLG		RESERVED
PFLG	F	A BV OR BC SUB-ELEMENT REFERENCES A MULTI-VALUED FIELD
QFLG		RESERVED

RFLG		RESERVED
SFLG		SCRATCH
TFLG		SCRATCH
UFLG	0	ITEM IS TO BE DELETED (X ELEMENT IN THE FILE-DEFINITION)
VFLG		SCRATCH
WFLG		SCRATCH
XFLG		RESERVED
YFLG	0	PRIMARY ITEM BEING DELETED
ZFLG		SCRATCH
SB1-SB9		SCRATCH
DAF10	0	SET IF SELECT/SSELECT IS DRIVING BATCH

TALLIES	LEVEL	DESCRIPTION
C1		SCRATCH
C2		SCRATCH
C3-C9		RESERVED
CTR1-CTR3		SCRATCH
CTR4	A	D1-D2 SET NUMBER (FOLLOWS THE 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 IF UNSPECIFIED
CTR14-CTR19		RESERVED

OTHER STORAGE	LEVEL	DESCRIPTION
FP1-FP3		SCRATCH
BASE		SCRATCH
MODULO		SCRATCH
SEPAR		SCRATCH
SBASE		SCRATCH
SMOD		SCRATCH
SSEP		SCRATCH
D7		SCRATCH
D9		SCRATCH
RMODE	0	RETURN MODE-ID FOR WRAPUP

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

A/R'S AND WORK SPACES	LEVEL	DESCRIPTION
BMS	A	WORK SPACE CONTAINS THE 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 THE BATCH STRING; IS POINTS TO THE AM BEFORE THE NEXT ELEMENT
OS		SCRATCH WORK SPACE
UPD	0	POINTS TO THE HISTORY STRING

S/R'S	LEVEL	DESCRIPTION
S1-S9		SCRATCH
SR0	0	POINTS ONE BEFORE THE COUNT FIELD OF THE PRIMARY ITEM ON FILE
SR1	0	POINTS TO THE END OF THE PRIMARY ITEM ON FILE
SR2		SCRATCH
SR3		RESERVED
SR4	F	POINTS TO THE END OF THE CURRENT ITEM ON FILE
SR5		RESERVED

SR6		RESERVED
SR7	0	POINTS TO THE END OF THE OS DELETION TABLE
SR8		RESERVED
SR9	A	POINTS TO THE LAST BYTE OF VALUE IN THE BMS AREA
SR10	F	
SR11	0	POINTS TO THE END OF THE PRIMARY UPDATE STRING IF FFLG IS SET
SR12	0	POINTS ONE BEFORE "DU" IN THE HISTORY STRING FOR PRIMARY ITEM UPDATE
SR13		RESERVED
SR14	F	POINTS TO THE LOCATION OF THE FILE-DEFINING ELEMENT IN IS
SR15	F	POINTS TO THE LOCATION OF IB WHEN THE CURRENT FILE-DEFINING ELEMENT WAS FOUND
SR16-SR19		RESERVED

ALSO NOTE ELEMENTS USED BY THE CONVERSION PROCESSOR

CONVERSION PROCESSOR AND FUNCTION PROCESSOR INTERFACES

THESE PROCESSORS ARE CALLED AS SUBROUTINES, AND MAY BE USED BY USER-WRITTEN ROUTINES. FOR MORE INFORMATION, SEE THE CONV AND FUNC DOCUMENTATION.

FRWSP

FRWSP (9,TAPEIO-I)*

FUNCTIONAL DESCRIPTION

THIS ROUTINE IS USED TO FORWARD-SPACE THE TAPE BY ONE RECORD. IT DOES THIS BY SETTING R15 TO LOCATION X'1FF' IN THE PCB AND ENTERING TPREAD; FOR MORE INFORMATION, SEE THE TPREAD DOCUMENTATION.

FUNC

FUNC (0,FUNC1)*

FUNCTIONAL DESCRIPTION

THIS ROUTINE IS USED TO PROCESS "F" CONVERSIONS AND CORRELATIVES, AND IS CALLED MAINLY BY THE ENGLISH LIST AND SORT PROCESSORS. EACH CALL TO FUNC RETURNS ONE VALUE. ON THE FIRST CALL, TALLIES CTR12 AND CTR13 ARE BOTH SET TO ONE; WHEN FUNC RETURNS A VALUE, THE TERMINAL DELIMITER OF THE RETURNED STRING DETERMINES WHAT ACTION TO TAKE ON SUBSEQUENT CALLS - A VM INDICATES INCREMENT OF CTR13 BEFORE THE NEXT CALL; A SVM INDICATES INCREMENT OF CTR12; AN AM INDICATES END OF PROCESSING. FOLLOWING IS A PROGRAMMING EXAMPLE ILLUSTRATING USE OF THIS ROUTINE:

```
                .
                .
                ONE     CTR13           SET VALUE # TO ONE
FC1             ONE     CTR12           SET SUB-VALUE # 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     SVM,R15,FC2     GET NEXT SUB-VALUE
                INC     CTR13           INCREMENT VALUE COUNT
                B       FC1             GET NEXT VALUE AND
                CMNT    *               RESET SUB-VALUE COUNT
END             EQU     *               CONTINUE
```

INPUT INTERFACE

	SR1	S	POINTS TO THE FIRST CHARACTER IN THE FUNCTION CODE STRING (NORMALLY "F")
	SR0	S	POINTS ONE BEFORE THE COUNT FIELD OF THE ITEM BEING PROCESSED
	SR4	S	POINTS TO THE LAST AM OF THE ITEM BEING PROCESSED

FUNC

	TSBEG	S	POINTS 350 BYTES PRIOR TO THE AREA WHERE THE RETURNED VALUE IS TO BE STORED
	D9	D	CONTAINS THE "ITEM NUMBER" CURRENTLY BEING PROCESSED; REQUIRED ONLY FOR "NI" ELEMENTS IN THE FUNCTION CODE STRING
	CTR13	T	CONTAINS THE "VALUE NUMBER" CURRENTLY BEING PROCESSED, =1 ON INITIAL ENTRY
	CTR12	T	CONTAINS THE "SUB-VALUE NUMBER" (D2 SUB-VALUE) CURRENTLY BEING PROCESSED, =1 ON INITIAL ENTRY
	DFLG	B	+ =0 (USED BY LIST AND SORT PROCESSORS)
	XFLG	B	+

OUTPUT INTERFACE

	IR	R	POINTS ONE BEFORE THE VALUE RETURNED, AT 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, AND 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
	MFLG	B	=0 IF CONV IS CALLED

ELEMENT USAGE

	GMBIT	B	+
	WMBIT	B	+

FUNC

```
SC2         C   +
CTR1        T   +
T3          T   +
T4          T   +
T5          T   +
D7          D   +
D8          D   + UTILITY
D9          D   +
FP0         F   +
FP1         F   +
FP2         F   +
FP3         F   +
FP4         F   +
FP5         F   +
FPX         F   +
FPY         F   +

D1          D   + USED BY MBDSUB
D2          D   +
```

OTHER ELEMENTS AS USED BY CONV FOR SPECIFIED CONVERSIONS

SUBROUTINE USAGE

MBDSUB; CVDR15; CVDIS; CONV FOR EXPLICITLY SPECIFIED CONVERSIONS IN THE FUNCTION STRING; TWO INTERNAL SUBROUTINES

AT LEAST FOUR ADDITIONAL LEVELS OF SUBROUTINE LINKAGE REQUIRED; FOR EXPLICITLY SPECIFIED CONVERSIONS, ONE LEVEL REQUIRED FOR CONV, WITH ADDITIONAL LEVELS AS REQUIRED BY THE INDIVIDUAL CONVERSIONS

EXITS

TO MD994 WITH MESSAGE 701 (VALUE IN C1) AND RMODE ZERO IF A FORMAT ERROR IS FOUND IN THE FUNCTION STRING

GBMS

GBMS (3,DISKFIO-II)*

FUNCTIONAL DESCRIPTION

GBMS SETS UP THE BASE FID, MODULO, AND SEPARATION PARAMETERS OF A FILE FROM ITS FILE DEFINITION ITEM. TYPICALLY THIS ROUTINE IS CALLED AFTER A CALL TO RETIX WHICH RETRIEVES THE FILE-DEFINITION ITEM FROM THE MASTER DICTIONARY.

THE ROUTINE HANDLES BOTH 'D' AND 'Q' CODE ITEMS; A 'D' CODE ITEM (OR 'DX' OR 'DY') 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. CODES OTHER THAN 'D', 'DX', 'DY', OR 'Q' ARE NOT CONSIDERED VALID FOR FILE-DEFINITION ITEMS, AND GBMS WILL EXIT WITH RMBIT ZERO IN THESE CASES.

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, RETRIEVAL LOCK-CODES IN THE LOGON ENTRY ARE USED FOR LOCK-CODE COMPARISON; IF SET, UPDATE LOCK CODES ARE USED
IR	R	POINTS TO, OR ONE PRIOR TO THE 'D' OR 'Q' CODE IN ATTRIBUTE 1 OF THE FILE-DEFINITION ITEM
SR4	S	POINTS TO THE AM AT THE END OF THE FILE-DEFINITION ITEM
LOCKSR	S	POINTS ONE PRIOR TO THE USER'S LOCK-CODE FIELD IN HIS SYSTEM DICTIONARY ENTRY

OUTPUT INTERFACE

GBMS

RMBIT	B	SET IF BASE, MODULO, AND SEPARATION ARE SUCCESSFULLY CONVERTED; ZEROED IF THE FILE DEFINITION ITEM IS IN BAD FORMAT OR A 'Q' ITEM IS NOT FOUND
BASE	D +	CONTAIN THE BASE, MODULO, AND SEPARATION
MODULO	T +	OF THE FILE (IF RMBIT IS SET)
SEPAR	T +	
IR	R	POINTS TO THE AM FOLLOWING ATTRIBUTE 4 OF THE FILE-DEFINITION ITEM (IF ATTRIBUTE 1 IS 'D', 'DX', 'DY', OR 'Q')

THE FOLLOWING ELEMENTS ARE ALTERED ONLY IF THE FILE ACCESS-PROTECTION TEST FAILS (FILE ACCESS IS DENIED):

C1	H	=210
RMODE	D	=0

```
     PGFLG     B     =0

     HSEND     S     =HSBEG
```

ELEMENT USAGE

```
     R14       S     USED IF LOCK CODES ARE PRESENT IN THE
                     FILE-DEFINITION ITEM

     SYSR0     S   +
     SYSR1     S   + USED WITH 'Q' CODE ITEMS
     SYSR2     S   +
```

SUBROUTINE USAGE

CVDR15; GMMBMS AND RETIX FOR 'Q' CODE ITEMS

FIVE ADDITIONAL LEVELS OF SUBROUTINE LINKAGE REQUIRED FOR 'Q' CODE ITEMS; TWO ADDITIONAL LEVELS REQUIRED FOR 'D' CODE ITEMS

EXITS

GBMS

TO MD995 IF LOCK CODE COMPARISON TEST FAILS

GDLID

GDLID (13,SYSTEM-SUBS-II)*

FUNCTIONAL DESCRIPTION

THIS ROUTINE GETS THE BASE, MODULO, AND SEPARATION PARAMETERS FROM THE DL/ID ITEM IN A DICTIONARY. TYPICALLY GDLID 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   + CONTAIN THE BASE, MODULO, AND SEPARATION
     MODULO    T   + OF THE FILE WHOSE DL/ID ITEM IS TO BE
     SEPAR     T   + OBTAINED
```

OUTPUT INTERFACE

```
     RMBIT     B     SET IF THE DL/ID ITEM IS SUCCESSFULLY
                     RETRIEVED; ZEROED BY RETIX IF NO DL/ID
                     ITEM IS FOUND, OR BY GBMS IF THE ITEM IS
                     IN BAD FORMAT OR A "Q" ITEM IS NOT FOUND
```

```
BMSEND     S  +
RECORD     D  +
NNCF       H  +
FRMN       D  +
FRMP       D  + AS SET BY RETIX
NPCF       H  +
XMODE      T  +
DAF9       B  +
SIZE       T  +
SR4        S  +

IR         R  + AS SET BY GBMS IF THE DL/ID ITEM IS
R14        R  + FOUND, OTHERWISE AS SET BY RETIX

BASE       D  + AS SET BY GBMS IF THE DL/ID ITEM IS
MODULO     T  + FOUND, OTHERWISE UNCHANGED
SEPAR      T  +
```

GDLID

```
BMS        R  + =BMSBEG
SYSR1      S  +
```

ELEMENT USAGE

ELEMENTS USED BY GBMS, IF THE DL/ID ITEM IS FOUND

SUBROUTINE USAGE

RETIX AND ROUTINES CALLED BY IT; ROUTINES CALLED BY GBMS IF THE DL/ID ITEM IS FOUND

FIVE ADDITIONAL LEVELS OF SUBROUTINE LINKAGE REQUIRED IF THE DL/ID ITEM IS A "Q" POINTER, OTHERWISE FOUR ADDITIONAL LEVELS REQUIRED

GETACBMS

GETACBMS (1,LOGOFF)*

FUNCTIONAL DESCRIPTION

THIS ROUTINE RETRIEVES THE BASE, MODULO, AND SEPARATION OF THE SYSTEM ACCOUNT FILE.

INPUT INTERFACE

NONE

OUTPUT INTERFACE

BASE	D	+ CONTAIN THE BASE, MODULO, AND SEPARATION
MODULO	T	+ OF THE ACCOUNT FILE, IF FOUND
SEPAR	T	+
RMBIT	B	SET IF THE ACCOUNT FILE IS FOUND (FROM RETIX AND GBMS)
REJ1	T	=331 IF THE ACCOUNT FILE IS NOT FOUND, OR IF THE FILE-DEFINITION ITEM IN THE SYSTEM MASTER DICTIONARY IS IN BAD FORMAT, OTHERWISE UNCHANGED
IR	R	POINTS TO THE AM AFTER ATTRIBUTE 4 OF THE ACCOUNT FILE-DEFINITION ITEM (FROM GBMS)

ELEMENT USAGE

SR1	S	USED TO SAVE BMSBEG
T6	T	USED TO SAVE USER
BMS	R	USED IN CALLING RETIX

SUBROUTINE USAGE

GMMBMS, RETIX, GBMS

FOUR ADDITIONAL LEVELS OF SUBROUTINE LINKAGE REQUIRED

GETBUF

GETBUF (5,TERMIO)*

FUNCTIONAL DESCRIPTION

THIS ROUTINE ACCEPTS INPUT DATA FROM THE TERMINAL AND PERFORMS SOME EDITING ON THE CHARACTERS OBTAINED. IT ALSO PRINTS AN INITIAL PROMPT CHARACTER AT THE TERMINAL BEFORE READING INPUT. CONTROL IS RETURNED WHEN A NON-EDITING CONTROL CHARACTER IS INPUT, OR WHEN THE NUMBER OF CHARACTERS SPECIFIED IN T0 HAS BEEN INPUT AND BIT TITFLG IS ZERO (SEE BELOW).

EDITING FEATURES

CONTROL-H	LOGICALLY BACKSPACES THE BUFFER POINTER; ECHOES THE CHARACTER IN BSPCH
CONTROL-X	LOGICALLY DELETES THE ENTIRE INPUT BUFFER; ECHOES A CR/LF, AND PRINTS THE PROMPT CHARACTER IF BIT FRMTFLG IS ZERO

CONTROL-R		RETYPES THE INPUT LINE IF BIT FRMTFLG IS ZERO
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 [, /,], ↑, + AND ←	

NOTE: EXCEPT FOR SYSTEM DELIMITER CONVERSION, THE HIGH ORDER BIT OF EACH CHARACTER INPUT IS ZEROED.

INPUT INTERFACE

FRMTFLG	B	IF SET, CONTROL-X CAUSES BACKSPACES TO THE BEGINNING OF THE INPUT AREA INSTEAD OF CR/LF TO A NEW INPUT LINE; ALSO,

GETBUF

		CONTROL-R IS IGNORED
TITFLG	B	IF SET, CONTROL WILL NOT BE RETURNED WHEN THE NUMBER OF CHARACTERS SPECIFIED IN TO HAS BEEN INPUT UNLESS A NON-EDITING CONTROL CHARACTER IS ENTERED
BSPCH	C	CONTAINS THE CHARACTER TO BE ECHOED TO THE TERMINAL WHEN THE BACK SPACE KEY IS PRESSED
PRMPC	C	CHARACTER OUTPUT AS A "PROMPT" WHEN INPUT IS FIRST REQUESTED, AND AFTER CERTAIN EDITING OPERATIONS
TO	T	CONTAINS THE MAXIMUM NUMBER OF CHARACTERS TO BE ACCEPTED
R14	R	POINTS ONE BYTE BEFORE THE BEGINNING OF THE INPUT BUFFER AREA

OUTPUT INTERFACE

R15	R	POINTS TO THE CONTROL CHARACTER CAUSING RETURN TO THE CALLING ROUTINE

ELEMENT USAGE

R2;C0	C	+ SCRATCH
D0	D	+

SUBROUTINE USAGE

NONE

GETITM

GETITM (0,DISKFIO-II)*

FUNCTIONAL DESCRIPTION

THIS ROUTINE SEQUENTIALLY RETRIEVES ALL ITEMS IN A FILE. IT IS CALLED REPETITIVELY TO OBTAIN ITEMS ONE AT A TIME UNTIL ALL ITEMS HAVE BEEN RETRIEVED. THE ORDER IN WHICH THE ITEMS ARE RETURNED IS THE SAME AS THE STORAGE SEQUENCE.

IF THE ITEMS RETRIEVED ARE TO BE UPDATED BY THE CALLING ROUTINE (USING ROUTINE UPDITM), THIS SHOULD BE FLAGGED TO GETITM BY SETTING BIT DAF1. FOR UPDATING, GETITM PERFORMS A TWO-STAGE RETRIEVAL PROCESS BY FIRST STORING ALL ITEM-IDS (PER GROUP) IN A TABLE, AND THEN USING 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 THE NEXT ITEM IN THE GROUP, AS IT DOES IF THE "UPDATE" OPTION IS NOT FLAGGED.

AN INITIAL ENTRY CONDITION MUST ALSO BE FLAGGED TO GETITM BY ZEROING BIT DAF7 BEFORE THE FIRST CALL. GETITM THEN SETS UP AND MAINTAINS CERTAIN POINTERS WHICH SHOULD NOT BE ALTERED BY CALLING ROUTINES UNTIL ALL THE ITEMS IN THE FILE HAVE BEEN RETRIEVED (OR DAF7 IS ZEROED AGAIN).

NOTE THE FUNCTIONAL EQUIVALENCE OF THE OUTPUT INTERFACE ELEMENTS WITH THOSE OF RETIX.

INPUT INTERFACE

DAF7	B	INITIAL ENTRY FLAG; MUST BE ZEROED ON THE FIRST CALL TO GETITM
DAF1	B	IF SET, THE "UPDATE" OPTION IS IN EFFECT
DBASE DMOD DSEP	D T T	+ CONTAIN THE BASE, MODULO, AND SEPARATION + OF THE FILE +
BMSBEG	R	POINTS ONE PRIOR TO AN AREA WHERE THE ITEM-ID OF THE ITEM RETRIEVED ON EACH

CALL MAY BE COPIED

OVRFLCTR	D	MEANINGFUL ONLY IF DAF1 IS SET; IF NON-ZERO, THE VALUE IS USED AS THE STARTING FID OF THE OVERFLOW SPACE TABLE WHERE THE LIST OF ITEM-IDS IS STORED; IF ZERO, GETSPC IS CALLED TO OBTAIN SPACE FOR THE TABLE

OUTPUT INTERFACE

RMBIT	B	+
SIZE	T	+
R14	R	+ (SEE RETIX DOCUMENTATION)
IR	R	+
SR4	S	+
XMODE	T	+
SR0	S	=R14 IF DAF1 IS SET, OTHERWISE AS SET BY GNSEQI
BMS	R	AS SET BY RETIX IF DAF1 IS SET, OTHERWISE AS SET BY GNSEQI
BMSEND	S	=BMS IF DAF1 IS SET, OTHERWISE UNCHANGED
DAF9	B	=0

ELEMENT USAGE

BASE	D	+
MODULO	T	+
SEPAR	T	+
RECORD	D	+ USED BY GETITM AND OTHER SUBROUTINES FOR
NNCF	H	+ ACCESSING FILE DATA
FRMN	D	+
FRMP	D	+
NPCF	H	+
OVRFLW	D	USED BY GETSPC IF DAF1 IS SET AND OVRFLCTR IS INITIALLY ZERO

GETITM

THE FOLLOWING ELEMENTS SHOULD NOT BE ALTERED BY ANY OTHER ROUTINE WHILE GETITM IS USED:

DAF1	B	+ (SEE INPUT INTERFACE)
DAF7	B	+
DBASE	D	CONTAINS THE BEGINNING FID OF THE CURRENT GROUP BEING PROCESSED

DMOD	T	CONTAINS THE NUMBER OF GROUPS LEFT TO BE PROCESSED
DSEP	T	(UNCHANGED)
SBASE SMOD SSEP	D T T	+ CONTAIN THE SAVED VALUES OF DBASE, DMOD, + AND DSEP WHEN THE ROUTINE WAS FIRST + CALLED
NXTITM	S	POINTS ONE BEFORE THE NEXT ITEM-ID IN THE PRE-STORED TABLE IF DAF1 IS SET, OTHERWISE POINTS TO THE LAST AM OF THE ITEM PREVIOUSLY RETURNED
OVRFLCTR	D	CONTAINS THE STARTING FID OF THE OVERFLOW SPACE TABLE IF DAF1 IS SET, OTHERWISE UNCHANGED

SUBROUTINE USAGE

RCREC, GNSEQI; GNTBLI (LOCAL), RETIX, AND GETSPC (IF OVRFLCTR =0) IF DAF1 IS SET

BMSOVF USED WITH XMODE

FOUR ADDITIONAL LEVELS OF SUBROUTINE LINKAGE REQUIRED

ERROR CONDITIONS

SEE RETIX DOCUMENTATION ("EXITS"); GETITM, HOWEVER, CONTINUES RETRIEVING ITEMS UNTIL NO MORE ARE PRESENT EVEN AFTER THE OCCURANCE OF ERRORS.

GETOPT

GETOPT (15,SYSTEM=SUBS=I)*

FUNCTIONAL DESCRIPTION

THIS ROUTINE PROCESSES AN OPTION STRING CONSISTING OF SINGLE ALPHABETIC CHARACTERS AND/OR A NUMERIC OPTION, SEPARATED BY COMMAS.

A NUMERIC OPTION CONSISTS OF A STRING OF NUMERIC CHARACTERS OR A PAIR OF SUCH STRINGS SEPARATED BY A HYPHEN OR PERIOD. IF A NUMERIC STRING IS IMMEDIATELY PRECEDED BY A PERIOD, IT IS TREATED AS HEXADECIMAL. IF THE OPTION STRING CONTAINS MORE THAN ONE NUMERIC OPTION, THE LAST ONE WILL BE USED.

ALPHABETIC OPTIONS SET THE CORRESPONDING BITS ("A" SETS ABIT, ETC.), BUT THESE BITS ARE NOT ZEROED UPON ENTRY.

THE OPTION STRING BEGINS ONE PAST THE ADDRESS POINTED TO BY REGISTER IB, AND MUST END WITH A RIGHT PARENTHESIS (")") OR SM.

INPUT INTERFACE

IB	R	POINTS ONE BEFORE THE OPTION STRING

OUTPUT INTERFACE

ABIT	B	+
.		+
.		+ SET AS DESCRIBED ABOVE
.		+
ZBIT	B	+
NUMFLG1	B	SET IF 1 NUMERIC OPTION IS FOUND
NUMFLG2	B	SET IF 2ND NUMERIC OPTION IS FOUND
RMBIT	B	SET IF NO ERRORS ARE FOUND IN THE OPTION FORMAT, OTHERWISE UNCHANGED
D4	D	=VALUE OF THE FIRST NUMBER IN A NUMERIC OPTION, IF FOUND, OTHERWISE UNCHANGED

GETOPT

D5	D	=VALUE OF THE SECOND NUMBER IN A NUMERIC OPTION, IF FOUND; =D4 IF A NUMERIC OPTION CONSISTS OF A SINGLE NUMBER; OTHERWISE UNCHANGED
IB	R	POINTS TO THE LAST CHARACTER PROCESSED (="")" OR SM FOR A VALID OPTION STRING)

ELEMENT USAGE

NONE (EXCEPT D0 AND D1)

SUBROUTINE USAGE

CVDIB IF A DECIMAL NUMERIC OPTION IS FOUND; CVXIB IF A HEXADECIMAL NUMERIC OPTION IS FOUND

TWO ADDITIONAL LEVELS OF SUBROUTINE LINKAGE REQUIRED

GETOVF, GETSPC, ATTOVF, ATTSPC, GETBLK

```
GETOVF (1,0F1)*
GETSPC (9,0F1)*
ATTOVF (0,0F1)*
ATTSPC (10,0F1)*
GETBLK (5,0F1)*
```

FUNCTIONAL DESCRIPTION

THESE ROUTINES OBTAIN OVERFLOW FRAMES FROM THE OVERFLOW SPACE POOL MAINTAINED BY THE SYSTEM. GETBLK IS USED TO OBTAIN A BLOCK OF CONTIGUOUS SPACE (USED MAINLY BY THE CREATE-FILE PROCESSOR); THE OTHER ROUTINES OBTAIN A SINGLE FRAME.

GETOVF AND GETSPC ZERO ALL THE LINK FIELDS OF THE FRAME THEY RETURN. ATTOVF AND ATTSPC LINK THE FRAME TO THE FRAME SPECIFIED IN DOUBLE TALLY RECORD: THE FORWARD LINK FIELD OF THE FRAME SPECIFIED IN RECORD IS SET TO POINT TO THE OVERFLOW FRAME OBTAINED, THE BACKWARD LINK OF THAT FRAME IS SET TO THE VALUE OF RECORD, AND THE OTHER LINK FIELDS OF THE OVERFLOW FRAME ARE ZEROED. THE LINK FIELDS OF THE FRAME(S) OBTAINED BY GETBLK ARE NOT RESET OR INITIALIZED IN ANY WAY - THIS IS A FUNCTION OF THE CALLING PROGRAM.

THESE ROUTINES CANNOT BE INTERRUPTED UNTIL PROCESSING IS COMPLETE.

INPUT INTERFACE

D0	D	CONTAINS THE NUMBER OF FRAMES NEEDED (BLOCK SIZE), FOR GETBLK ONLY
RECORD	D	CONTAINS THE FID OF THE FRAME TO WHICH AN OVERFLOW FRAME IS TO BE LINKED (FOR ATTOVF AND ATTSPC ONLY)

OUTPUT INTERFACE

OVRFLW	D	IF THE NEEDED SPACE IS OBTAINED, THIS ELEMENT CONTAINS THE FID OF THE FRAME RETURNED (FOR GETOVF, GETSPC, ATTOVF, AND ATTSPC) OR THE FID OF THE FIRST FRAME IN THE BLOCK RETURNED (FOR GETBLK); IF THE SPACE IS UNAVAILABLE, OVRFLW=0

ELEMENT USAGE

FRMN	D		USED BY ATTOVF AND ATTSPC ONLY
D0	D	+	
D1	D	+	
R14	R	+	UTILITY
R15	R	+	
SYSR0	S	+	
D2	D	+	USED BY SYSGET
SYSR1	S	+	

SUBROUTINE USAGE

SYSGET (BUT NOT USED BY THE SINGLE-FRAME ROUTINES IF A FRAME IS OBTAINED FROM A MULTIPLE-FRAME BLOCK IN THE SYSTEM OVERFLOW TABLE); THREE INTERNAL SUBROUTINES; NOSPACE CALLED BY GETSPC AND ATTSPC IF NO FRAMES ARE AVAILABLE

TWO ADDITIONAL LEVELS OF SUBROUTINE LINKAGE REQUIRED BY GETOVF, ATTOVF, AND GETBLK; THREE LEVELS REQUIRED BY GETSPC AND ATTSPC

EXITS

FOR GETSPC AND ATTSPC: TO NSPCQ IF NO MORE FRAMES ARE AVAILABLE AND PROCESSING IS ABORTED BY THE USER; THIS IS A FUNCTION OF NOSPACE

GETUPD

GETUPD (7,DISKFIO-I)*

FUNCTIONAL DESCRIPTION

GETUPD INITIALIZES THE UPD REGISTER TRIAD TO POINT TO THE UPD WORK SPACE (FRAME PCB+28).

INPUT INTERFACE

NONE

OUTPUT INTERFACE

UPD UPDBEG	R S	* POINT TO THE FIRST DATA BYTE OF THE * FRAME 28 FRAMES AFTER THE PROCESS'S PCB
UPDEND	S	POINTS TO THE LAST BYTE OF THE ABOVE FRAME

ELEMENT USAGE

NONE

SUBROUTINE USAGE

NONE

GETUPD

GROUP LOCKS

A TABLE OF FILE GROUPS WHICH ARE LOCKED FOR UPDATE IS KEPT IN THE SYSTEM. A GROUP IS UNLOCKED WHEN AN ITEM IS UPDATED IN THAT GROUP BY THE SUBROUTINE UPDITM. THE FILE-SAVE PROCESSOR LOCKS EACH GROUP WHILE SAVING IT. BASIC AND PROC 'READ FOR UPDATE' COMMANDS USE THESE LOCKS.

GLOCK	0,GLOCK	LOCK THE GROUP WHOSE STARTING FID IS IN 'RECORD'.
GUNLOCK	1,GLOCK	UNLOCK THE GROUP WHOSE STARTING FID IS IN 'RECORD'.
GUNLOCK.LINE	2,GLOCK	UNLOCK ALL GROUPS LOCKED BY THE CALLING PROCESS.

GMMBMS

GMMBMS (4,OF1)*

FUNCTIONAL DESCRIPTION

GMMBMS SETS UP POINTERS TO THE SYSTEM DICTIONARY.

INPUT INTERFACE

NONE

OUTPUT INTERFACE

BASE	D	+ CONTAIN THE BASE, MODULO, AND SEPARATION
MODULO	T	+ OF THE SYSTEM DICTIONARY
SEPAR	T	+

ELEMENT USAGE

NONE

SUBROUTINE USAGE

NONE

GPCB0

GPCB0 (4,ABSL1)*

FUNCTIONAL DESCRIPTION

GPCB0 RETURNS THE FID OF THE PCB FOR LINE ZERO IN THE ACCUMULATOR, D0. THE 16 HIGH-ORDER BITS OF D0 ARE SET TO ZERO. NO OTHER INTERFACE OR ELEMENT USAGE IS ASSOCIATED WITH THIS ROUTINE.

ISINIT

ISINIT (2,TCL-INIT)*

FUNCTIONAL DESCRIPTION

ISINIT SIMPLY INVOKES WSINIT AND HSISOS TO INITIALIZE ALL THE PROCESS WORK SPACE POINTERS.

INPUT AND OUTPUT INTERFACES

SEE WSINIT AND HSISOS DOCUMENTATION.

ELEMENT USAGE

NONE (EXCEPT DO)

SUBROUTINE USAGE

WSINIT, HSISOS

THREE ADDITIONAL LEVELS OF SUBROUTINE LINKAGE REQUIRED

ITPIB, TPIB, OBTP, FOBTP

ITPIB (2,TAPEIO-I)*
TPIB (3,TAPEIO-I)*
OBTP (1,TAPEIO-I)*
FOBTP (0,TAPEIO-I)*

FUNCTIONAL DESCRIPTION

THESE ROUTINES ALLOW READING AND WRITING VARIABLE LENGTH RECORDS, BLOCKED IN FIXED-LENGHTH RECORDS WHOSE SIZE IS DETERMINED BY THE "T-ATT (N)" VERB. 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. THE INITIAL CALL MUST SET REGISTER OS TO BYTE ZERO OF THE TAPE BUFFER (FRAME FFFF10), AND SET TALLY T4 TO THE RECORD SIZE. 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.

INPUT INTERFACE

IBBEG	S	POINTS ONE PRIOR TO THE BUFFER AREA WHERE DEBLOCKED DATA IS TO BE COPIED, FOR TPIB
OBBEG	S	POINTS ONE PRIOR TO THE BUFFER AREA CONTAINING DATA TO BE BLOCKED, FOR OBTP
OS	R	POINTS TO THE FIRST BYTE OF THE TAPE BUFFER AREA (FRAME FFFF10), FOR OBTP
OB	R	POINTS TO THE LAST BYTE OF DATA, FOR OBTP; A SB IS PLACED ONE PAST THIS

ITPIB, TPIB, OBTP, FOBTP

LOCATION BY THE ROUTINE

T4	T	CONTAINS THE TAPE BUFFER SIZE IN BYTES, FOR OBTP

OUTPUT INTERFACE

IB	R	=IBBEG, FOR TPIB
IBEND	S	POINTS TO A SM OVERWRITING THE SB AT THE END OF THE INPUT BLOCK, FOR TPIB
OB	R	=OBBEG, FOR OBTP

ELEMENT USAGE

CMODE	T	+	
R14	R	+	UTILITY
R15	R	+	
D0	D	+	
D1	D	+	
R2;C0	T	+	AS USED BY TPREAD AND TPWRITE
CTR1	T	+	
BYTESRD	T		USED TO HOLD POSITION IN BLOCKING
SYSR0	S	+	USED BY OBTP
SYSR1	S	+	
SC0	C		USED BY TPIB AND OBTP (CONTAINS A SB ON EXIT)

ELEMENTS USED BY RDPARITY AND FRMDMP IF RDPARITY IS CALLED BY TPREAD

SUBROUTINE USAGE

TPREAD (FOR ITPIB AND TPIB) OR TPWRITE (FOR OBTP AND FOBTP); TWO INTERNAL SUBROUTINE CALLS FOR OBTP, FOBTP, AND TPIB

UP TO TEN ADDITIONAL LEVELS OF SUBROUTINE LINKAGE

ITPIB, TPIB, OBTP, FOBTP

REQUIRED FOR ITPIB AND TPIB (SEE TPREAD DOCUMENTATION); FIVE LEVELS REQUIRED BY OBTP AND FOBTP

LINK

```
LINK (9,DISKFIO-I)*
```

FUNCTIONAL DESCRIPTION

THIS ROUTINE CREATES A LINKED GROUP FROM A BLOCK OF CONTIGUOUS FRAMES. UP TO 127 FRAMES CAN BE SO LINKED. FOR EACH FRAME IN THE GROUP, THE ROUTINE SETS UP THE FIELDS SPECIFYING THE NUMBER OF NEXT CONTIGUOUS FRAMES, THE NEXT OR FORWARD LINK, THE PREVIOUS OR BACKWARD LINK, AND THE NUMBER OF PREVIOUS FRAMES.

INPUT INTERFACE

RECORD	D	CONTAINS THE FIRST FID OF THE GROUP TO BE LINKED
NNCF	H	CONTAINS ONE LESS THAN THE NUMBER OF FRAMES IN THE GROUP (MAY BE ZERO, BUT IS ALWAYS LESS THAN 127)

OUTPUT INTERFACE

R14	R	POINTS ONE PRIOR TO THE FIRST DATA BYTE OF THE FIRST FRAME IN THE GROUP
R15	R	POINTS TO THE LAST BYTE OF THE LAST FRAME IN THE GROUP
RECORD	D	CONTAINS THE FID OF THE LAST FRAME IN THE GROUP
NNCF	H	+
FRMN	D	+ CONTAIN THE VALUES OF THE LINK FIELDS OF
FRMP	D	+ THE LAST FRAME IN THE GROUP
NPCF	H	+

ELEMENT USAGE

NONE (BESIDES R14, R15, AND D0)

SUBROUTINE USAGE

NONE

LINK

SIX-BYTE BINARY TO STRING CONVERSION

MBDSUB (0,SYSTEM-SUBS-I)*
MBDNSUB (1,SYSTEM-SUBS-I)*

FUNCTIONAL DESCRIPTION

THESE ROUTINES CONVERT A BINARY NUMBER TO THE EQUIVALENT STRING OF DECIMAL ASCII CHARACTERS. MBDSUB RETURNS ONLY AS MANY CHARACTERS AS ARE NEEDED TO REPRESENT THE NUMBER, WHEREAS MBDNSUB ALWAYS RETURNS A SPECIFIED MINIMUM NUMBER OF CHARACTERS (PADDING WITH LEADING ZEROES OR BLANKS WHENEVER NECESSARY). A MINUS PRECEDES THE NUMERIC 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.

FP0 IS DESTROYED BY THE CONVERSION PROCESS.

INPUT INTERFACE

FP0	F	CONTAINS THE NUMBER TO BE CONVERTED
T4	T	CONTAINS THE MINIMUM NUMBER OF CHARACTERS TO BE RETURNED (MBDNSUB ONLY)
BKBIT	B	SET IF LEADING BLANKS WISHED FOR FILL; ZERO IF ZEROS
R15	R	POINTS ONE PRIOR TO THE AREA WHERE THE CONVERTED STRING IS TO BE STORED (UP TO 9 BYTES REQUIRED)

OUTPUT INTERFACE

R15	R	POINTS TO THE LAST CONVERTED CHARACTER
T4	T	=1 FOR MBDSUB, OTHERWISE UNCHANGED

ELEMENT USAGE

T5	T

FPY F

R14 R

SUBROUTINE USAGE

NONE

NEWPAGE

NEWPAGE (1,SYSTEM-SUBS-II)*

FUNCTIONAL DESCRIPTION

THIS ROUTINE IS USED TO SKIP TO A NEW PAGE ON THE TERMINAL OR LINE PRINTER AND PRINT A HEADING. NO ACTION IS PERFORMED, HOWEVER, IF BIT PAGINATE OR TALLY PAGSIZE IS ZERO.

INPUT INTERFACE

AS FOR WRTLIN, EXCEPT OB IS FIRST SET EQUAL TO OBBEG BY THIS ROUTINE

OUTPUT INTERFACE

SAME AS FOR WRTLIN

ELEMENT USAGE

SAME AS FOR WRTLIN

SUBROUTINE USAGE

WRTLIN AND ROUTINES CALLED BY IT, IF PAGINATE IS SET AND PAGSIZE IS GREATER THAN ZERO

ADDITIONAL SUBROUTINE LINKAGE REQUIRED ONLY IF WRTLIN IS CALLED; SEE WRTLIN DOCUMENTATION FOR THE NUMBER OF ADDITIONAL LEVELS OF LINKAGE REQUIRED, AND ADD 1

NEXTIR, NEXTOVF

NEXTIR (1,WRAPUP-II)*
NEXTOVF (3,WRAPUP-II)

FUNCTIONAL DESCRIPTION

NEXTIR OBTAINS THE FORWARD LINKED FRAME OF THE FRAME TO WHICH REGISTER IR (R6) CURRENTLY POINTS; IF THE FORWARD LINK IS ZERO, THE ROUTINE ATTEMPTS TO OBTAIN AN AVAILABLE FRAME FROM THE SYSTEM OVERFLOW SPACE POOL AND LINK IT UP APPROPRIATELY (SEE ATTOVF DOCUMENTATION). IN ADDITION, IF A FRAME IS OBTAINED, THE IR REGISTER TRIAD IS SET UP BEFORE RETURN, USING ROUTINE RDREC.

NEXTOVF MAY BE USED IN A SPECIAL WAY TO HANDLE END-OF-LINKED-FRAME CONDITIONS AUTOMATICALLY WHEN USING REGISTER IR WITH SINGLE- OR MULTIPLE-BYTE MOVE OR SCAN INSTRUCTIONS (MIID, MII, OR MCI). TALLY XMODE SHOULD BE SET TO THE MODE-ID OF NEXTOVF BEFORE THE INSTRUCTION IS EXECUTED; IF THE INSTRUCTION CAUSES IR TO REACH AN END-OF-LINKED-FRAME CONDITION (FORWARD LINK ZERO), THE SYSTEM WILL GENERATE A SUBROUTINE CALL TO NEXTOVF, WHICH WILL ATTEMPT TO OBTAIN AND LINK UP AN AVAILABLE FRAME, AND THEN RESUME EXECUTION OF THE INTERRUPTED INSTRUCTION (ASSUMING A FRAME WAS GOTTEN). IF THERE ARE NO MORE FRAMES IN THE OVERFLOW SPACE POOL, NOSPACE IS CALLED. NOTE THAT THE "INCREMENT REGISTER BY TALLY" INSTRUCTION CANNOT BE HANDLED IN THIS MANNER.

NEXTOVF IS ALSO USED BY UPDITM WITH REGISTER TS (R13). IF NEXTOVF IS ENTERED WITH TS AT AN END-OF-LINKED-FRAMES CONDITION, A BRANCH IS TAKEN TO A POINT INSIDE UPDITM. UNDER ANY OTHER CONDITION (OTHER THAN IR OR TS END-OF-LINKED-FRAME), NEXTOVF IMMEDIATELY ENTERS THE DEBUGGER.

INPUT INTERFACE

IR	R	POINTS INTO THE FRAME WHOSE FORWARD-LINKED FRAME IS TO BE OBTAINED (DISPLACEMENT UNIMPORTANT)
ACF	H	FOR NEXTOVF ONLY, MUST CONTAIN X'06' FOR IR END-OF-LINKED-FRAME HANDLING (SET

AUTOMATICALLY BY MIID, MII, AND MCI INSTRUCTIONS)

OUTPUT INTERFACE

IR IRBEG	R S	+ POINT TO THE FIRST DATA BYTE OF THE + FORWARD LINKED FRAME
IREND	S	POINTS TO THE LAST BYTE OF THE FORWARD LINKED FRAME

RECORD	D	CONTAINS THE FID OF THE FRAME TO WHICH IR POINTS
R15	R +	
NNCF	H +	
FRMN	D +	AS SET BY RDLINK FOR THE FID IN RECORD
FRMP	D +	
NPCF	H +	
OVRFLW	D	=RECORD IF ATTOVF CALLED, OTHERWISE UNCHANGED

ELEMENT USAGE

R14	R	USED BY RDLINK

ELEMENTS USED BY ATTOVF IF A FRAME IS OBTAINED FROM THE OVERFLOW SPACE POOL

SUBROUTINE USAGE

RDLINK; ATTOVF IF A FRAME MUST BE OBTAINED FROM THE OVERFLOW SPACE POOL; NOSPACE IF ATTOVF CANNOT FIND ANY MORE FRAMES

THREE ADDITIONAL LEVELS OF SUBROUTINE LINKAGE REQUIRED

EXITS

NORMALLY RETURNS VIA RDREC; POSSIBLY TO NSPCQ IF NOSPACE USED (SEE NOSPACE DOCUMENTATION); TO 5,DB1 IF ACF NOT X'06' OR X'0D' (NEXTOVF ONLY)

NEXTIR, NEXTOVF

PCRLF (7,TERMIO)*
FFDLY (9,TERMIO)*

FUNCTIONAL DESCRIPTION

PCRLF PRINTS A CARRIAGE RETURN AND LINE FEED ON THE TERMINAL AND ENTERS FFDLY, WHICH PRINTS A SPECIFIED NUMBER OF DELAY CHARACTERS (X'00').

INPUT INTERFACE

LFDLY	T	CONTAINS THE DELAY COUNT (FOR PCRLF ONLY)
T0	T	CONTAINS THE DELAY COUNT (FOR FFDLY ONLY)

OUTPUT INTERFACE

NONE

ELEMENT USAGE

R14	R

SUBROUTINE USAGE

NONE

PCRLF, FFDLY

PQ-INSERT

PQ-INSERT IS A SUBROUTINE WHICH WILL INSERT SINGLE OR MULTIPLE FIELDS INTO ANY OF THE PROC BUFFERS.

INPUT INTERFACE:

PBUF	POINTS TO FIRST BYTE OF RECEIVING FIELD. SENDING FIEL MUST BE TERMINATED BY AN ATTRIBUTE MARK. MULTIPLE FIELD INSERTS CAN BE ACCOMPLISHED BY SEPARATING THE SENDING FIELDS WITH BLANKS.
IB	POINTS TO ONE BYTE BEFORE SENDING FIELD.
SB2	0 = CAUSES BLANKS IN SENDING FIELD TO BE REPLACED BY ATTRIBUTE MARKS IN THE RECEIVING FIELD. 1 = BLANKS ARE LEFT AS BLANKS.
R9	POINTS TO PROC CONTROL FRAME (MOV PQ-REG,R9).

OUTPUT INTERFACE:

PBUF	POINT TO FIRST BYTE OF RECEIVING FIELD
AND IB	SB2 = 1. POINT ONE BYTE BEFORE RECEIVING FIELD IF SB2 = 0

INTERNAL USAGE:

BMS
R14
R15

PRINT, CRLFPRINT

```
PRINT (11,SYSTEM-SUBS-IV)*
CRLFPRINT (12,SYSTEM-SUBS-IV)*
```

FUNCTIONAL DESCRIPTION

THESE ROUTINES SEND A MESSAGE TO THE TERMINAL FROM TEXTUAL DATA IN THE CALLING PROGRAM; CRLFPRINT FIRST PRINTS A CARRIAGE RETURN AND LINE FEED. THESE ROUTINES ARE NOT COMPATIBLE WITH CONVENTIONS REGARDING THE LINE PRINTER, AND WITH THE PAGINATION ROUTINES. THE MESSAGE SENT IS A STRING OF CHARACTERS ASSEMBLED IMMEDIATELY FOLLOWING THE SUBROUTINE CALL IN THE CALLING PROGRAM. THE STRING MUST BE TERMINATED BY ONE OF THE FOUR DELIMITERS SM, AM, VM, OR SVM. CONTROL IS RETURNED TO THE INSTRUCTION AR THE LOCATION IMMEDIATELY FOLLOWING THE TERMINAL DELIMITER.

DELIMITER	ACTION
SM (X'FF') + AM (X'FE')	+ END OF MESSAGE; CR/LF PRINTED, AND RETURN
VM (X'FD')	CR/LF PRINTED, BUT MESSAGE PROCESSING CONTINUED
SVM (X'FC')	END OF MESSAGE; RETURN WITHOUT PRINTING CR/LF

INPUT INTERFACE

LFDLY	T	CONTAINS (IN THE LOW-ORDER BYTE) THE NUMBER OF "FILL" CHARACTERS (NULLS) TO BE ISSUED AFTER A CR/LF ECHO TO THE TERMINAL; REQUIRED BY PCRLF

TEXT FOLLOWING SUBROUTINE CALL IN CALLING PROGRAM

OUTPUT INTERFACE

NONE

ELEMENT USAGE

PRINT, CRLFPRINT

R14	R	+ SCRATCH
R15	R	+

SUBROUTINE USAGE

PCRLF

ONE ADDITIONAL LEVEL OF SUBROUTINE LINKAGE REQUIRED

PRIVTST1, PRIVTST2, PRIVTST3

```
PRIVTST1 (7,SYSTEM-SUBS-III)*
PRIVTST2 (5,SYSTEM-SUBS-III)*
 PRIVTST3 (4,SYSTEM-SUBS-III)
```

FUNCTIONAL DESCRIPTION

THESE ROUTINES CHECK TO SEE IF THE CALLING PROCESS HAS APPROPRIATE SYSTEM PRIVILEGE LEVELS. IF NOT, BITS PQFLG AND LISTFLAG AND TALLY RMODE ARE SET TO ZERO, THE HISTORY STRING IS SET NULL (HSEND=HSBEG), TALLY REJCTR IS SET TO 82 (AN ERROR MESSAGE NUMBER), AND AN EXIT IS TAKEN TO MD99. OTHERWISE THE ROUTINES RETURN NORMALLY.

ENTRY	BIT TESTED (ERROR IF NOT SET)
PRIVTST1	SYSPRIV1
PRIVTST2	SYSPRIV2
PRIVTST3	R0;B245

PRNTHDR, NPAGE

```
PRNTHDR (7,SYSTEM-SUBS-II)*
  NPAGE (8,SYSTEM-SUBS-II)
```

FUNCTIONAL DESCRIPTION

THESE ARE ENTRY POINTS INTO THE SYSTEM ROUTINE FOR PAGINATION AND HEADING CONTROL OF OUTPUT (ALSO USED BY WRTLIN, WT2, AND WRITOB WHEN PAGINATION IS SPECIFIED). PRNTHDR IS USED TO INITIALIZE BIT PAGINATE TO 1, AND TALLIES LINCTR AND PAGNUM TO ZERO AND ONE, RESPECTIVELY. PRNTHDR THEN FALLS IMMEDIATELY INTO NPAGE, WHICH OUTPUTS A HEADER MESSAGE.

A PAGE HEADING, IF PRESENT, MUST BE STORED IN A BUFFER DEFINED BY REGISTER PAGHEAD. THE HEADER MESSAGE IS A STRING OF DATA TERMINATED BY A SM; SYSTEM DELIMITERS IN THE MESSAGE INVOKE SPECIAL PROCESSING AS FOLLOWS:

SM (X'FF')	TERMINATES THE HEADER LINE WITH A CR/LF
AM (X'FE')	INSERTS THE CURRENT PAGE NUMBER INTO THE HEADING
VM (X'FD')	PRINTS ONE LINE OF THE HEADING AND STARTS A NEW LINE
SVM (X'FC')	SINGLY, INSERTS THE CURRENT TIME AND DATE INTO THE HEADING, BUT TWO SVM'S IN SUCCESSION INSERT THE DATE ONLY

SB (X'FB') INSERTS DATA FROM ONE OF VARIOUS BUFFERS INTO THE HEADING; IF THE CHARACTER FOLLOWING THE SB IS 'I', DATA IS COPIED FROM THE AREA BEGINNING ONE BYTE PAST THE ADDRESS SPECIFIED BY REGISTER BMSBEG; IF THE CHARACTER IS 'A', REGISTER AFBEG IS USED; FOR ANY OTHER CHARACTER, DATA IS COPIED FROM THE AREA BEGINNING THREE BYTES PAST THE ADDRESS SPECIFIED BY REGISTER ISBEG; DATA TO BE COPIED CAN BE TERMINATED BY ANY SYSTEM DELIMITER

PRNTHDR, NPAGE

CARRIAGE RETURNS, LINE FEEDS, AND FORM FEEDS SHOULD NOT BE INCLUDED IN HEADER MESSAGES, OR THE AUTOMATIC PAGINATION WILL NOT WORK PROPERLY.

INPUT INTERFACE

PAGINATE B =1 (NPAGE ONLY; SET AUTOMATICALLY BY PRNTHDR)

LINCTR T CONTAINS THE NUMBER OF THE LINE TO BE PRINTED ON THE CURRENT PAGE (NPAGE ONLY; SET TO ZERO AUTOMATICALLY BY PRNTHDR)

PAGNUM T CONTAINS THE CURRENT PAGE NUMBER (NPAGE ONLY; SET TO ONE AUTOMATICALLY BY PRNTHDR)

OTHER PARAMETERS AS FOR WT2 (SEE WRTLIN DOCUMENTATION), EXCEPT FOR PAGINATE AND PAGNUM (SEE ABOVE) AND OB (INITIALIZED TO OBBEG BY NPAGE); NOTE THAT THE OUTPUT BUFFER WHERE THE TRANSLATED HEADING MESSAGE IS BUILT (SPECIFIED BY REGISTER OBBEG) MUST BE AT LEAST TWO BYTES GREATER THAN THE LONGEST LINE OUTPUT IN THE TRANSLATED HEADING (NOT NECESSARILY THE TOTAL HEADING SIZE, IF THE ORIGINAL HEADING STRING CONTAINS ANY VMS)

OUTPUT INTERFACE

SAME AS FOR WT2

ELEMENT USAGE

SAME AS FOR WT2

SUBROUTINE USAGE

SAME AS FOR WT2

EXITS

TO WT2

PROC USER EXITS

SUMMARY

A USER-WRITTEN PROGRAM CAN GAIN CONTROL DURING EXECUTION OF A PROC BY USING THE UXXXX OR PXXXX COMMAND IN THE PROC, WHERE "XXXX" IS THE HEXADECIMAL MODE-ID OF THE USER ROUTINE. THE ROUTINE CAN PERFORM SPECIAL PROCESSING, AND THEN RETURN CONTROL TO THE PROC PROCESSOR. NECESSARILY, CERTAIN ELEMENTS USED BY THE PROC PROCESSOR MUST BE MAINTAINED BY THE USER PROGRAM; THESE ELEMENTS ARE MARKED WITH AN ASTERISK IN THE TABLE BELOW.

INPUT INTERFACE

PQFLG	B		SET, INDICATING THAT A PROC IS BEING EXECUTED
*BASE	D	+	CONTAIN THE BASE, MODULO, AND SEPARATION OF THE MASTER DICTIONARY
*MODULO	T	+	
*SEPAR	T	+	
*PQBEG	S		POINTS ONE PRIOR TO THE FIRST PROC STATEMENT; THIS WILL BE WITHIN THE FILE IN WHICH THE PROC RESIDES
*PQEND	S		POINTS TO THE TERMINAL AM OF THE PROC
PQCUR	S	+	POINT TO THE AM FOLLOWING THE UXXXX OR
IR	R	+	PXXXX STATEMENT
*PBUFBEG	S		POINTS TO A BUFFER CONTAINING THE PRIMARY AND SECONDARY INPUT BUFFERS; FORMAT IS SB ... PRIMARY INPUT ... SM SB ... SECONDARY INPUT ... SM; LOGON SETS THIS AREA TO ONE FRAME IN LENGTH, WITH ADDITIONAL FRAMES ADDED BY SUBROUTINE PQNEXTOVF AS THEY ARE REQUIRED; ADDITIONAL FRAMES ARE RELEASED BY LOGOFF
*ISBEG	S		POINTS ONE PRIOR TO THE FIRST CHARACTER OF THE PRIMARY OUTPUT BUFFER (IN THE

*STKBEG	S		POINTS ONE PRIOR TO THE FIRST CHARACTER OF THE SECONDARY OUTPUT BUFFER (STACK); THIS IS INITIALLY TWO LINKED FRAMES, THOUGH MORE FRAMES MAY BE LINKED TO IT AUTOMATICALLY BE SUBROUTINE PQNEXTOVF; ADDITIONAL FRAMES ARE RELEASED BY LOGOFF; THIS BUFFER SHOULD ALWAYS BE TERMINATED WITH A SM
IB	R		IS THE CURRENT INPUT BUFFER POINTER (MAY POINT WITHIN EITHER THE PRIMARY OR SECONDARY INPUT BUFFERS)
R9	R	+	POINT TO THE PROC CONTROL BLOCK (PCB+6);
PQ-REG	S	+	USER PROGRAMS MAY CHANGE R9, WITH THE CONSIDERATION THAT ELEMENTS DEFINED RELATIVE TO IT (SUCH AS PQ-CUR-IB) WILL NOT BE AVAILABLE
PQ-CUR-IB	B		SET IF IB POINTS INTO THE SECONDARY INPUT BUFFER, RESET OTHERWISE; THIS BIT IS DEFINED RELATIVE TO R9, SO R9 MUST BE SET TO THE PROC CONTROL BLOCK (PCB+6) IN ORDER TO REFERENCE THIS BIT
*SFLG	B		SET IF A ST ON COMMAND IS IN EFFECT
*ZFLG	B		RESET TO IDENTIFY THE PROC PROCESSOR IN CERTAIN SYSTEM SUBROUTINES
*SC2	C		CONTAINS A BLANK

		SFLG ON	SFLG OFF
IS	R	POINTS TO THE LAST BYTE MOVED INTO THE SECONDARY OUTPUT BUFFER	POINTS TO THE LAST BYTE MOVED INTO THE PRIMARY OUTPUT BUFFER
UPD	R	POINTS TO THE LAST	POINTS TO THE LAST

PROC USER EXITS

		BYTE MOVED INTO THE PRIMARY OUTPUT BUFFER	BYTE MOVED INTO THE SECONDARY OUTPUT BUFFER

OUTPUT INTERFACE

IR	R		POINTS TO THE AM PRECEDING THE NEXT PROC STATEMENT TO BE EXECUTED; MAY BE ALTERED TO CHANGE PROC EXECUTION
IS	R	+	MAY BE ALTERED AS NEEDED TO ALTER DATA
UPD	R	+	WITHIN THE INPUT AND OUTPUT BUFFERS, BUT
IB	R	+	THE FORMATS DESCRIBED MUST BE MAINTAINED

PROC BUFFERS			EACH MUST BE TERMINATED WITH A SM; AM'S MAY SEPARATE PARAMETERS
SFLG	B		SET IF "STACK ON" IS IN EFFECT, RESET OTHERWISE
PQ-CUR-IB	B		SET IF SECONDARY INPUT BUFFER IS ACTIVE, RESET OTHERWISE
PQFLG	B		SET IF PROC EXECUTION IS TO CONTINUE, RESET OTHERWISE
ABIT-ZBIT	B		ZERO
BASE	D	+	
MODULO	T	+	
SEPAR	T	+	
SC0	C	+	
SC1	C	+	
SC2	C	+	SET TO VALUE ON ENTRY
PQBEG	S	+	
PQEND	S	+	
PBUFBEG	S	+	
ISBEG	S	+	
STKBEG	S	+	

EXIT CONVENTION

THE NORMAL METHOD OF RETURNING CONTROL TO THE PROC PROCESSOR

PROC USER EXITS

IS TO EXECUTE AN EXTERNAL BRANCH INSTRUCTION (ENT) TO 2,PROC-I. IF IT IS NECESSARY TO ABORT PROC CONTROL AND EXIT TO WRAPUP, BIT PQFLG SHOULD BE RESET BEFORE BRANCHING TO ANY OF THE WRAPUP ENTRY POINTS (SEE WRAPUP DOCUMENTATION).

NOTE THAT WHEN A PROC EVENTUALLY TRANSFERS CONTROL TO TCL (VIA THE "P" OPERATOR), CERTAIN ELEMENTS ARE EXPECTED TO BE IN AN INITIAL CONDITION. THEREFORE, IF A USER ROUTINE USES THESE ELEMENTS, THEY SHOULD BE RESET BEFORE RETURNING TO THE PROC, UNLESS THE ELEMENTS ARE DELIBERATELY SET UP AS A MEANS OF PASSING PARAMETERS TO OTHER PROCESSORS. SPECIFICALLY, THE BITS AFLG THROUGH ZFLG 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. ALSO, THE SCAN CHARACTER REGISTERS SC0, SC1, AND SC2 MUST CONTAIN A SB, A BLANK, AND A BLANK, RESPECTIVELY.

PQNEXTOVF

SUBROUTINE PQNEXTOVF MAY BE USED BY USER-WRITTEN PROGRAMS WHEN ADDING PARAMETERS TO THE PRIMARY OR SECONDARY INPUT BUFFERS OR THE SECONDARY OUTPUT BUFFER. THIS ROUTINE WILL AUTOMATICALLY ATTACH ADDITIONAL FRAMES TO THESE BUFFERS AS REQUIRED. IT SHOULD BE USED BY SETTING XMODE TO ITS MODE-ID, AND WILL THEN BE ENTERED ON A "FORWARD LINK ZERO" CONDITION ON REGISTER 8 OR 15.

PQNEXTOVF INPUT INTERFACE

R8 OR R15	R	POINTS TO THE END OF THE FRAME ON WHICH THE FORWARD LINK ZERO TRAP OCCURS
R9	R	POINTS TO THE PROC CONTROL BLOCK

PQNEXTOVF OUTPUT INTERFACE

R8 OR R15	R	UNCHANGED

OTHER ELEMENTS AS FROM ATTOVF; IN PARTICULAR, OVRFLW=0 IF NO FRAME IS AVAILABLE

PRTERR

PRTERR (0,WRAPUP=III)*

FUNCTIONAL DESCRIPTION

PRTERR IS USED TO RETRIEVE AND PRINT A MESSAGE FROM THE SYSTEM FILE ERRMSG. A PARAMETER STRING MAY BE PASSED TO THE ROUTINE, IN WHICH CASE THE PARAMETERS ARE FORMATTED AND INSERTED ACCORDING TO THE CODES IN THE MESSAGE ITEM.

ITEMS IN THE ERRMSG FILE CONSIST OF AN ARBITRARY NUMBER OF LINES (WHERE A LINE IS DELIMITED BY AN AM), WITH EACH LINE CONTAINING A CODE LETTER IN COLUMN ONE, POSSIBLY FOLLOWED BY A STRING OR NUMERIC PARAMETER (NUMERIC PARAMETERS ENCLOSED IN PARENTHESES). THE POSSIBLE CODES AND THEIR MEANINGS ARE LISTED BELOW. (BRACKETS INDICATE OPTIONAL PARAMETERS.)

A [(DEC. #)]	PARAMETER INSERTION CODE; THE NEXT PARAMETER FROM THE PARAMETER STRING, IF ANY, IS PLACED INTO THE OUPUT BUFFER; IF "DEC. #" IS SPECIFIED, THE PARAMETER IS LEFT-JUSTIFIED IN A BLANK FIELD OF THAT LENGTH
D	THE SYSTEM DATE IN DD MMM YYYY FORMAT IS ADDED TO THE OUTPUT BUFFER
E [STRING]	THE MESSAGE ITEM-ID, SURROUNDED BY BRACKETS, IS PLACED INTO THE OUTPUT BUFFER, FOLLOWED BY "STRING" IF PRESENT

H STRING		THE CHARACTER STRING IS PLACED IN THE OUTPUT BUFFER (NO BLANK IS NECESSARY BETWEEN THE CODE LETTER AND THE BEGINNING OF THE STRING)
L [(DEC. #)]		THE OUTPUT BUFFER IS PRINTED, AND THE SPECIFIED NUMBER OF LINE FEEDS IS OUTPUT (ONE IF "DEC. #" IS NOT SPECIFIED)
R [(DEC. #)]		LIKE A, ONLY THE PARAMETER IS RIGHT-JUSTIFIED, IN A FIELD OF "DEC. #" BLANKS IF "DEC. #" IS SPECIFIED

PRTERR

S [(DEC. #)]		THE POINTER TO THE CURRENT POSITION IN THE OUTPUT BUFFER IS REPOSITIONED TO THE SPECIFIED COLUMN (COLUMN ONE IF "DEC. #" IS NOT PRESENT)
T		THE SYSTEM TIME IN HH:MM:SS IS ADDED TO THE OUTPUT BUFFER
X		THE NEXT PARAMETER IS SKIPPED

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 END WITH A SM
EBASE	D	+ USED AS THE BASE, MODULO, AND SEPARATION
EMOD	T	+ FOR THE MESSAGE FILE IF EBASE IS
ESEP	T	+ NON-ZERO; IF EBASE IS ZERO, PRTERR ATTEMPTS TO SET EBASE, EMOD, AND ESEP TO THE PARAMETERS FOR THE SYSTEM FILE ERRMSG, AND EXITS ABNORMALLY IF UNABLE TO DO SO
MBASE	D	+ USED AS THE PARAMETERS FOR THE MASTER
MMOD	T	+ DICTIONARY IF NECESSARY TO SET UP EBASE,
MSEP	T	+ EMOD, AND ESEP, BUT PRTERR EXITS ABNORMALLY IF MBASE IS ZERO
OBSIZE	T	CONTAINS THE MAXIMUM NUMBER OF CHARACTERS TO BE OUTPUT ON A LINE (NORMALLY SET AT LOGON TIME)

OBBEG	S	+	POINT TO THE BEGINNING AND END OF THE
OBEND	S	+	OUTPUT BUFFER (NORMALLY SET AT LOGON TIME)

OTHER ELEMENTS AS REQUIRED BY WRTLIN (SEE WRTLIN DOCUMENTATION)

OUTPUT INTERFACE

PRTERR

TS	R		POINTS TO THE AM AFTER THE MESSAGE ITEM-ID IF NO PARAMETERS ARE PROCESSED, OTHERWISE TO THE AM OR SM AFTER THE LAST PARAMETER PROCESSED
EBASE	D	+	CONTAIN THE BASE, MODULO, AND SEPARATION
EMOD	T	+	PARAMETERS FOR THE SYSTEM FILE ERRMSG IF
ESEP	T	+	EBASE WAS ORIGINALLY ZERO (AND THE FILE WAS SUCCESSFULLY RETRIEVED)
LINCTR	T	+	UPDATED IF BIT PAGINATE IS SET
PAGNUM	T	+	

ELEMENT USAGE

SB60	B	+	
SB61	B	+	
CTR0	T	+	
T6	T	+	
BASE	D	+	
MODULO	T	+	
SEPAR	T	+	UTILITY
AF	R	+	
IR	R	+	
BMS	R	+	
BMSBEG	S	+	
OB	R	+	
R14	R	+	
SR4	S	+	
CTR1	T		USED WITH "R" CODE MESSAGES
SYSR1	S		USED WITH "S" CODE MESSAGES
INHIBITH	H		INCREMENTED DURING RETRIEVAL OF FILE ERRMSG IF EBASE IS ORIGINALLY ZERO, AND DECREMENTED AFTERWARDS

ALL ELEMENTS USED BY RETIX, AND BY WRTLIN (UNLESS PRTERR EXITS ABNORMALLY), AND ELEMENTS USED BY GBMS IF PRTERR ATTEMPTS RETRIEVAL OF THE SYSTEM FILE ERRMSG

PRTERR

SUBROUTINE USAGE

RETIX, WRTLIN, DECINHIB, DATE (FOR "D" CODE MESSAGES), TIME (FOR "T" CODE MESSAGES), GBMS (FOR RETRIEVING ERRMSG)

SIX ADDITIONAL LEVELS OF SUBROUTINE LINKAGE REQUIRED IF GBMS ATTEMPTS RETRIEVAL OF AN ERRMSG FILE WHICH IS A "Q" CODE ITEM, OTHERWISE FOUR LEVELS REQUIRED

EXITS

TO 2,ABSL IF EBASE AND MBASE ARE BOTH ZERO

R.ETA, R.ETA.M, R.ATE, R.ETA.M

```
 R.ETA (1,R.FIO-VI)*
R.ETA.M (2,R.FIO-VI)*
 R.ATE (3,R.FIO-IV)*
R.ATE.M (4,R.FIO-IV)*
```

FUNCTIONAL DESCRIPTION

THESE ROUTINES PERFORM TRANSLATION OF CHARACTER STRINGS FROM EBCDIC OR ASCII TO ASCII OR EBCDIC. EBCDIC CHARACTERS WITH NO ASCII EQUIVALENT ARE TREATED AS BINARY OR PACKED DATA; R.ETA AND R.ETA.M TRANSLATE THESE CHARACTERS TO VALUES HAVING THE HIGH-ORDER BIT SET, AND R.ATE AND R.ATE.M TRANSLATE THESE VALUES TO THE APPROPRIATE EBCDIC CHARACTERS. R.ETA AND R.ATE OVERLAY THE INPUT STRING WITH THE TRANSLATED STRING, WHILE THE OTHER TWO ROUTINES STORE THE TRANSLATED STRING IN A SPECIFIED BUFFER AREA.

INPUT INTERFACE

CTR1	T	CONTAINS THE NUMBER OF CHARACTERS TO BE TRANSLATED
R8	R	POINTS TO THE FIRST CHARACTER OF THE ASCII STRING BUFFER, FOR "MOVE" ROUTINES ONLY (R.ETA.M AND R.ATE.M)
R15	R	POINTS TO THE FIRST CHARACTER OF THE EBCDIC STRING BUFFER; FOR NON-"MOVE" ROUTINES, THIS IS ALSO THE ASCII STRING BUFFER

OUTPUT INTERFACE

CTR1	T	=0
R8	R	+ POINT TO THE LAST CHARACTERS IN THEIR
R15	R	+ RESPECTIVE BUFFERS; FOR NON-"MOVE" ROUTINES, R8=R15

ELEMENT USAGE

T0	T	+
R13	R	+ UTILITY

R.ETA, R.ETA.M, R.ATE, R.ETA.M

R14	R	+

SUBROUTINE USAGE

NONE

RDLABEL, RDLABELY

```
RDLABEL (2,TAPEIO-II)*
RDLABELY (8,TAPEIO-II)
```

FUNCTIONAL DESCRIPTION

THESE ROUTINES READ MAGNETIC TAPE LABELS AND STORE THEM IN THE QUATERNARY CONTROL BLOCK (PCB+3). TAPE LABELS HAVE THE FOLLOWING FORMAT:

SM L ... LABEL DATA ... VM TIME DATE AM REEL # AM SM

DATA IS STORED IN PCB+3 AS FOLLOWS:

HEX BYTE DISPLACEMENT	TYPE	DESCRIPTION
197 (BIT 0)	B	"UNLABELED TAPES IN USE" FLAG
198	T	REEL NUMBER
19A	T	RECORD SIZE SAVE AREA
19C	-	LABEL SAVE BUFFER (44 BYTES)
1CE	-	LABEL READ/WRITE BUFFER (44 BYTES)

SINCE THE TAPE I/O ROUTINES ARE NON-REENTRANT, INTERNAL STORAGE IS UTILIZED WHEN AN EOT CONDITION IS HANDLED BY THE TAPE READ OR WRITE SUBROUTINES. THESE ROUTINES SAVE REGISTERS R13, R14, AND R15 IN INTERNAL SAVE AREAS (DEFINED IN TAPEIO-II), AND SET UP R13 TO DISPLACEMENT X'196' IN THE QUATERNARY CONTROL BLOCK IN ORDER TO ADDRESS ELEMENTS IN THAT BLOCK. R13, R14, AND R15 ARE RESTORED ON EXIT.

RDLABEL 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. RDLABELY IS SIMILAR TO RDLABEL, EXCEPT THAT THE REEL NUMBER IS SPECIFIED IN TALLY CTR1.

INPUT INTERFACE

CTR1	T	CONTAINS THE REEL NUMBER, FOR RDLABELY

RDLABEL, RDLABELY

ONLY

OUTPUT INTERFACE

THE LABEL SAVE AREA IS SET UP AS DESCRIBED

ELEMENT USAGE

R13	R	+
R14	R	+ UTILITY
R15	R	+
D0	D	+
D1	D	+ AS USED BY TPREAD
R2;C0	C	+
T4	T	+

SUBROUTINE USAGE

TPREAD; BCKSP (FOR UNLABELED TAPES); CVDR15 (FOR LABELED TAPES); CRLFPRINT (FOR ERROR MESSAGES); ONE INTERNAL SUBROUTINE

MAXIMUM TEN ADDITIONAL LEVELS OF SUBROUTINE LINKAGE REQUIRED (SEE TPREAD DOCUMENTATION)

ERROR CONDITIONS

SEE TPREAD DOCUMENTATION

RDLINK, WTLINK

```
RDLINK (4,DISKFIO-I)*
WTLINK (6,DISKFIO-I)*
```

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	CONTAINS THE FID OF THE FRAME WHOSE LINKS ARE TO READ OR WRITTEN
NNCF	H	CONTAINS THE NUMBER OF NEXT CONTIGUOUS FRAMES
FRMN	D	CONTAINS THE FID OF THE NEXT OR FORWARD LINKED FRAME
FRMP	D	CONTAINS THE FID OF THE PREVIOUS OR BACKWARD LINKED FRAME
NPCF	H	CONTAINS THE NUMBER OF PREVIOUS CONTIGUOUS FRAMES
R15	R	POINTS TO BYTE ZERO OF THE FRAME

ELEMENT USAGE

R14	R	SCRATCH

SUBROUTINE USAGE

NONE

RDREC

RDREC (3,DISKFIO-I)*

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, AND FRMP.

INPUT INTERFACE

RECORD	D	CONTAINS THE FID REQUIRED

OUTPUT INTERFACE

IR	R	+	POINT ONE PRIOR TO THE FIRST DATA BYTE
IRBEG	S	+	OF THE FRAME
IREND	S		POINTS TO THE LAST DATA BYTE OF THE FRAME
R15	R	+	
NNCF	H	+	
FRMN	D	+	(SEE RDLINK/WTLINK DOCUMENTATION)
FRMP	D	+	
NPCF	H	+	

ELEMENT USAGE

NONE

SUBROUTINE USAGE

NONE

READLIN, READLINX, READIB

```
READLIN (0,TERMIO)*
READLINX (6,TERMIO)*
 READIB (8,TERMIO)*
```

FUNCTIONAL DESCRIPTION

THESE 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 HAS BEEN INPUT, A SM WILL BE ADDED AT THE END OF THE LINE.

THESE ROUTINES CALL GETBUF TO READ INPUT DATA FROM THE TERMINAL, AND THEN DETERMINE IF THE LAST CHARACTER WAS A CARRIAGE RETURN OR LINE FEED. IF THE LAST CHARACTER WAS A CONTROL CHARACTER (SEE GETBUF DOCUMENTATION), THESE ROUTINES EITHER ACCEPT OR DELETE THE CHARACTER, DEPENDING ON THE VALUE OF BIT CCDEL, AND CALL GETBUF AGAIN. READLIN AND READLINX ALSO ECHO A CR/LF AT THE END OF THE INPUT LINE.

THE ENTRIES READLIN AND READIB ALSO PROVIDE THE FACILITY FOR TAKING INPUT FROM A STACK INSTEAD OF DIRECTLY FROM THE TERMINAL (SEE BELOW). THIS FEATURE IS USED, FOR EXAMPLE, BY THE PROC PROCESSOR TO STORE INPUT LINES WHICH ARE RETURNED TO REQUESTING PROCESSORS AS IF THEY ORIGINATED AT THE TERMINAL. IF THE LAST CHARACTER IN A STACKED LINE IS A "<", IT IS REPLACED WITH A SM. TERMINAL INPUT RESUMES WHEN THE STACKED INPUT IS EXHAUSTED. READLINX DOES NOT TEST FOR STACKED INPUT.

TAB CHARACTERS (CONTROL-I, X'09') WILL BE PROCESSED IF BIT ITABFLG IS SET. THE INPUT TAB TABLE IS IN THE QUADRENARY CONTROL BLOCK, STARTING AT BYTE 64. UP TO ELEVEN VALUES MAY BE STORED IN TALLIES BEGINNING AT THIS LOCATION, WITH VALUES IN INCREASING ORDER OF COLUMN POSITION. VALID TAB VALUES CAUSE AN APPORPRIATE NUMBER OF BLANKS TO BE OUTPUT TO THE TERMINAL IN ORDER TO POSITION THE CURSOR.

READLIN, READLINX, READIB

INPUT INTERFACE

ITABFLG	B	IF SET, TAB CHARACTERS ARE PROCESSED
CCDEL	B	IF SET, CONTROL CHARACTERS ARE DELETED FROM TERMINAL INPUT
IBBEG	S	POINTS ONE BYTE BEFORE THE BUFFER AREA WHERE INPUT IS TO BE STORED; THE BUFFER MUST BE TWO BYTES GREATER THAN IBSIZE
IBSIZE	T	CONTAINS THE MAXIMUM NUMBER OF CHARACTERS ACCEPTED FOR INPUT
LFDLY	T	CONTAINS (IN THE LOW-ORDER BYTE) THE NUMBER OF "FILL" CHARACTERS (NULLS) TO BE ISSUED AFTER A CR/LF ECHO TO THE TERMINAL (FOR READLIN AND READLINX ONLY)
FRMTFLG	B	IF SET, CONTROL-X CAUSES BACKSPACES TO THE BEGINNING OF THE INPUT AREA INSTEAD OF CR/LF TO A NEW INPUT LINE; ALSO, CONTROL-R IS IGNORED; REQUIRED BY GETBUF
TITFLG	B	IF SET, CONTROL WILL NOT BE RETURNED IF THE MAXIMUM NUMBER OF CHARACTERS IS INPUT UNLESS A NON-EDITING CONTROL CHARACTER IS ENTERED (E.G. CARRIAGE RETURN); REQUIRED BY GETBUF
PRMPC	C	TERMINAL PROMPT CHARACTER; REQUIRED BY GETBUF
BSPCH	C	CONTAINS THE CHARACTER TO BE ECHOED TO THE TERMINAL WHEN THE BACK SPACE KEY IS PRESSED; REQUIRED BY GETBUF

STKFLG	B	IF SET, GETIB TESTS FOR "STACKED" INPUT; TERMINAL INPUT WILL NOT BE REQUESTED UNTIL STACKED INPUT IS EXHAUSTED
STKINP	S	POINTS TO THE NEXT "STACKED" INPUT LINE; LINES ARE DELIMINATED BY AM'S, WITH A SM

READLIN, READLINX, READIB

INDICATING THE END OF THE STACK

OUTPUT INTERFACE

IB	R	=IBBEG
IBEND	S	POINTS TO A SM ONE BYTE PAST THE END OF INPUT DATA (OVERWRITES THE CR OR LF)
STKFLG	B	ZEROED IF THE END OF STACKED INPUT WAS REACHED; NOT CHANGED IF INITIALLY ZERO
STKINP	S	POINTS TO THE NEXT LINE OF STACKED INPUT (OR END OF STACK) IF STACKED INPUT IS BEING PROCESSED

ELEMENT USAGE

R2;C0	C	+
D0	D	+ UTILITY
R14	R	+
R15	R	+

SYSR0	S	USED IF ITABFLG SET AND TAB CHARACTERS PROCESSED

SUBROUTINE USAGE

IF NO STACKED INPUT: GETBUF, PCRLF (EXCEPT FOR READLINX)

TWO ADDITIONAL LEVELS OF SUBROUTINE LINKAGE REQUIRED IF TAB CHARACTERS PROCESSED, OTHERWISE ONE LEVEL REQUIRED

ERROR CONDITIONS

IF A STACKED INPUT LINE EXCEEDS IBSIZE, THE LINE IS TRUNCATED AT IBSIZE; THE REMAINDER OF THE LINE IS LOST.

RELOVF, RELBLK, RELCHN

```
RELOVF (2,0F1)*
RELBLK (6,0F1)*
RELCHN (3,0F1)*
```

FUNCTIONAL DESCRIPTION

THESE ROUTINES ARE USED TO RELEASE FRAMES TO THE OVERFLOW SPACE POOL. RELOVF IS USED TO RELEASE A SINGLE FRAME, RELBLK IS USED TO RELEASE A BLOCK OF CONTIGUOUS FRAMES, AND RELCHN IS USED TO RELEASE A CHAIN OF LINKED FRAMES (WHICH MAY OR MAY NOT BE CONTIGUOUS). 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 (FOR RELOVF), OR THE FIRST FID OF THE BLOCK OR CHAIN TO BE RELEASED (FOR RELBLK AND RELCHN, RESPECTIVELY)
D0	D	CONTAINS THE NUMBER OF FRAMES (BLOCK SIZE) TO BE RELEASED, FOR RELBLK ONLY

OUTPUT INTERFACE

NONE

ELEMENT USAGE

OVRFLW	D	+	
R14	R	+	UTILITY
R15	R	+	
D0	D	+	
D1	D	+	USED BY SYSREL
D2	D	+	

SUBROUTINE USAGE

SYSREL; TWO INTERNAL SUBROUTINES

RETIX, RETI, RETIXX, RETIXU

```
RETIX (1,DISKFIO-I)*
RETI (0,DISKFIO-I)*
RETIXX (12,DISKFIO-I)*
RETIXU (11,DISKFIO-I)*
```

RELOVF, RELBLK, RELCHN

TWO ADDITIONAL LEVELS OF SUBROUTINE LINKAGE REQUIRED

FUNCTIONAL DESCRIPTION

THESE ARE THE ENTRY POINTS TO THE STANDARD SYSTEM ROUTINE FOR RETRIEVING AN ITEM FROM A FILE. THE ITEM-ID IS EXPLICITLY SPECIFIED TO THE ROUTINE, AS ARE THE FILE PARAMETERS BASE, MODULO, AND SEPARATION. ADDITIONALLY, THE NUMBER OF THE FIRST FRAME IN THE GROUP IN WHICH THE ITEM MAY BE STORED MUST BE SPECIFIED IF THE ENTRY RETIXX IS USED. THE OTHER ENTRIES PERFORM A "HASHING" ALGORITHM TO DETERMINE THE GROUP (SEE HASH DOCUMENTATION). THE GROUP IS SEARCHED SEQUENTIALLY FOR A MATCHING ITEM-ID. IF THE ROUTINE FINDS A MATCH, IT RETURNS POINTERS TO THE BEGINNING AND END OF THE ITEM, AND THE ITEM SIZE (FROM THE ITEM COUNT FIELD). IF ENTRY RETIXU IS USED, THE GROUP IS LOCKED DURING PROCESSING, PREVENTING OTHER PROGRAMS FROM ACCESSING (AND POSSIBLY CHANGING) THE DATA.

THE ITEM-ID IS SPECIFIED IN A BUFFER DEFINED BY REGISTER BMSBEG; IF ENTRY RETI IS USED, REGISTER BMS MUST POINT TO THE LAST BYTE OF THE ITEM-ID, AND AN AM WILL BE APPENDED TO IT BY THE ROUTINE. FOR ALL OTHER ENTRY POINTS, THE ITEM-ID MUST ALREADY BE TERMINATED BY AN AM.

INPUT INTERFACE

BMSBEG	S		POINTS ONE BYTE BEFORE THE ITEM-ID
BMS	R		POINTS TO THE LAST CHARACTER OF THE ITEM-ID, FOR RETI AND RETIXX ONLY
BASE	D	+	CONTAIN THE BASE, MODULO, AND SEPARATION
MODULO	T	+	OF THE FILE TO BE SEARCHED
SEPAR	T	+	
RECORD	D		CONTAINS THE BEGINNING FID OF THE GROUP TO BE SEARCHED, FOR RETIXX ONLY

OUTPUT INTERFACE

RETIX, RETI, RETIXX, RETIXU

BMS	R	+	POINT TO THE LAST CHARACTER OF THE
BMSEND	S	+	ITEM-ID
RECORD	D		CONTAINS THE BEGINNING FID OF THE GROUP TO WHICH THE ITEM-ID HASHES (SET IF HASH IS CALLED)
NNCF	H	+	
FRMN	D	+	CONTAIN THE LINK FIELDS OF THE FRAME
FRMP	D	+	SPECIFIED IN RECORD; SET BY RDREC
NPCF	H	+	

XMODE	T	=0	
		ITEM FOUND:	ITEM NOT FOUND:
RMBIT	B	=1	=0
SIZE	T	=VALUE OF ITEM COUNT FIELD	=0
R14	R	POINTS ONE PRIOR TO THE ITEM COUNT FIELD	POINTS TO THE LAST AM OF THE LAST ITEM IN THE GROUP
IR	R	POINTS TO THE FIRST AM OF THE ITEM	POINTS TO THE AM INDICATING END OF GROUP DATA (=R14+1)
SR4	S	POINTS TO THE LAST AM OF THE ITEM	=R14

ELEMENT USAGE

NONE (EXCEPT D0, D1, AND R15)

SUBROUTINE USAGE

RDREC (LOCAL), HASH (EXCEPT FOR RETIXX; LOCAL), GLOCK (RETIXU ONLY), IROVF (FOR IR OVERFLOW SPACE HANDLING AND ERROR CONDITIONS)

RETIX, RETI, RETIXX, RETIXU

THREE ADDITIONAL LEVELS OF SUBROUTINE LINKAGE REQUIRED (FOR IROVF AND GLOCK; RDREC AND HASH REQUIRE ONE LEVEL)

EXITS

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 XXXXXX

IS RETURNED (WHERE XXXXXX IS THE 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.

REWIND

REWIND (8,TAPEIO-I)*

FUNCTIONAL DESCRIPTION

THIS ROUTINE REWINDS THE TAPE UNIT. IT CALLS INIT AND REQUIRES FOUR ADDITIONAL LEVELS OF SUBROUTINE LINKAGE.

SETPIB

SETPIB (3,LOGON)*

FUNCTIONAL DESCRIPTION

SETPIB SETS R14 POINTING TO THE PIB OF THE CALLING PROCESS.

INPUT INTERFACE

NONE

OUTPUT INTERFACE

R14	R	POINTS TO THE FIRST BYTE OF THE PROCESS'S PIB

ELEMENT USAGE

NONE

SUBROUTINE USAGE

NONE

SETPIBF

SETPIBF (3,ABSL1)*

FUNCTIONAL DESCRIPTION

SETPIBF SETS R15 POINTING TO THE PIB ASSOCIATED WITH LINE ZERO. NO OTHER INTERFACE OR ELEMENT USAGE IS REQUIRED BY THIS ROUTINE.

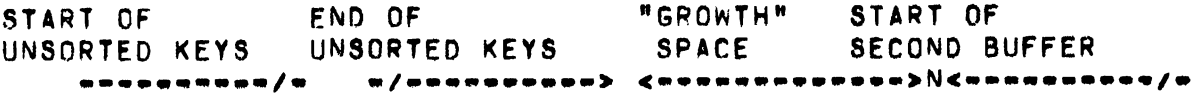
SORT

SORT (1,SORT)*

FUNCTIONAL DESCRIPTION

THIS ROUTINE 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. SORT AUTOMATICALLY OBTAINS AND LINKS OVERFLOW SPACE WHENEVER 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 IS 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 THE SM PRECEDING THE FIRST KEY
SR2	S	POINTS TO THE SM TERMINATING THE LAST KEY

SORT

SR3	S	POINTS TO THE BEGINNING OF THE SECOND BUFFER

OUTPUT INTERFACE

SR1	S	POINTS BEFORE THE SB PRECEDING THE FIRST SORTED KEY (THE EXACT OFFSET VARIES FROM CASE TO CASE); THE END OF THE SORTED KEYS (SEPARATED BY SB'S) IS MARKED BY A SM

ELEMENT USAGE

SB1	B	+	
SC2	C	+	
XMODE	T	+	
FP0	F	+	
FP1	F	+	
IS	R	+	
OS	R	+	
BMS	R	+	
TS	R	+	
CS	R	+	UTILITY
R14	R	+	
R15	R	+	
S1	S	+	
S2	S	+	
S3	S	+	
S5	S	+	
S7	S	+	
S8	S	+	
S9	S	+	

SUBROUTINE USAGE

COMP (0,SORT)

GWS USED WITH XMODE

FOUR ADDITIONAL LEVELS OF SUBROUTINE LINKAGE REQUIRED

TCL-I

MD1 (0,TCL-I)*
MD1B (2,TCL-I)*

FUNCTIONAL DESCRIPTION (MD1)

MD1 IS THE BASIC ENTRY POINT (NOT A SUBROUTINE) FOR THE TERMINAL CONTROL LANGUAGE (TCL) PROCESSOR. WHEN THIS ENTRY POINT IS USED, TCL CHECKS FOR PROC CONTROL, AND IF SO, ENTERS THE PROC PROCESSOR. IF A PROC IS NOT IN CONTROL (AND BIT CHAINFLG IS ZERO), AN INPUT LINE IS OBTAINED FROM THE TERMINAL, AND CONTROL PASSES IMMEDIATELY TO MD1B.

INPUT INTERFACE (MD1)

CHAINFLG	B	IF SET, TERMINAL INPUT IS NOT OBTAINED (AS WHEN CHAINING FROM ONE DATA/BASIC PROGRAM TO ANOTHER)
PGFLG	B	SET TO INDICATE PROC CONTROL

```
OUTPUT INTERFACE (MD1)

     AFLG       B   +
      .             +
      .             +
      .             + =0
     AFLG+87    B   +
     DAF9       B   +

     ABIT       B   +
      .             +
      .             + =0
      .             +
     ZBIT+6     B   +

     RMODE      T     =0

     PRMPC      C     CONTAINS A COLON (':')

     SC0        C     CONTAINS A 3B (X'FB')

     SC1        C   + CONTAIN A BLANK
     SC2        C   +

TCL=I


     HSEND      S     =HSBEG

     IS         R     =ISBEG

     OS         R     =OSBEG

     OTHER ELEMENTS AS SET BY MD18, IF ENTERED

ELEMENT USAGE (MD1)

     R14        R

SUBROUTINE USAGE (MD1)

     IF PQFLG=0 AND CHAINFLG=0:

          WRTLIN, READLIN

          FOUR ADDITIONAL LEVELS OF SUBROUTINE LINKAGE USED

EXITS (MD1)

     TO 6,PQ=00 IF PQFLG=1, OTHERWISE TO MD18
```

FUNCTIONAL DESCRIPTION (MD1B)

MD1B IS THE POINT WHERE TCL ATTEMPTS TO RETRIEVE A VERB (FIRST SET OF CONTIGUOUS NON-BLANK DATA IN THE INPUT BUFFER) FROM A USER'S MASTER DICTIONARY, AND VALIDATE IT AS SUCH. IF NO ERRORS ARE FOUND, THE REST OF THE DATA IN THE INPUT BUFFER IS EDITED AND COPIED INTO THE IS WORK SPACE, AND CONTROL PASSES TO THE PROCESSOR SPECIFIED IN THE PRIMARY-MODE-ID ATTRIBUTE OF THE VERB, OR TO THE PROC PROCESSOR IF THE DATA DEFINES A PROC (ATTRIBUTE 1='PQ').

OPTION STRINGS, ENCLOSED IN PARENTHESES, ARE ALSO PROCESSED BY TCL AT THIS POINT, UNLESS CHARACTER SCP='0'. SEE GETOPT DOCUMENTATION FOR FURTHER INFORMATION ABOUT OPTIONS.

INPUT INTERFACE (MD1B)

IB	R	POINTS ONE CHARACTER BEFORE THE INPUT DATA
BMSBEG	S	POINTS TO THE BEGINNING OF THE BMS WORK SPACE

OUTPUT INTERFACE (MD1B)

CHAINFLG	B	+	=0
DAFB	B	+	
BASE	D	+	
MODULO	T	+	=MBASE, MMODULO, MSEPAR
SEPAR	T	+	
IB	R	+	POINT TO THE SM AT THE END OF THE
IBEND	S	+	INPUT LINE
BMS	R	+	POINT TO THE LAST CHARACTER IN THE VERB
BMSEND	S	+	NAME (FOR RETIX)
IR	R		POINTS TO THE AM FOLLOWING ATTRIBUTE 4 OF THE VERB ITEM, OR TO THE END-OF-DATA AM IN THE ITEM, OR TO THE "Q" IN ATTRIBUTE ONE IF THE ITEM DEFINES A PROC

SR4	S		POINTS TO THE AM AT THE END OF THE VERB ITEM IN THE MASTER DICTIONARY, IF FOUND
RECORD	D	+	
NNCF	H	+	
FRMN	D	+	
FRMP	D	+	(SEE RETIX DOCUMENTATION)
NPCF	H	+	
SIZE	T	+	

THE FOLLOWING SPECIFICATIONS ARE MEANINGFUL ONLY IF THE FIRST TWO INPUT CHARACTERS ARE NOT 'PQ':

SCP	C	CONTAINS THE CHARACTER IMMEDIATELY FOLLOWING 'P' IN THE VERB DEFINITION, IF PRESENT, OTHERWISE CONTAINS A BLANK
CTR0	T	CONTAINS THE PRIMARY MODE-ID SPECIFIED IN THE VERB DEFINITION
MODEID2	T	CONTAINS THE SECONDARY MODE-ID FROM THE VERB, IF PRESENT, OTHERWISE 0
MODEID3	T	CONTAINS THE TERTIARY MODE-ID FROM THE VERB, IF PRESENT, OTHERWISE 0
BKBIT	B	+
IFLG	B	+ =0, IF NO ERRORS ENCOUNTERED (SEE BELOW)
VFLG	B	+
OS	R	=OSBEG
IS	R	+ POINT ONE CHARACTER BEFORE THE BEGINNING
ISBEG	S	+ OF THE EDITED INPUT LINE. CHARACTERS ARE COPIED FROM THE IB, SUBJECT TO THE FOLLOWING RULES:

1) ALL CONTROL CHARACTERS AND SYSTEM DELIMITERS (SB, SM, AM, VM, SVM) IN THE INPUT BUFFER ARE IGNORED EXCEPT WHEN WITHIN DOUBLE QUOTES ("). CONTROL CHARACTERS (<X'20') ARE ALSO IGNORED WHEN WITHIN SINGLE QUOTES (').

2) REDUNDANT BLANKS (TWO OR MORE BLANKS IN SEQUENCE) 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 AS: SM Z.

ELEMENT USAGE (MD1B)

R14	R
R15	R

XMODE T

REJCTR T (USED ON ERROR CONDITIONS)

D0 D + USED BY GETOPT
D1 D +

SUBROUTINE USAGE (MD1B)

RETIX; CVXIR IF FIRST INPUT CHARACTERS NOT 'PQ'; GETOPT IF A LEFT PARENTHESIS IS ENCOUNTERED OUTSIDE QUOTE MARKS

FOUR ADDITIONAL LEVELS OF SUBROUTINE LINKAGE REQUIRED

EXITS (MD1B)

TO 0,PQ=00 IF FIRST INPUT CHARACTERS ARE 'PQ' AND PQFLG IS ZERO, OR TO 7,PQ=00 IF FIRST INPUT CHARACTERS ARE 'PQ' AND PQFLG IS SET, OTHEREWISE TO THE ENTRY POINT SET UP IN CTR0. IF THE VERB IS NOT FOUND IN THE MASTER DICTIONARY, OR HAS A BAD FORMAT, CONTROL PASSES TO MD99 IN THE WRAPUP PROCESSOR, WHICH PRINTS AN ERROR MESSAGE.

TCL=I

ERROR NUMBER (IN REJCTR)	ERROR TYPE
2	UNEVEN NUMBER OF SINGLE OR DOUBLE QUOTE MARKS IN THE INPUT DATA
3	VERB CANNOT BE IDENTIFIED IN THE M/DICT
30	VERB FORMAT ERROR (PREMATURE END OF DATA OR A NON=HEXADECIMAL CHARACTER PRESENT IN THE MODE=ID)

TCL=II

MD200 (0,TCL=II)
MD201 (1,TCL=II)

FUNCTIONAL DESCRIPTION

THESE ARE THE ENTRY POINTS (NOT SUBROUTINES) INTO THE TCL=II PROCESSOR, USED WHENEVER A VERB REQUIRES ACCESS TO A FILE, OR TO ALL OR EXPLICITLY SPECIFIED ITEMS WITHIN A FILE. MD200 IS ENTERED FROM THE TCL=I PROCESSOR AFTER DECODING THE VERB (PRIMARY MODE=ID = 2). MD201 IS USED BY TCL=II ITSELF TO REGAIN CONTROL FROM WRAPUP UNDER CERTAIN CONDITIONS (SEE BELOW). 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 WICH WAS SPECIFIED IN THE INPUT DATA.

ON ENTRY, TCL-II CHECKS THE VERB DIFINITION FOR A SET OF OPTION CHARACTERS IN ATTRIBUTE 5; VERB OPTIONS ARE SINGLE CHARACTERS IN ANY SEQUENCE AND COMBINATION, AND ARE LISTED BELOW (ALL OTHER CHARACTERS ARE IGNORED).

OPTION	MEANING
C	COPY - ITEMS RETRIEVED ARE COPIED TO THE IS WORKSPACE
E	EXPAND - ITEMS RETRIEVED ARE EXPANDED AND COPIED TO THE IS WORK SPACE (SEE EXPAND DOCUMENTATION); IGNORED IF THE "C" OPTION IS NOT PRESENT
F	FILE ACCESS ONLY - FILE PARAMETERS ARE SET UP BUT ANY ITEM-LIST IS IGNORED BY TCL-II; IF THIS OPTION IS PRESENT, ANY OTHERS ARE IGNORED
N	NEW ITEM ACCEPTABLE - IF THE ITEM SPECIFIED IS NOT ON GILE, THE SECONDARY PROCESSOR STILL GETS CONTROL (THE EDITOR, FOR EXAMPLE, CAN PROCESS A NEW ITEM)
P	PRINT - ON A FULL FILE RETRIEVAL (ALL

	ITEMS), THE ITEM-ID OF EACH ITEM IS PRINTED AS IT IS RETRIEVED
U	UPDATING SEQUENCE FLAGGED - IF ITEMS ARE TO UPDATED AS RETRIEVED, THIS OPTION IS MANDATORY
Z	FINAL ENTRY REQUIRED - THE SECONDARY PROCESSOR WILL BE ENTERED ONCE MORE AFTER ALL ITEMS HAVE BEEN RETRIEVED (THE COPY PROCESSOR, FOR INSTANCE, USES THIS OPTION 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 IN THE FILE.

INPUT INTERFACE

IR	R	POINTS TO THE AM BEFORE ATTRIBUTE 5 OF THE VERB
SR4	S	POINTS TO THE AM AT THE END OF THE VERB

MODEID2	T	CONTAINS THE MODE-ID OF THE PROCESSOR TO WHICH TCL-II TRANSFERS CONTROL (ASSUMING NO ERROR CONDITIONS ARE ENCOUNTERED)
BMSBEG	S	POINTS ONE PRIOR TO AN AREA WHERE THE FILE NAME IS TO BE COPIED, IF THE "F" OPTION IS PRESENT, OTHERWISE ONE PRIOR TO AN AREA WHERE ITEM-IDS ARE TO BE COPIED
ISBEG	S	POINTS ONE PRIOR TO AN AREA WHERE ITEMS ARE TO BE COPIED, IF THE "C" OPTION IS PRESENT

ELEMENTS AS REQUIRED BY GETFILE

OUTPUT INTERFACE

TCL-II

DAF1	B	SET IF THE "U" OPTION IS SPECIFIED
DAF2	B	SET IF THE "C" OPTION IS SPECIFIED
DAF3	B	SET IF THE "P" OPTION IS SPECIFIED
DAF4	B	SET IF THE "N" OPTION IS SPECIFIED
DAF5	B	SET IF THE "Z" OPTION IS SPECIFIED
DAF6	B	SET IF THE "F" OPTION IS SPECIFIED, OR IF A FULL FILE RETRIEVAL IS SPECIFIED (NO "F" OPTION)
DAF10	B	SET IF MORE THAN ONE ITEM IS SPECIFIED IN THE INPUT DATA, BUT NOT A FULL FILE RETRIEVAL ("*")
DAF11	B	SET IF THE "E" OPTION IS SPECIFIED

NOTE: THE ABOVE BITS ARE NOT INITIALIZED TO ZERO

DAF8	B	SET IF A FILE DICTIONARY IS BEING ACCESSED, OTHERWISE RESET (FROM GETFILE)
DAF9	B	=0
IS	R	POINTS ONE PAST THE END OF THE FILE NAME IN THE INPUT STRING IF THE "F" OPTION IS PRESENT; POINTS TO THE LAST AM IN THE COPIED ITEM IF THE "C" OPTION IS PRESENT, OTHERWISE TO THE END OF THE INPUT STRING

ISBEG	S	+ UNCHANGED
BMSBEG	S	+
RMBIT	B	SET IF THE FILE IS SUCCESSFULLY RETRIEVED IF THE "F" OPTION IS PRESENT
SBASE	D	+ CONTAIN THE BASE, MODULO, AND SEPARATION

SMOD	T	+ OF THE FILE BEING ACCESSED
SSEP	T	+
BASE	D	+ =SBASE, SMOD, SSEP ON THE FIRST EXIT
MODULO	T	+ ONLY (FROM MD200)
SEPAR	T	+
DBASE	D	+ CONTAIN THE BASE, MODULO, AND SEPARATION
DMOD	T	+ OF THE DICTIONARY OF THE FILE BEING
DSEP	T	+ ACCESSED IF THE "F" OPTION IS PRESENT
SC0	C	CONTAINS A SB IF THE LAST ITEM-ID IN THE INPUT STRING IS ENCLOSED IN QUOTE MARKS, OTHERWISE CONTAINS A BLANK

THE FOLLOWING SPECIFICATIONS ARE MEANINGFUL ONLY WHEN THE "F" OPTION IS NOT PRESENT:

SR0	S	POINTS ONE PRTOR TO THE COUNT FIELD OF THE RETRIEVED ITEM
SIZE	T	CONTAINS THE VALUE OF THE COUNT FIELD OF THE RETRIEVED ITEM
SR4	S	POINTS TO THE LAST AM OF THE RETRIEVED ITEM
ISEND	S	=IS IF THE "C" OPTION IS PRESENT
IR	R	POINTS TO THE LAST AM OF THE RETRIEVED ITEM TO BE COPIED, IF THE "C" OPTION IS PRESENT, OTHERWISE POINTS TO THE AM FOLLOWING THE ITEM-ID
RMODE	T	=MD201 IF ITEMS ARE LEFT TO BE PROCESSED, OTHERWISE=0
XMODE	T	=0
VOBIT	B	=0 (MD201 ONLY)

C1 T USED FOR ERROR MESSAGES

ELEMENTS USED BY THE VARIOUS SUBROUTINES BELOW

SUBROUTINE USAGE

GETFILE: IF NO 'F' OPTION: GETITM FOR FULL FILE RETRIEVAL, RETIX AND ONE INTERNAL SUBROUTINE IF NOT FULL FILE RETRIEVAL, GETSPC IF MORE THAN ONE ITEM (BUT NOT "*") SPECIFIED, EXPAND IF THE "E" OPTION IS PRESENT, WRTLIN IF THE "P" OPTION IS PRESENT

MD201 ONLY: WSINIT; GNTBLI IF MORE THAN ONE ITEM (BUT NOT "*") SPECIFIED

MD995 AND BMSOVF USED WITH XMODE

SEVEN ADDITIONAL LEVELS OF SUBROUTINE LINKAGE REQUIRED BY MD200: FIVE ADDITIONAL LEVELS REQUIRED BY MD201 FOR FULL FILE RETRIEVAL, OTHERWISE THREE LEVELS REQUIRED

ERROR CONDITIONS

THE FOLLOWING CONDITIONS CAUSE AN EXIT TO THE WRAPUP PROCESSOR WITH THE ERROR NUMBER INDICATED:

ERROR	CONDITION
13	DL/ID ITEM NOT FOUND, OR IN BAD FORMAT
199	IS WORK SPACE NOT BIG ENOUGH WHEN THE "C" OPTION IS SPECIFIED
200	NO FILE NAME SPECIFIED
201	FILE NAME ILLEGAL OR INCORRECTLY DEFINED IN THE M/DICT
202	ITEM NOT ON FILE; ALL MESSAGES OF THIS TYPE ARE STORED UNTIL ALL ITEMS HAVE BEEN PROCESSED; ITEMS WHICH ARE ON FILE ARE STILL PROCESSED

TCL-II

203	NO ITEM LIST SPECIFIED

```
TIME (5,SYSTEM=SUBS=II)*
DATE (6,SYSTEM=SUBS=II)*
TIMDATE (4,SYSTEM=SUBS=II)*
```

FUNCTIONAL DESCRIPTION

THESE ROUTINES RETURN THE SYSTEM TIME AND/OR THE SYSTEM DATE, AND STORE IT IN THE BUFFER AREA SPECIFIED BY REGISTER R15. THE TIME IS RETURNED AS ON A 24-HOUR CLOCK.

ENTRY	BUFFER SIZE REQUIRED (BYTES)	FORMAT
TIME	9	HH:MM:SS
DATE	12	DD MMM YYYY
TIMDATE	22	HH:MM:SS DD MMM YYYY

INPUT INTERFACE

R15	R	POINTS ONE PRIOR TO THE BUFFER AREA

OUTPUT INTERFACE

R15	R	POINTS TO THE LAST BYTE OF THE DATA STORED; THE BYTE IMMEDIATELY FOLLOWING CONTAINS A BLANK
R14FID	D	=0 (DATE AND TIMDATE ONLY)

ELEMENT USAGE

D0	D	*
D1	D	* USED BY TIME AND TIMDATE ONLY
D2	D	*
D3	D	*

SUBROUTINE USAGE

TIME USED BY TIMDATE; MBDSUB USED BY TIME

TWO ADDITIONAL LEVELS OF SUBROUTINE LINKAGE REQUIRED BY

TIME, DATE, TIMDATE

TIMDATE, ONE LEVEL REQUIRED BY TIME, NONE BY DATE

```
TPREAD (6,TAPEIO-I)*
TPWRITE (7,TAPEIO-I)*
```

FUNCTIONAL DESCRIPTION

TPREAD READS ONE RECORD FROM THE TAPE INTO THE TAPE BUFFER (FRAME FFFF10); THE READ STOPS EITHER WHEN THE INTER-RECORD GAP IN THE TAPE IS DETECTED, OR AT THE END OF THE BUFFER.

TPWRITE WRITES ONE RECORD FROM THE TAPE BUFFER TO MAGNETIC TAPE; THE NUMBER OF BYTES WRITTEN IS SET UP BY THE "T-ATT" VERB.

INPUT INTERFACE

ATTACH	B	MUST BE SET, INDICATING THAT THE TAPE UNIT IS ATTACHED

OUTPUT INTERFACE

R15	R	FOR TPREAD, POINTS TO THE LAST BYTE READ
BYTESRD	T	SET TO THE NUMBER OF BYTES READ IN BY TPREAD; SET TO THE TAPE RECORD SIZE BY TPWRITE
PQFLG	B +	
VOBIT	B +	SET TO ZERO ON ERROR EXITS (SEE BELOW)
RMODE	T +	

THE TAPE STATUS BITS ARE RESET APPROPRIATELY (SEE TPSTAT DOCUMENTATION)

ELEMENT USAGE

D0	D +	
D1	D +	UTILITY
T4	T +	
R14	R +	
R2;C0	C	USED TO IDENTIFY EITHER A READ OPERATION (BIT ZERO SET) OR WRITE OPERATION (BIT

ZERO RESET) IN PROGRESS, FOR USE BY COMMON ROUTINES

CTR1 T USED IF TPWRITE ENCOUNTERS AN END-OF-TAPE CONDITION

ALL ELEMENTS USED BY FRMDMP IF THAT ROUTINE IS CALLED BY RDPARITY (SEE RDPARITY AND FRMDMP DOCUMENTATION)

SUBROUTINE USAGE

INIT; TPSTAT; REWIND, CRLFPRINT, PCRLF, PRINT (FOR UNLABELED TAPES), AND THREE INTERNAL SUBROUTINES ON END-OF-TAPE CONDITIONS, AND TPREAD, CVDR15, BCKSP, AND ANOTHER INTERNAL SUBROUTINE ON TPREAD END-OF-TAPE CONDITIONS ONLY; BCKSP ON PARITY ERRORS, AND RDPARITY ON TPREAD PARITY ERRORS ONLY AFTER THE NORMAL NUMBER OF RETRIES (SEE BELOW)

MAXIMUM NINE ADDITIONAL LEVELS OF SUBROUTINE LINKAGE REQUIRED BY TPREAD (FOR PARITY ERRORS ON READING LABELS OF TAPE REELS AFTER THE FIRST); FOUR LEVELS REQUIRED BY TPWRITE

ERROR CONDITIONS

READ PARITY ERROR: THE READ IS REPEATED TWENTY TIMES; IF THE PARITY ERROR PERSISTS, RDPARITY IS CALLED

WRITE PARITY ERROR: THE SEQUENCE "BACKSPACE / WRITE END-OF-FILE MARK / BACKSPACE AND REPEAT WRITE" IS REPEATED TWENTY TIMES; IF THE PARITY ERROR PERSISTS, AN EXIT IS TAKEN TO MD99 WITH MESSAGE 98

END OF FILE: AN EXIT IS TAKEN TO MD99 WITH MESSAGE 94 ON A READ

TAPE NOT ATTACHED: AN EXIT IS TAKEN TO MD99 WITH MESSAGE 93

TPSTAT, TPINIT

```
TPSTAT (15,TAPEIO-I)*
TPINIT (12,TAPEIO-I)*
```

FUNCTIONAL DESCRIPTION

THESE TAPE CONTROL SUBROUTINES ARE USED BY ALL OTHER SYSTEM TAPE I/O ROUTINES. TPINIT OUTPUTS A FUNCTION-CODE OF "1" TO THE TAPE CONTROLLER, THEREBY SETTING IT TO AN INITIAL CONDITION, AND THEN FALLS INTO TPSTAT TO GET THE TAPE STATUS FROM THE CONTROLLER. IT RETURNS IF THE TAPE IS IN A "READY" STATE. IF THE TAPE IS REWINDING, THE SUBROUTINE WAITS UNTIL IT IS FINISHED. OTHERWISE, THE STATUS IS TESTED UP TO ONE HUNDRED TIMES; IF THE TAPE IS STILL NOT READY, NOTREADY IS CALLED.

INPUT INTERFACE

ATTACH	B	=1; CONVENTIONALLY, THE TCL VERB T=ATT IS USED TO SET THIS BIT
R14	R	POINTS TO HALF-TALLY TAPSTW; TPINIT SETS THIS REGISTER AS PART OF THE INITIALIZATION PROCESS

OUTPUT INTERFACE

REJCTR	T	SET TO ZERO BY INIT, AND BY TPSTAT AFTER A "NOT-READY" CONDITION
PQFLG	B	+
VOBIT	B	+ SET TO ZERO IF ATTACH IS ZERO
RMODE	T	+

TAPE STATUS BITS:

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
NORING	B	SET, ON A WRITE OPERATION, IF THE WRITE RING IN THE TAPE IS NOT PRESENT

TPSTAT, TPINIT

PARITY	B	SET IF A PARITY ERROR IS DETECTED
TPRDY	B	SET IF THE TAPE IS READY

ELEMENT USAGE

T6	T	USED AS A DELAY COUNTER
R15	R	UTILITY

SUBROUTINE USAGE

NOTREADY IF THE TAPE IS NOT READY

THREE ADDITIONAL LEVELS OF SUBROUTINE LINKAGE REQUIRED (FOR NOTREADY)

EXITS

TO MD99 WITH MESSAGE 93 ("ATTACH THE TAPE UNIT") IF ATTACH=0

TSINIT

TSINIT (3,TCL-INIT)*

FUNCTIONAL DESCRIPTION

THIS ROUTINE INITIALIZES THE REGISTER TRIAD ASSOCIATED WITH THE TS WORK SPACE.

INPUT INTERFACE

NONE

OUTPUT INTERFACE

```
TS        R  + POINT TO THE BEGINNING OF THE TS WORK
TSBEG     S  + SPACE (PCB+5)
(R14      R) +

TSEND     S  + POINT TO THE LAST BYTE OF THE TS WORK
(R15      R) + SPACE (511 BYTES PAST TSBEG); NOTE THIS
               IS AN UNLINKED WORK SPACE
```

THE FIRST BYTE OF THE WORK SPACE IS SET TO X'00'.

ELEMENT USAGE

NONE (EXCEPT DO)

SUBROUTINE USAGE

ONE INTERNAL SUBROUTINE

ONE ADDITIONAL LEVEL OF SUBROUTINE LINKAGE REQUIRED

UPDITM

UPDITM (0,WRAPUP-II)*

FUNCTIONAL DESCRIPTION

UPDITM PERFORMS UPDATES TO A DISC FILE DEFINED BY ITS BASE FID, MODULO, AND SEPARATION. IF THE ITEM IS TO BE DELETED, THE ROUTINE COMPRESSES THE REMAINDER OF THE DATA IN THE GROUP IN WHICH THE ITEM RESIDES; IF THE ITEM IS TO BE ADDED, IT IS ADDED AT THE END OF THE CURRENT DATA IN THE GROUP; IF THE ITEM IS TO BE REPLACED, IT IS REPLACED IN PLACE, SLIDING THE REMAINING ITEMS IN THE GROUP TO THE LEFT OR RIGHT AS NECESSARY.

IF THE UPDATE CAUSES THE DATA IN THE GROUP TO REACH THE END OF THE LINKED FRAMES, NEXTOVF IS ENTERED TO OBTAIN ANOTHER FRAME FROM THE OVERFLOW SPACE POOL AND LINK IT TO THE PREVIOUS LINKED SET; AS MANY FRAMES AS REQUIRED ARE 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, IT IS RELEASED TO THE OVERFLOW SPACE POOL.

RETIXU IS USED TO RETRIEVE THE ITEM TO BE UPDATED, LOCKING THE GROUP.

ONCE THE ITEM IS RETRIEVED, PROCESSING CANNOT BE INTERRUPTED UNTIL COMPLETED.

INPUT INTERFACE

BMSBEG	S	POINTS ONE PRIOR TO THE ITEM-ID OF THE ITEM TO BE UPDATED; THE ITEM-ID MUST BE TERMINATED BY AN AM
TS	R	POINTS ONE PRIOR TO THE ITEM BODY TO BE ADDED OR REPLACED (NO ITEM-ID OR COUNT FIELD); NOT NEEDED FOR DELETIONS; THE ITEM BODY MUST BE TERMINATED BY A SM
CH8	C	CONTAINS THE CHARACTER 'D' FOR ITEM DELETION; 'U' FOR ITEM ADDITION OR REPLACEMENT

UPDITM

BASE	D	+ CONTAIN THE BASE, MODULO, AND SEPARATION
MODULO	T	+ OF THE FILE BEING UPDATED
SEPAR	T	+

OUTPUT INTERFACE

NONE

ELEMENT USAGE

DAF9	B	+
T3	T	+
T4	T	+
T5	T	+
SIZE	T	+
D3	D	+
D4	D	+
OVRFLW	D	+
RECORD	D	+
NNCF	H	+ UTILITY
FRMN	D	+
FRMP	D	+
NPCF	H	+

```
IR          R  +
UPD         R  +
BMS         R  +
CS          R  +
R14         R  +
R15         R  +
SR4         S  +
```

ELEMENTS USED BY THE VARIOUS SUBROUTINES BELOW

SUBROUTINE USAGE

RDLINK; RDREC; RETIXU; DECINHIB; GUNLOCK; RELCHN IF OVERFLOW FRAMES RETURNED; ATTSPC IF MORE OVERFLOW FRAMES NEEDED; TWO INTERNAL SUBROUTINES

NEXTOFV AND ONE LOCAL SUBROUTINE USED WITH XMODE

FOUR ADDITIONAL LEVELS OF SUBROUTINE LINKAGE REQUIRED

UPDITM

ERROR CONDITIONS

1. IF THE GROUP DATA IS BAD (PREMATURE END OF LINKED FRAMES, OR NON-HEXADECIMAL CHARACTER FOUND IN AN ITEM COUNT FIELD), IROVF IS ENTERED TO PRINT A WARNING MESSAGE, AND THE GROUP DATA IS TERMINATED AT THE END OF THE LAST GOOD ITEM BEFORE PROCESSING CONTINUES

2. IF THE FILE BEING UPDATE IS THE M/DICT (BASE=MBASE), AND BIT SYSPRIV1 IS ZERO, PRIVTST2 IS ENTERED AND NO UPDATE IS PERFORMED

3. IF THE ITEM-ID CONTAINS MORE THAN 50 CHARACTERS, IT IS TRUNCATED WITHOUT ANY INDICATION

4. IF THE ITEM EXCEEDS THE MAXIMUM SIZE (32267 BYTES, X'7E0B'), THE ITEM IS TRUNCATED TO THE MAXIMUM SIZE, AND NO INDICATION IS GIVEN

WEOF

WEOF (11,TAPEIO-I)*

FUNCTIONAL DESCRIPTION

WEOF WRITES AN END-OF-FILE MARK ON THE TAPE. IF BIT PROTECT IS SET, IT CALLS NORING REPEATEDLY UNTIL IT IS RESET (OR BIT ATTACH IS RESET) BEFORE ATTEMPTING A WRITE. WEOF ALSO CALLS INIT AND TPSTAT, AND REQUIRES FOUR ADDITIONAL LEVELS OF SUBROUTINE LINKAGE.

WRAPUP PROCESSOR

```
MD992 (6,WRAPUP-I)*
MD993 (2,WRAPUP-I)*
MD994 (4,WRAPUP-I)*
MD995 (3,WRAPUP-I)*
 MD99 (0,WRAPUP-I)*
MD999 (1,WRAPUP-I)*
```

FUNCTIONAL DESCRIPTION

THESE ARE THE ENTRY POINTS INTO THE SYSTEM ROUTINE WHICH "WRAPS UP" THE PROCESSING INITIATED BY A TCL STATEMENT, PERFORMS DISK UPDATES AND PRINTS MESSAGES AS REQUIRED, AND REINITIALIZES FUNCTIONAL ELEMENTS FOR PROCESSING ANOTHER TCL STATEMENT. WRAPUP MAY ALSO BE TREATED AS A SUBROUTINE BY SETTING TALLY RMODE TO THE MODE-ID OF THE ROUTINE TO WHICH WRAPUP SHOULD RETURN CONTROL AFTER IT IS DONE. NOTE, HOWEVER, THAT WRAPUP ALWAYS SET THE RETURN STACK TO A NULL OR EMPTY CONDITION BEFORE EXITING.

THE VARIOUS ENTRY POINTS ARE PROVIDFD TO SIMPLIFY THE INTERFACE REQUIREMENTS WHEN WRAPUP IS USED TO STORE OR PRINT MESSAGES FROM THE ERRMSG FILE; THE FEATURES OF EACH CAN BE SEEN IN THE FOLLOWING TABLE:

MD992	C1 CONTAINS A MESSAGE NUMBER; D9 CONTAINS A NUMERIC PARAMETER; THE VALUE IN C1, CONVERTED TO AN ASCII STRING, IS USED AS THE ITEM-ID OF AN ITEM TO BE RETRIEVED FROM THE MESSAGE FILE (NORMALLY ERRMSG); THE MESSAGE IS SET UP IN THE HISTORY STRING (SEE BELOW), AND CONTROL PASSES TO MD99
MD993	C1 CONTAINS A MESSAGE NUMBER; C2 CONTAINS A NUMERIC PARAMETER; THE VALUE IN C1, CONVERTED TO AN ASCII STRING, IS USED AS THE ITEM-ID OF AN ITEM TO BE RETRIEVED FROM THE MESSAGE FILE (NORMALLY ERRMSG); THE MESSAGE IS SET UP IN THE HISTORY STRING (SEE BELOW), AND CONTROL PASSES TO MD99

WRAPUP

MD994	C1 CONTAINS A MESSAGE NUMBER; IS POINTS ONE BEFORE THE BEGINNING OF A STRING PARAMETER, WHICH IS TERMINATED BY AN AM OR SM; THE MESSAGE IS SET UP IN THE HISTORY STRING AND CONTROL PASSES TO MD99

MD995 — LIKE MD994, EXCEPT THE STRING PARAMETER IS STORED AT 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); IF RMODE IS ZERO, MESSAGES ARE PRINTED REGARDLESS OF THE VALUE OF VOBIT (SEE BELOW); THE MESSAGES ARE SET UP IN THE HISTORY STRING AND CONTROL PASSES TO MD999

MD999 — THE HISTORY STRING IS PROCESSED, AND PROCESS WORK SPACES ARE REINITIALIZED; CONTROL PASSES TO TCL IF RMODE IS ZERO, OTHERWISE TO THE ROUTINE SPECIFIED BY RMODE

INPUT INTERFACE

HSBEG S + POINT ONE BEFORE THE BEGINNING AND TO
HSEND S + THE END, RESPECTIVELY, OF THE HISTORY STRING; IF HSBEG=HSEND, THE STRING IS NULL

THREE TYPES OF HISTORY STRING ELEMENTS ARE RECOGNIZED BY WRAPUP; ALL OTHERS ARE IGNORED. THE TYPE OF PROCESSING DONE FOR EACH ELEMENT DEPENDS ON THE SECOND, AND POSSIBLY THIRD CHARACTER OF THE ELEMENT STRING. (THE QUOTE MARKS IN THE FOLLOWING EXAMPLES ARE NOT PART OF THE STRINGS.)

1. OUTPUT MESSAGE

SM "O" AM MESSAGE-ID AM (PARAMETER AM...) SM

WRAPUP

WHERE "MESSAGE-ID" IS THE ITEM-ID (NORMALLY A DECIMAL NUMERIC) OF AN ITEM IN THE MESSAGE FILE

THE PARAMETER STRING IS PASSED TO PRTERR FOR MESSAGE FORMATTING (SEE PRTERR DOCUMENTATION)

2. DISK UPDATE/DELETE

SM "DU" AM BASE VM MODULO VM SEPARATION AM ITEM-ID AM ITEM-BODY AM SM

SM "DD" AM BASE VM MODULO VM SEPARATION AM ITEM-ID AM SM

WHERE "DU" CAUSES THE ITEM IN THE FILE SPECIFIED BY "BASE", "MODULO", AND "SEPARATION" TO BE REPLACE, AND "DD" DELETES IT

3. (END OF HISTORY STRING)

SM "Z"

CONVENTIONALLY, A PROCESS WISHING TO ADD DATA TO THE HISTORY STRING BEGINS AT HSEND+1; AFTER THE ADDITIONAL ELEMENTS HAVE BEEN ADDED, THE STRING IS TERMINATED (ONCE AGAIN) BY A SM AND "Z", AND HSEND IS SET POINTING TO THIS SM.

WMODE	T	IF NON-ZERO, THE VALUE IS USED AS THE MODE-ID FOR AN INDIRECT SUBROUTINE CALL (BSLI *) EXECUTED IMMEDIATELY AFTER THE HISTORY STRING HAS BEEN PROCESSED, AND BEFORE WORK SPACE AND PRINTER CHARACTERISTICS ARE RESET; THIS ALLOWS SPECIAL PROCESSING TO BE DONE ON ANY ENTRY INTO WRAPUP
RMODE	T	IF NON-ZERO, WRAPUP EXITS TO THE SPECIFIED MODE-ID INSTEAD OF TO TCL
VOBIT	B	IF SET, AND RMODE IS NON-ZERO, MESSAGES ARE STORED IN THE HISTORY STRING, FOR OUTPUT ON A LATER ENTRY INTO WRAPUP WITH

WRAPUP

RMODE ZERO

REJCTR	T	+	MAY CONTAIN MESSAGE NUMBERS WHICH DO NOT
REJ0	T	+	REQUIRE PARAMETERS; REJCTR IS ALWAYS
REJ1	T	+	TESTED FIRST, THEN REJ0, AND THEN REJ1; NO ACTION IS TAKEN ON A ZERO VALUE; A VALUE OF 9999 IS USED INTERNALLY BY WRAPUP TO IDENTIFY WHICH MESSAGES HAVE BEEN PROCESSED, AND SHOULD NOT NORMALLY BE USED AS AN INPUT VALUE FOR REJ0 OR REJ1
C1	T	+	(SEE MD992, MD993, MD994, AND MD995
C2	T	+	ABOVE)
D9	D	+	
LPBIT	B		IF SET, ALL OPEN SPOOL FILES ARE CLOSED
OVRFLCTR	D		IF NON-ZERO, USED AS THE STARTING FID OF A LINKED SET OF OVERFLOW FRAMES WHICH IS RELEASED TO THE SYSTEM OVERFLOW SPACE POOL; USED BY SORT, FOR INSTANCE, TO STORE THE BEGINNING FID OF A SORTED TABLE, IN WHICH CASE THE OVERFLOW SPACE USED BY SORT IS ALWAYS RELEASED, EVEN IF PROCESSING IS ABORTED BY AN "END" COMMAND FROM DEBUG

```
     USER       T    USED TO CONTROL THE FINAL EXIT FROM
                     WRAPUP WHEN RMODE=0; SEE "EXITS"

OUTPUT INTERFACE

     HSEND      S    =HSBEG EXCEPT WHEN MESSAGES ARE STORED
                     INSTEAD OF PRINTED

     VOBIT      B  +
     LPBIT      B  +
     WMODE      T  + =0
     REJCTR     T  +
     REJ0       T  +
     REJ1       T  +

WRAPUP

     RETURN STACK   NULL; RSEND=X'01B0', RSCWA=X'0184', AND
                    THE REST OF THE RETURN STACK IS FILLED
                    WITH X'FF'

     RMODE      T   SET TO ZERO BY TCLXIT AND NSPCQ

     INHIBITH   H    =0

     ELEMENTS AS INITIALIZED BY WSINIT (AND ISINIT IF
     RMODE=0)

     THE FOLLOWING ELEMENTS ARE SET UP ONLY IF RMODE=0;

     XMODE      T  + =0
     OVRFLCTR   T  +

     IBSIZE     T    =140

ELEMENT USAGE

     UPD        R

     BASE       D  +
     MODULO     T  + USED IN DISK UPDATES
     SEPAR      T  +
     CH8        C  +

     ELEMENTS USED BY THE SUBROUTINES BELOW

SUBROUTINE USAGE

     WSINIT; MBDSUB FOR MESSAGE NUMBERS; PRTERR TO PRINT
     MESSAGES; CVDIS AND UPDITM TO DO DISK UPDATES;
     CRLFPRINT IF A FORMAT ERROR IS FOUND IN A "DD" OR "DU"
     HISTORY STRING ELEMENT; PCLOSEALL IF LPBIT=1; IF
     RMODE=0; ISINIT, RESETTERM, RELSP (IF USER=2), RELCHN
     (IF OVRFLCTR IS NON-ZERO)
```

MAXIMUM OF SEVEN ADDITIONAL LEVELS OF SUBROUTINE LINKAGE REQUIRED IF RELCHN MUST PRINT AN ERROR MESSAGE; MAXIMUM OF SIX LEVELS REQUIRED FOR PRTERR; FOUR LEVELS REQUIRED FOR UPDITM; THREE LEVELS REQUIRED FOR ISINIT; TWO LEVELS ALWAYS NEEDED FOR WSINIT

WRAPUP

EXITS

TO THE ENTRY POINT SPECIFIED IN RMODE IF NON-ZERO; TO LOGOFF IF USER=3 (SET, FOR INSTANCE, BY THE DEBUG "OFF" COMMAND); TO MD0 IF USER=2 (SET BY THE LOGOFF PROCESSOR); OTHERWISE TO MD1

ERROR CONDITIONS

IF A FORMAT ERROR IS FOUND IN A "DD" OR "DU" HISTORY STRING ELEMENT, THE MESSAGE

DISK-UPD STRING ERR

IS DISPLAYED, AND PROCESSING CONTINUES WITH THE NEXT ELEMENT

WRTLIN, WT2, WRITOB

```
WRTLIN (2,TERMIO)*
 WT2 (10,TERMIO)
WRITOB (3,TERMIO)*
```

FUNCTIONAL DESCRIPTION

THESE ARE THE STANDARD ROUTINES FOR OUTPUTTING DATA TO THE TERMINAL OR LINE PRINTER. ENTRY WRTLIN DELETES TRAILING BLANKS FROM THE DATA AND THEN ENTERS WT2. WT2 ADDS A TRAILING CARRIAGE RETURN AND LINE FEED, INCREMENTS LINCTR, AND ENTERS WRITOB, WHICH OUTPUTS THE DATA.

THE DATA TO BE OUTPUT IS POINTED TO BY OBBEG, AND CONTINUES THROUGH THE ADDRESS POINTED TO BY OB. OUTPUT IS ROUTED TO THE TERMINAL IF BIT LPBIT IS OFF, OTHERWISE IT IS STORED IN THE PRINTER SPOOLING AREA. PAGINATION AND PAGE-HEADING ROUTINES ARE INVOKED AUTOMATICALLY IF BIT PAGINATE IS SET. IF IT IS SET, THEN 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 HEADING IS PRINTED (SEE PRNTHDR DOCUMENTATION). A VALUE OF ZERO IN PAGSIZE SUPPRESSES PAGINATION, HOWEVER, REGARDLESS OF THE SETING OF PAGINATE.

INPUT INTERFACE

OBBEG	S	POINTS ONE BYTE PRIOR TO THE OUTPUT DATA BUFFER
OB	R	POINTS TO THE LAST CHARACTER IN THE BUFFER; THE BUFFER MUST EXTEND AT LEAST ONE CHARACTER BEYOND THIS LOCATION
LPBIT	B	IF SET, OUTPUT IS ROUTED TO THE SPOOLER (NOTE: ROUTINE SETLPTR SHOULD BE USED TO SET THIS BIT SO PRINTER CHARACTERISTICS ARE SET UP CORRECTLY)
LISTFLAG	B	IF SET, ALL OUTPUT TO THE TERMINAL IS SUPPRESSED

WRTLIN, WT2, WRITOB

NOBLNK	B	IF SET, BLANKING OF THE OUTPUT BUFFER IS SUPPRESSED
LFDLY	T	LOWER BYTE CONTAINS THE NUMBER OF "FILL" CHARACTERS TO BE OUTPUT AFTER A CR/LF
PAGINATE	B	IF SET, PAGINATION AND PAGE-HEADINGS ARE INVOKED
PFILE	T	CONTAINS THE PRINT FILE NUMBER FOR PPUT; MEANINGFUL ONLY IF LPBIT IS SET

THE FOLLOWING SPECIFICATIONS ARE MEANINGFUL ONLY IF PAGINATE IS SET:

PAGHEAD	S	POINTS ONE BYTE BEFORE THE BEGINNING OF THE PAGE-HEADING MESSAGE; IF THE FRAME FIELD OF THIS REGISTER IS ZERO, NO HEADING IS PRINTED
PAGSIZE	T	CONTAINS THE NUMBER OF PRINTABLE LINES PER PAGE
PAGSKIP	T	CONTAINS THE NUMBER OF LINES TO BE SKIPPED AT THE BOTTOM OF EACH PAGE
PAGNUM	T	CONTAINS THE CURRENT PAGE NUMBER
PAGFRMT	B	IF SET, THE PROCESS PAUSES AT THE END OF EACH PAGE OF OUTPUT UNTIL SOME TERMINAL INPUT (EVEN JUST A CARRIAGE RETURN) IS ENTERED

LFDLY	T	IF THE UPPER BYTE IS GREATER THAN ONE, AND OUTPUT IS TO THE TERMINAL, A FORM-FEED (X'0C') IS OUTPUT AT THE TOP EACH PAGE, AND THE NUMBER IN THE UPPER BYTE IS USED AS THE NUMBER OF "FILL" CHARACTERS OUTPUT AFTER THE FORM-FEED

OUTPUT INTERFACE

WRTLIN, WT2, WRITOB

OB	R	=OBBEG

THE FOLLOWING SPECIFICATIONS ARE MEANINGFUL ONLY IF PAGINATE IS SET:

LINCTR	T	+ RESET APPROPRIATELY
PAGNUM	T	+
T7	T	CONTAINS THE ORIGINAL VALUE OF PAGNUM

ELEMENT USAGE

R14	R	+
R15	R	+ SCRATCH
SYSR1	S	+
R8	R	+
RECORD	T	+ USED BY PPUT (WHEN LPBIT IS SET)
OVRFLW	T	+
SYSR2	S	USED IF PAGINATE IS SET AND THE HEADER MESSAGE CONTAINS A VM
T4	T	+
T5	T	+ USED IF PAGINATE IS SET AND THE HEADER
D2	D	+ MESSAGE CONTAINS A SVM
D3	D	+

ALL ELEMENTS USED BY ATTOVF (CALLED BY PPUT IF MORE DISK SPACE NEEDED)

SUBROUTINE USAGE

FFDLY, PPUT (IF LPBIT SET), WT2 (IF PAGINATE SET AND THE HEADER MESSAGE CONTAINS A VM), TIMDATE (IF PAGINATE SET AND THE HEADER MESSAGE CONTAINS A SVM), DATE (IF PAGINATE SET AND THE HEADER MESSAGE CONTAINS TWO SVMS IN SUCCESSION)

FOUR ADDITIONAL LEVELS OF SUBROUTINE LINKAGE REQUIRED IF LPBIT IS SET; THREE LEVELS REQUIRED FOR TIMDATE; ONE LEVEL ALWAYS REQUIRED FOR LFDLY

WSINIT (1,TCL-INIT)*

FUNCTIONAL DESCRIPTION

THIS ROUTINE INITIALIZES THE FOLLOWING PROCESS WORK SPACE POINTER TRIADS: BMS, BMSBEG, BMSEND; CS, CSBEG, CSEND; AF, AFBEG, AFEND; TS, TSBEG, TSEND; IB, IBBEG, IBEND; OB, OBBEG, OBEND; ALSO PBUFBEG AND PBUFEND. IN EACH CASE, THE "BEGINNING" STORAGE REGISTER (AND ASSOCIATED ADDRESS REGISTER, IF PRESENT) IS SET POINTING TO THE FIRST BYTE OF THE WORK SPACE, AND THE "ENDING" STORAGE REGISTER IS SET POINTING TO THE LAST DATA BYTE. ALL WORK SPACES EXCEPT THE LAST (PROC) ARE CONTAINED IN 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
TSBEG-TSEND	511
PBUFBEG-PBUFEND	20000 (4 LINKED FRAMES)

INPUT INTERFACE

IBSIZE	T	SIZE OF IB BUFFER
OBSIZE	T	SIZE OF OB BUFFER

OUTPUT INTERFACE

REGISTERS ARE SET UP AS DESCRIBED ABOVE. THE FIRST BYTE OF EACH WORK SPACE, EXCEPT THE OB, IS SET TO X'00'. THE OB WORK SPACE IS FILLED WITH BLANKS (X'20'). IBSIZE AND OBSIZE ARE SET TO 140 IF INITIALLY

GREATER.

ELEMENT USAGE

R14	R
R15	R

SUBROUTINE USAGE

TSININIT (LOCAL), AND ONE INTERNAL SUBROUTINE

TWO ADDITIONAL LEVELS OF SUBROUTINE LINKAGE REQUIRED

WTLABEL

WTLABEL (2,TAPEIO-III)*

FUNCTIONAL DESCRIPTION

THIS ROUTINE MAY BE CALLED ONCE BY ANY ROUTINE TO WRITE A LABEL AT THE BEGINNING OF A MAGNETIC TAPE FILE. THE LABEL PASSED AS AN INPUT PARAMETER IS WRITTEN TO THE TAPE, ALONG WITH THE CURRENT TIME AND DATE, REEL NUMBER (ONE), AND THE TAPE RECORD SIZE. THE LABEL INFORMATION IS ALSO STORED IN THE LABEL SAVE BUFFER IN THE QUATERNARY CONTROL BLOCK (PCB+3). FOR THE FORMAT OF THE LABEL DATA ON THE TAPE AND IN THE SAVE BUFFER, SEE THE RDLABEL DOCUMENTATION.

INPUT INTERFACE

IS	R	POINTS ONE BEFORE THE LABEL DATA, WHICH MUST BE TERMINATED BY A STANDARD SYSTEM DELIMITER (SM, AM, VM, SVM, OR SB); IF THE LABEL DATA IS GREATER THAN SIXTEEN CHARACTERS LONG, IT WILL BE TRUNCATED TO SIXTEEN CHARACTERS

OUTPUT INTERFACE

IS	R	POINTS TO THE DELIMITER TERMINATING THE LABEL, OR TO SIXTEEN BYTES PAST THE INPUT POSITION IF NONE IS FOUND

THE LABEL SAVE AREA IS INITIALIZED AS DESCRIBED, AND THE REEL NUMBER IS SET TO ONE

ELEMENT USAGE

R13	R	+
R14	R	+ UTILITY
R15	R	+
D2	D	+ USED BY TIMDATE
D3	D	+

SUBROUTINE USAGE

INIT; TIMDATE; TPWRITE; TWO INTERNAL SUBROUTINES

WTLABEL

FIVE ADDITIONAL LEVELS OF SUBROUTINE LINKAGE REQUIRED

WTREC

WTREC (5,DISKFIO=I)

FUNCTIONAL DESCRIPTION

THIS ROUTINE COPIES THE CONTENTS OF THE UPD WORKSPACE (1 FRAME) INTO THE FRAME SPECIFIED BY TALLY RECORD. ADDITIONALLY THE SUBROUTINE WTLINK IS ENTERED TO SET UP R15 POINTING TO THE LINK PORTION OF THE FRAME, AND TO SET UP THE LINK ELEMENTS NNCF, NPCF, FRMN, AND FRMP.

INPUT INTERFACE

RECORD	D	CONTAINS THE FID OF THE FRAME INTO WHICH DATA IS TO BE COPIED
UPDBEG	S	POINTS ONE PRIOR TO THE FIRST DATA BYTE OF THE UPD WORK SPACE (OR ANY FRAME FROM WHICH DATA IS TO BE COPIED)

OUTPUT INTERFACE

UPD	R	=UPDBEG+500
R14	R	POINTS TO THE LAST BYTE OF THE FRAME SPECIFIED BY RECORD
R15	R +	
NNCF	H +	
FRMN	D +	(SEE RDLINK/WTLINK DOCUMENTATION)
FRMP	D +	
NPCF	H +	

ELEMENT USAGE

NONE (BESIDES UPD AND R14)

SUBROUTINE USAGE

NONE

XISOS (14,DISKFIO=I)*

FUNCTIONAL DESCRIPTION

XISOS SIMPLY EXCHANGES THE CONTENTS OF THE IS/ISBEG/ISEND AND OS/OSBEG/OSEND REGISTER TRIADS. REGISTER R14 IS USED FOR SCRATCH PURPOSES.

SECTION 6
CHANGES AFFECTING ASSEMBLY CODE

THIS CHAPTER IS CONCERNED WITH THE ASSEMBLY LANGUAGE CHANGES RELEVANT TO 2.X LEVEL UPGRADES TO RELEASE 3.0.

OVERVIEW

1) ALL ROUTINES MUST BE REASSEMBLED TO INSURE CORRECT OPCODES AND REFERENCES TO PSYM ELEMENTS

2) THE FIRST 6 LINES OF ALL ASSEMBLY MODES SHOULD BE OF THE FORMAT DOCUMENTED BELOW.

3) SYSTEM ROUTINES SHOULD BE REFERENCED BY 3.0 NAMES. THESE NAMES HAVE BEEN CHANGED TO MORE ACCURATELY REFLECT THE FUNCTIONS OF THE SUBROUTINES OR ELEMENTS. IF THIS IS NOT DESIREABLE, THE USER MAY RECREATE THE OLD NAMES.

4) CALLS TO THE ROUTINE CVDR15 (AND OTHER CVD.. ROUTINES) ALWAYS ASSUME THAT THE INPUT REGISTER POINTS ONE BEFORE THE INPUT DATA. THIS WAS DONE TO DECREASE THE CHANCES OF PROGRAM BUGS.

5) THE RESTRICTIONS ON THE USE OF THE 'BE' AND 'BU' INSTRUCTIONS HAVE BEEN GREATLY REDUCED. THE FORMS 'BRE' AND 'BRU' ARE OBSOLETE.

6) CALLS TO 'MBDSUB CONVERT A 6-BYTE NUMBER (D0+T2) INSTEAD OF A FOUR BYTE NUMBER (D0). 6 BYTE LOAD INSTRUCTIONS (I.E. LOADX ..) ARE PROVIDED TO MAKE THIS EASIER.

7) OPTION STRINGS IN INPUT LINES SET BITS ABIT-ZBIT. BITS AFLG TO ZFLG ARE PROVIDED FOR USE OTHER THAN AS OPTIONS. OPTION PARSING CAN BE INHIBITED IN VERBS BY SETTING SCP = 0. SEE SUBROUTINE 'GETOPT'. TEXT CONTAINING '(' WHICH DOES NOT LOOK LIKE AN OPTION STRING WILL NOT BE TREATED AS AN OPTION STRING, AND THE MESSAGE 'INVALID OPTION STRING' HAS BEEN ELIMINATED.

8) PROCESSORS SUCH AS BASIC AND ENGLISH HAVE BEEN MODIFIED TO AVOID USING THE FOLLOWING ELEMENTS WHICH ARE NOW AVAILABLE FOR USER CODE:

A) SR20 TO SR29
B) CTR30 TO CTR42
C) SB20 TO SB32

9) BECAUSE OF UPDATES IN PLACE, THE OUTPUT INTERFACE OF UPDITM HAS BEEN REDUCED.

10) THE INTERFACE FOR USER EXITS FROM PROC HAS BEEN EXPANDED. SEE THE SECTION CONCERNING PROC IN CHAPTER 5.

11) LOCAL 'BSL' INSTRUCTIONS NOW PLACE A FID IN THE RETURN STACK. THE FIRST 2 BYTES OF THE RETURN STACK ARE ASSUMED BY THE FIRMWARE AS X'01B0'.

12) THE RIGHTMOST BIT OF THE LINK FIELD OF REGISTERS IS USED AS AN EXTENSION OF THE WA FIELD. COMPARES OF FID'S WHICH WERE WRITTEN WITH THE ASSUMPTION THAT THE FLAG FIELD COULD BE INCLUDED IN THE COMPARE ARE NO LONGER VALID. (E.G. BE R6FID,RECORD,LABEL). A NEW FORM OF INSTRUCTION (BE3, BU3, BL3, BH3) HAS BEEN INTRODUCED TO COMPARE ONLY THE LAST 3 BYTES OF A 4 BYTE FIELD (DOUBLE TALLY).
(E.G. BE3 R6FID,RECORD,LABEL).

13) THE INTERFACE TO THE TAPE ROUTINES HAS CHANGED TO ALLOW LARGE TAPE BLOCKS. SEE THE DOCUMENTATION ON 'TPREAD' AND 'TPWRITE'.

14) THE ASSEMBLER OUTPUT LISTING INCLUDES A FIELD FOR DEFINITION LINES THAT SHOWS THE VALUES GENERATED IN THE TSYM ENTRY FOR THAT DEFINITION.

15) THE LOGICAL COMPARE INSTRUCTIONS NOW ORDER THE VALUES OF BYTES AS:

00,01,02...7F,80,81,...FE,FF

FORMERLY, THESE WERE ORDERED AS:

80,81,...FF,00,01,...7E,7F

THIS AFFECTS THE BCL AND BCH INSTRUCTIONS ONLY.

16) THE BASIC DEBUGGER USES PCB+28.

PSYM ELEMENTS WITH NEW NAMES

OLD NAME	NEW NAME
CVTNIB	CVDIB
CVTNIR	CVDIR
CVTNIS	CVDIS
CVTNOS	CVDOS
CVTHIB	CVXIB
CVTHIR	CVXIR
CVTHIS	CVXIS
CTHOS	CVXOS
TILD	DECINHIB
GETIB	READLIN
GETIBX	READLINX
ASEND	BDESCTBL
STKEND	STKINP
CARRIER	BREAKKEY
CVDR15X	CVDR15
CVTHISX	CVXIS
CVTNISX	CVDIS
CVXR15X	CVXR15
IOBIT14	OTABFLG
IOBIT2	PATTACH
IOBIT4	ITABFLG
SMCONV	FRMTFLG

DELETED PSYM ENTRIES

ABSD	ASBEG	ASEND	ASTR
BDIV	BMUL	C8	C9
CARRIER	CHARGE-UNITS-EXT	CHKSUM	CONFIG
CTR	CVDR15X	CVFR15	CVXR15X
D2L	D2U	DATEQ	DDUMP
DFREE	DISKERR	DISP	DIVFF
DOODAH	ENDBIT	FP0(T0T1)	FP0(T2)
FP1(H0)	FP1(L)	FP1(T1T2)	FP1(U)
FP2(L)	FP2(U)	FP3(L)	FP3(U)
FPX(T0)	FPX(T0T1)	FPX(T1)	FPX(T2)
FPY(T0)	FPY(T0T1)	FPY(T1T2)	FPY(T2)
GETIB	GETIBX	HEADING	ICONVMD
II	IIBEG	IIEND	IOBIT14
IOBIT2	IOBIT4	ITAPEBIT	LOCKBITS
LOCX	LOGUNLOCK	MD10	MD10FL
MD11	MD16	MD18	MD23
MD9	MULFF	NEGFP0	NOBIT
NREC	OCONVMD	OVRFLWQ	PSYM
PTRPAG	QSTR	R3SAVE	RDUMP
REG	REJ3	REJ4	REJ5
S5DSP	SB36	SB40	SB43
SETPIB	SMCONV	SMODSEP	STKINP
T-LOAD	T3T2T1	T4T5T6	T5T6T7
TCLXIT	TFREE	TIL	TILD
TPBIT	TYMO	TYPE	WSINT

NEW PSYM ENTRIES

AFLG	ATTSPC	B14	B29
B8	B9	BCKSP	BFLG
CFLG	CVDIB	CVDIR	CVDIS
CVDOS	CVXIB	CVXIR	CVXIS
CVXOS	DAF0	DFLG	DMODDSEP
EFLG	FFLG	FLAGS	FRMTFLG
GACBMS	GETFILE	GETLSPC	GFLG
GLOCKFLG	H8	H9	HFLG
IFLG	JFLG	KFLG	LFLG
MFLG	NFLG	NUMBIT	NUMFLG1
NUMFLG2	OFLG	PFLG	QFLG
R2WADSP	R7WADSP	R8WADSP	READIB
READLIN	READLINX	RETIXU	RFLG
SFLG	SLEEPSUB	SMODSSEP	STKEND
SYSTEM-SUBS-IV	TFLG	TPINIT	TPSTAT
UFLG	VFLG	WFLG	XFLG
YFLG	ZFLG		

DELETED OSYM ENTRIES

	B:A	BITNN	BSLA
CHAIN	DECF	DEFA	DEFBY
DEFDX	DEFDY	DEFF	DEFHY
DEFTH	DEFTY	EQU	ESSR
FTLY	INCF	MBDFR	MBDNFR
RVP	SVP		

Microdata Corporation
17481 Red Hill Avenue, Irvine, California 92714
Post Office Box 19501, Irvine, California 92713
Telephone: 714/540-6730 · TWX: 910-595-1764