Assembly Language

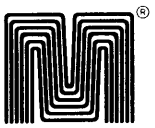
REFERENCE MANUAL

$20.00

REALITY®

ASSEMBLY LANGUAGE REFERENCE MANUAL

771009A
Preliminary 5/76



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

TABLE OF CONTENTS (Continued)

TABLE OF CONTENTS (Continued)

TABLE OF CONTENTS (Continued)

TABLE OF CONTENTS (Continued)

SECTION 1

INTRODUCTION

1.1 THE REALITY CPU

The Reality CPU, althouth physically small in size, has the architecture of a medium scale computer. Its main memory is core, and is expandable from 8,192 bytes to 65,536 bytes in increments of 8,192 bytes. Its full cycle operation is 1 microsecond per byte.

The virtual memory is disc which is oriented into 512-byte frames, expandable from 4,871 frames (2.5 million bytes) to 12,192,320 frames (6.4 billion bytes). This is the virtual memory addressing range of the CPU itself. However, in standard configurations, the Reality Computer System is configured from 5 million bytes to 80 million bytes of disc storage. The CPU is capable of handling a large number of asynchronous processes, each associated with an input/output device. The Reality CPU will support in excess of 32 terminals (or asynchronous processes).

The CPU has 16 addressing registers and one extended accumulator for each terminal. A variable return stack accommodating up to 31 recursive subroutine calls for each terminal is also provided; however, current software convention allows only 11 entries in the stack. By indirect addressing through any one of the 16 registers, any byte in the virtual memory can be accessed. Relative addressing is also possible using an offset displacement plus one of the 16 registers to any bit, byte, word (16 bits), double word (32 bits), or triple word (48 bits) in the entire virtual memory.

1.2 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 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.

1.3 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 describes the Reality System Software, which may be used to facilitate assembly level programming.

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 elipses (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 a separate document.

2.1 SYSTEM STRUCTURE

The Reality system consists of a core storage unit, a disc storage device used as a virtual storage unit, a central processing unit (CPU), and from one to 64 input/output terminals. There is a one-to-one correspondence between a terminal attached to the system and a process. Additionally, input/output devices such as magnetic tape units, disc units, card readers, and printers may be attached to the system. Input/output devices, other than the process terminal, may be accessed by any process. It should be noted that the disc unit containing the virtual store cannot be accessed as an input/output unit, except by the monitor.

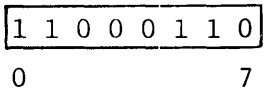
2.1.1 INFORMATION FORMATS

The system transmits information between the CPU and core storage (and between core storage and virtual storage) in units of 8 bits, or in multiples of 8 bits at a time. Each 8 bit unit is called a byte.

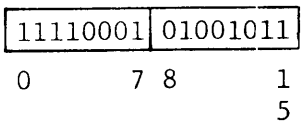
Information may be a single byte, or may be grouped together in fields. Fields of two, four, and six bytes are called words, double words, and triple words, respectively. A field made up of an arbitrary number of bytes is called a string. The location of any field is specified by the address of the left most byte of the field. Addresses increase from left to right.

Within any information format, the bits making up the format are numbered from left to right, starting with 0. The figure below shows the information formats:

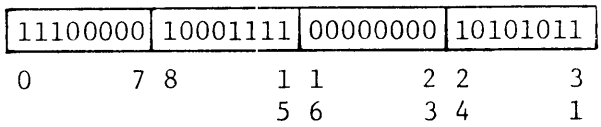
BYTE



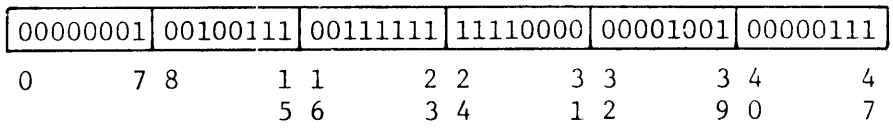
WORD



DOUBLE WORD



TRIPLE WORD



2.1.2 ADDRESSING

Byte locations in main storage are consecutively numbered starting with zero. Each number is the address of a byte. A group of bytes is addressed by the leftmost byte of the group. The number of bytes in a group is either implied or explicitly defined by the operation. The addressing mechanism uses a 16-bit binary address giving a maximum of 65,536 addressable bytes. Main storage is available from 8,192 bytes to 65,536 bytes in 8,192 byte increments. Main storage is partitioned into blocks of 512 bytes each. A main storage block is called a buffer.

Virtual storage is also partitioned into blocks of 512 bytes each. A block of virtual storage is called a frame. Frames are numbered consecutively starting with zero. Each number is the address of a frame. A frame address is also called a frame identification (FID). FID's are 24 bit binary numbers giving an addressing capacity of 16,777,216 frames or 8,589,934,592 bytes. Virtual storage is available in 9,744 frame (4,988,928 byte) increments.

All program references to information are references to virtual storage. Fields in virtual storage are referenced via a frame number and a displacement. If the field being referenced is a single byte or a string, the displacement is the number of bytes relative to the first data byte of the frame. If the reference is to a word, double word or triple word, the displacement is the number of words relative to the first data byte of the frame. References to instructions are via a 12-bit frame number. Therefore, programs must be located in the first 4,096 frames.

2.2. VIRTUAL MEMORY MANAGEMENT

The CPU directly accesses information from buffers in main storage. These buffers contain the contents of virtual storage frames. The virtual frames are moved between the disc and main storage as required by processes in progress. Two of the main storage buffers, 0 and Z, contain the monitor program that performs the actual operation of swapping frames in and out of main storage. Main storage buffers 1 and 3 contain information about each of the main storage buffers and a map of the frames currently contained in each main storage buffer.

2.2.1 BUFFER STATUS

Main storage locations X'200' through X'27F' contain the status of each of the main storage buffers. One byte is used for the status of each buffer. Location X'200' contains the status of buffer 0; location X'201' contains the status of buffer 1; and so forth. The information contained in the buffer status byte is given below:

Buffer Status Byte

Bit	
0	I/O BUSY/
1	CORELOCK1/
2	CORELOCK/
3	WRTREQD/
4	
5	
6	
7	

PSYM Name	Bit	Description
I/O BUSY/	0	Zeroed whenever an I/O (disc or peripheral) is in progress for this buffer; set when I/O completes. Firmware prevents "attachment" by a virtual process to a buffer with this bit zero.
CORELOCK1/	1	This bit is zeroed during cold-start tape generation, along with bit 2, if a buffer is to remain core-locked.

PSYM Name	Bit	Description
CORELOCK/	2	A zero indicates that this buffer may not be selected for disc input.
WRTREQD/	3	A zero indicates that data in this buffer has changed since it has been read from disc, and must therefore be written back to disc.
	4	Unused
	5-7	These bits are used by the 'FAR' instruction which changes the buffer status.

2.2.2 BUFFER MAP

Main storage locations X'280' through X'2FF' and locations X'700' through X'7FF' contain the addresses of the frames currently in the main storage buffers. The map is divided into two sections. Locations X'280' through X'2FF' contain the least significant byte of each of the frame addresses. Locations X'700' through X'7FF' contain the most significant two bytes of each of the frame addresses. For example, the virtual storage address of the frame in buffer 4 is found by concatenating the contents of main storage bytes X'708', X'709', and X'204'.

2.2.3 BUFFER QUEUE

A buffer queue is maintained by the firmware in main storage locations X'300' through X'3FF'. The buffer queue consists of a doubly linked list of buffer numbers ordered according to their time of attachment by the firmware. Each time a register is attached to a buffer, the firmware moves the attached buffer to the head of the buffer queue.

When the contents of a buffer must be replaced because of a frame fault, the buffer queue is used to identify the least recently attached buffer for replacement. The buffer used is then moved to the head of the buffer queue.

Each entry in the buffer queue consists of two bytes. The word (two bytes) displacement of the entry from main storage location X'300' corresponds to the buffer number. The two bytes forming each entry in the buffer queue are the two pointers forming the doubly linked list.

The first byte of each entry points to the next more recently attached buffer entry in the queue. The second byte of each entry points to the next less recently attached buffer entry. The first byte of the most recently attached entry (i.e., head of the queue) contains X'FF'. The second byte of the least recently attached entry (i.e., tail of the queue) contains zero.

Bytes at locations X'300' and X'301' contain pointers to the head and tail of the buffer queue respectively. The head of the queue identifies the most recently attached buffer number. The tail of the queue identifies the least recently attached buffer number.

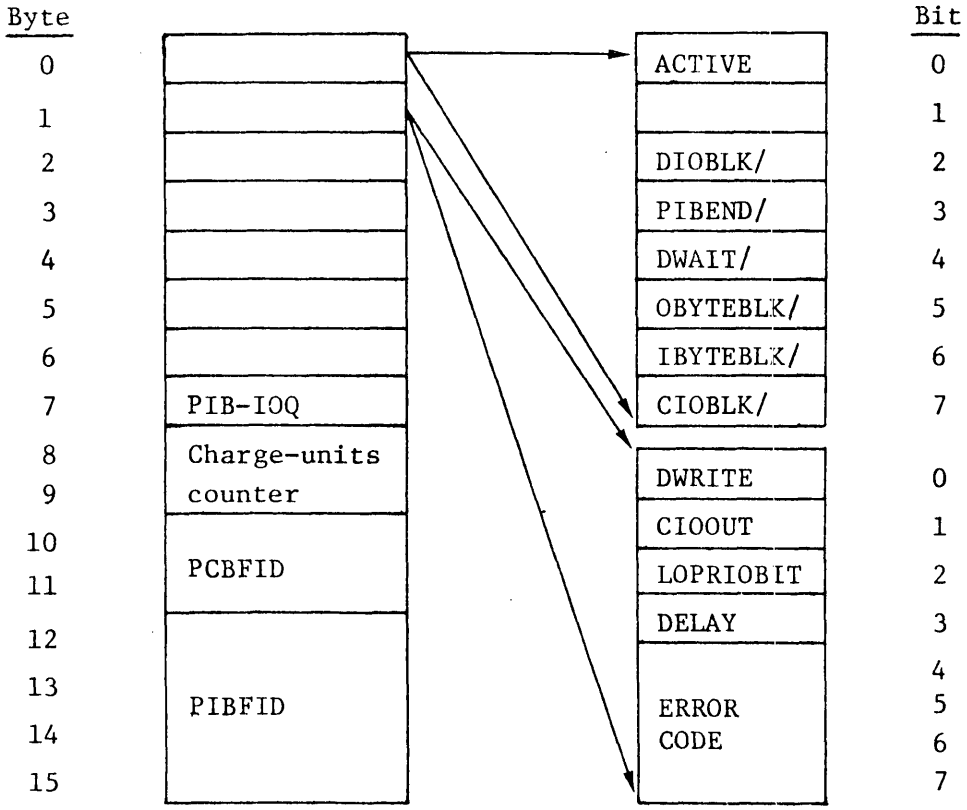
2.3 PROCESS

The Reality CPU is designed as an interactive system capable of communicating with several users simultaneously. A user communicates with the system via a communication terminal such as a Teletype or CRT terminal. Associated with each terminal is a process. A process is not an element of the system but rather a continuing operation on a set of functional elements. Refer to Section 2.8 for peripheral I/O details.

2.3.1 PROCESS IDENTIFICATION BLOCK

For each process attached to the system, there is a Process Identification BLOCK (PIB). Each PIB is 32 bytes long. The PIB for terminal zero is in main storage locations X'800' through X'81F'; locations X'820' through X'83F' contain the PIB for terminal one; and so forth. The PIB contains information about the status of the process with which it is associated. The following is a description of the PIB contents. Bytes 0 through 6 are determined by firmware.

PIB Status Bytes



PIB Status Bytes

Name	Byte	Bits	Meaning
ACTIVE	0	0	One indicates that process may be activated (candidate for Select Next User process).
--	0	1	Unused
DIOBLK/	0	2	Zero (zeroed by firmware on a frame fault) indicates that process is roadblocked waiting for referenced frame to be input from virtual storage. Set to one when monitor accepts request by moving FID onto IOQ.
PIBEND/	0	3	Zero indicates the end of the PIBs.
DWAIT/	0	4	Zeroed by monitor when frame fault request is accepted (FID moved to IOQ). Set to one when disc transfer is complete.
OBYTEBLK/	0	5	Zeroed by firmware when process is roadblocked waiting for terminal to complete output. Set to one when output is complete.
IBYTEBLK/	0	6	Zeroed by firmware when process is roadblocked waiting for terminal to complete input. Set to one when input is complete.
CIOBLK/	0	7	Zeroed by monitor when process is roadblocked waiting for concurrent I/O block transfer to complete. Set to one when block transfer is complete.
DWRITE	1	0	One indicates read request is roadblocked waiting for buffer to be written out to disc. Zeroed when output is complete and read request has been replaced in IOQ.
CIOOUT	1	1	One indicates process is roadblocked waiting for concurrent output to complete. Zeroed when output is complete.
LOPRIOBIT	1	2	Set to one on a Release Quantum entry to monitor, if the process does not have either a byte input or byte output roadblock. Also set to one during concurrent block output.
DELAY	1	3	One indicates a one-cycle delay to the Select Next User process, on a Release Quantum entry to monitor.

Name	Byte	Bits	Meaning
ERROR CODE	1	4-7	Software generated error trap codes: 08 - illegal FID 09 - disc error 0C - register zero detached 0E - charge-units counter overflow
--	2	0-7	Last byte address of PIB I/O buffer (bytes 16-31 of PIB).
--	3	0-7	Number of bytes in PIB I/O buffer less one (X'FF' = no bytes).
--	4	0-7	Mask byte used by Communications controller.
--	5	0-7	"Unusual status" of Communications Controller line associated with this PIB.
--	6	0-7	Data byte received from Communications controller line associated with this PIB.
PIB--IOQ	7	0-7	Pointer connecting PIB to IOQ entry.
Charge-units counter	8-9	0-15	Number of charge-units associated with this process.
PCBFID	10-11	0-15	Frame-id of the Primary Control Block (PCB) for this process.
PIBFID	12-15	0-31	When a process is roadblocked because of a frame fault, the frame-id is placed in these bytes. When the monitor is entered as a result of a call operation, these bytes contain parameters:

			Frame I/O Request	Monitor Call
--	12	0-7	Buffer number which contains PCB.	High-order address of PCB frame
--	13	0-7	High-order byte of absent FID.	Mask byte from Call instruction.
--	14	0-7	Middle byte of absent FID.	High-order byte of address of register referenced in Call.
--	15	0-7	Low-order byte of absent FID.	Low-order byte of address of register referenced in Call.

--	16-31		Input/output buffer for the terminal associated with this process.

2.3.2 PRIMARY CONTROL BLOCK

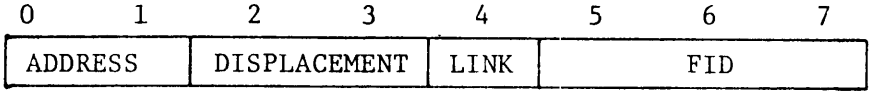
For each process there is a frame called the Primary Control Block (PCB). The PCB contains the accumulator, address registers, subroutine return stack and string scan control characters associated with the process. The location of the PCB is contained in the PIB of the process. The following paragraphs describe the contents of the PCB. The bytes that are not described are not accessed by the Firmware. However, the remaining bytes of the PCB contain information used by the operating system.

Bytes	Description
0	This byte is reserved for a lock code used for storage protection.
1	This byte contains the condition code resulting from a previous arithmetic instruction execution.
3-5	These bytes are used for controlling the Move and Scan through Delimiter instructions.
6-7	These bytes are used for controlling the debug trace mode of operation.
8-X'0B'	These bytes contain the double word accumulator extension. The accumulator extension contains the most significant portion of a product after a multiply operation. It contains the remainder after a divide operation.
X'0C'-X'0F'	These bytes contain the double word accumulator.
X'100'-X'17F'	These bytes contain the 16 address registers. See the description of the address registers below.
X'180'-X'181'	These bytes contain the address (relative to byte zero of the PCB) of the limit of the subroutine stack.
X'182'-X'183'	These bytes contain the pointer to the current top of the subroutine stack.
X'184' and above	The bytes contain the subroutine return stack. The number of bytes allocated for the stack is determined by the contents of bytes X'180' and X'181'.

Address Registers

All references to data, except immediate data, are made indirectly through an address register. There are 16 address registers in each PCB. Each address register contains 8 bytes:

Address Register Format



Bytes	Description
0-1	These bytes contain the 16 bit main storage address of the referenced data. If the address is less than X'800', the frame containing the data may be absent from main storage.
2-3	These bytes contain the displacement of the referenced data relative to the first data byte of the frame. The displacement is a 16-bit signed number. Negative values are represented in two's complement form. These bytes are meaningful only when the register is detached. (See Register Attachment below.)
4	Zero in bit zero of this byte indicates that the register references data in the linked format. If bit zero is a one, the register references the data in the unlinked format. One in bit one indicates that frame attachment is in progress. Bit one can only be set during the execution of instructions that increment addresses with data movement.
5-7	These bytes contain the virtual storage frame number of the byte being referenced.

Address Register Attachment

When a program loads ("restores") an address register, the first two bytes of the register are set to zero. Bytes 2 through 7 of the address contain a virtual frame number and displacement. A register in this format is said to be detached. When a subsequent instruction uses the detached register for a data reference, an attempt is made to convert the address register to the attached format. The attaching attempt is automatic and performed as follows. The buffer map is scanned to determine if the referenced frame is located in main storage. If the frame is in main storage, the location of the required byte is computed by adding the buffer address from the map to the displacement from the address register. The address is then

placed into bytes 0 and 1 of the address register, thus forming the attached format. Once the register is attached, instruction execution takes place.

If the referenced frame is not in main storage, the frame number is placed into bytes 12 through 15 of the PIB. Byte 0, bit 2 of the PIB is set to 0, thus roadblocking the process. Next all of the address registers in the PCB are converted to detached format and a fault interrupt to the monitor is taken.

Address Register Zero

Register zero is used in a special way. This register always contains the FID of the PCB. Register zero is attached when the process is activated. The displacement field of this register is always assumed to be zero.

Address Register One

When a process is not active, address register one contains the FID and displacement (minus one) for the next instruction to be executed. When the process is activated, the buffer address of the program frame (as determined from the buffer map) is added to the displacement from register one. This value is placed into a hardware instruction counter. The register is then converted to the attached form with the buffer address set to the base address (byte zero) of the program frame. When the process is deactivated, the main storage location from the instruction counter is converted to the corresponding FID and displacement and the register is detached with these values placed into it.

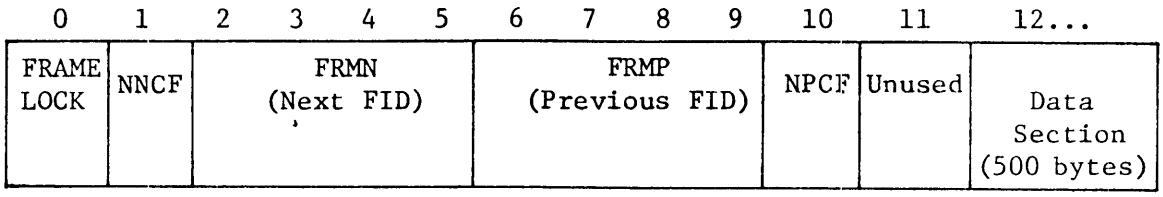
2.3.3 FRAME FORMATS

The Reality system recognizes two types of frame formats; linked and unlinked. In both formats byte zero of the frame is reserved for a frame lock.

Unlinked frames contain 511 data bytes. For unlinked frames the displacement portion of an address is relative to byte 0 of the frame, i.e., a displacement of 1 is a reference to the first data byte. Displacements outside the range 0 through 511 are not valid for frames in the unlinked format.

Linked frames contain 500 data bytes. For linked frames, the displacement field in the address is relative to byte 11 of a frame. However, a displacement of zero is a reference to byte 511 of the frame to the left of the current frame. Displacements for linked

frames may be positive or negative so long as the displacement references a logically linked item of data. The following paragraphs describe the linked format.



Linked Frame Format

Bytes	Description
0	This byte is reserved for a frame lock.
1	This byte contains a count of the number of next contiguous frames to the right of this frame (NNCF). A zero in this byte indicates that this frame is the rightmost frame in a contiguously linked set of frames.
2-5	This field contains the frame number of the frame that is logically to the right of this frame. If byte 1 contains other than zero, the frame to the right is the next higher numbered frame. If byte 1 contains a zero the frame to the right may be any frame number. A zero in this field indicates that this is the rightmost frame of a linked set.
6-9	This field is similar to bytes 2 through 5 except that it contains the number of the frame to the left of this frame.
10	This byte is similar to byte 1 except that it contains a count of the number of previous contiguous frames to the left of this frame (NPCF).
11	Unused.
12-511	Data section.

2.4 THE MONITOR

The monitor is a program that is an integral part of the Reality system. The monitor process is the only one not associated with a PIB. The PCB for the monitor is defined as buffer 0 of main storage.

The function of the monitor is to initiate the transmission of information between main storage buffers and virtual storage and to schedule each of the processes.

When the system is operating in monitor mode, address registers are not checked for attachment. Instead all data references are assumed by the firmware to be references to absolute core addresses. The system is in monitor mode whenever the location of the PCB is at core-address zero.

The monitor gives control to another process by executing either a Resume Virtual Process or a Start Virtual Process instruction.

The multi-disc monitor may be used with one or two drives per controller, and with one through four disc controllers. In any multi-disc configuration, the capacity of every drive in the system must be the same; i.e., either 5 megabyte or 10 megabytes/drive; also all controllers must have the same number of drives attached to them.

2.4.1 MONITOR PCB

The PCB associated with the monitor is at absolute core-address 0 through X'1FF'. Beside the functional elements that are described in Section 2.3.2, the following locations are used:

Bytes	Description
2	Contains the Interrupt Address code on an External interrupt fault trap to the monitor.
3	Contains monitor status flags (bits).
6	Contains the hardware clock counter; a fault is generated when this is incremented (every one millisecond) to zero.
7	Extension of clock counter used by the monitor.
X'10'-X'1F'	Exclusively a hardware save area.
X'20'-X'FF'	Contains the bootstrap software executable code.
X'1A0'-X'1A1'	Contains the system date (days since 31 Dec 1967).
X'1A4'-X'1A7'	Contains the system time (seconds since midnight).
X'1C0'-X'1DF'	Contains PIB pointers for peripheral devices 0 through 15.
X'1E0'-X'1FF'	Contains address pointers for peripheral devices 0 through 15.

Initial Condition of Monitor PCB Registers

LOC									
100	R0	00	00	03	05	BITS			
108	R1	04	00	01	C0	01	E0	02	7F
110	R2	06	00	14	80	10	F0	0C	84
118	R3	0C	00	FF	FF	20	A0	28	3F
		PIBWA		PIBSTART		PIBSIZE		PFID	
120	R4	--	--	07	E0	00	20	--	--
128	R5	01	--	F6	--	FE	64	06	3F
130	R6	01	60	--	--	00	01	51	81
		IOQWAL		IOQSTART	IOQSIZE	IOQMAX	IOMAX	IPQ#	NUMCONT
138	R7	06	--	*	10	*	*	00	*
		DCTWAL				FIDMAX			
140	R8	06	--	--	04	*			
				=H24					
148	R9	00	0B	0C	18	02	00		
150	R10	--	--	--	--	--	--	--	--
158	R11	0E	--	--	--	--	--	--	--
160	R12	--	--	--	--	--	--	--	--
168	R13	--	--	--	--	--	--	--	--
170	R14	--	--	--	--	--	--	--	--
178	R15	--	--	--	--	--	--	--	--
180		--	A0	--	84	FF	FF	FF	FF

-- : Unused or scratch

* : Preset by MSETUP Program.

Monitor Register Assignment

Register No.	PSYM Name	Address	Description of Usage
0	None	X'0000'	Addresses Monitor PCB.
1	None	X'0400'	Addresses MMONITOR.
2	None	X'0600'	Addresses MMONITORX.
3	None	X'0C00'	Addresses MMONITORY/Nx.
4	PIB	Variable	Current PIB pointer.
5	None	X'0100'	Addresses Monitor PCB, lower half.
6	None	X'0160'	Addresses WA of R12.
7	IOQ	X'06xx'	Current IOQ entry pointer.
8	DCT	X'06xx'	Current DCT entry pointer.
9	None	X'000B'	Addresses H4 in Monitor accumulator.
10	None	--	Scratch
11	None	X'0E00'	Addresses MMONITORZ.
12-15	None	--	Scratch

2.4.2 INTERRUPTS AND MONITOR CALLS

Once a virtual process gains control, it remains in control until the occurrence of an interrupt or until the process executes a Monitor Call instruction. The occurrence of a Monitor Call instruction will cause all registers in the current PCB to be converted to the detached form.

There are three types of interrupts to the monitor: external, internal, and fault.

An external interrupt is generated when a device (including the virtual storage device) completes an operation. When an external interrupt (excluding the virtual storage device) occurs, the status of the active process is saved in the hardware. The process that was interrupted must be resumed by executing a Resume Virtual Process instruction. After the occurrence of an external interrupt, further external interrupts will be inhibited until a Resume Virtual Process instruction has been executed by the monitor. Refer to Section 2.4.4 for an explanation of external interrupts from the Virtual Storage Device. Device addresses 0-X'F' are assumed to be non-virtual storage devices and X'10' - X'17' are assumed to be virtual storage devices.

When an internal interrupt occurs with a dependent process active, all registers in the PCB are converted to detached form and control passes to the monitor. An internal interrupt can be recognized with the monitor active at any time except in the case of a real time clock runout which can only be recognized after the execution of a Test Interrupt instruction (X'01').

A fault interrupt can occur only when a virtual process is active. A fault interrupt causes all the registers in the PCB to be converted to detached form.

The monitor must execute a Start Virtual Process instruction to start a process that was interrupted by an internal or fault interrupt.

Interrupts cause entry to the monitor at predefined locations. The table below shows the monitor entry point for each interrupt condition.

Entry Address	Interrupt Condition
1	Reference to absent frame (fault)
3	Input/output operation complete (external)
5	Power fail console interrupt, or clock runout (internal)
7	Terminal input/output with device not ready, or attempt to attach a buffer with input or output active in the buffer (fault)
9	Attempt to attach register 0 when not in the monitor (fault)
11	Power restored entry point (internal)
13	Hardware abnormal condition while in Monitor mode
15	Not used
17 thru 31	Monitor Call instruction entry points

Traps

Certain operations can cause a trap condition to be signaled. The occurrence of a trap causes a Branch and Stack location instruction

to be executed to a predefined location in virtual storage frame one. The table below shows the entry point and the cause for each trap.

Entry Address	Cause
1	Illegal operation code encountered.
3	The return stack is empty. This occurs when a Return instruction is executed with the current stack position pointing to the beginning of the return stack.
5	The return stack is full. This occurs when a stack location type of operation is executed and the current stack position is equal to the end of stack location. The current stack location is reset.
7	Attempt to reference frame 0 when not in monitor mode. The number of the address register that contained the reference is placed into the condition code byte of the PCB.
9	Attempt to cross a frame boundary for an unlinked frame or a word, double word, or triple word not entirely in one frame. The register number containing the reference is placed into the condition code byte of the PCB.
11	Attempt to link across a frame with a forward link of zero. The register number containing the reference is placed into the condition code byte of the PCB.
13	Attempt to link across a frame with a backward link of zero. The register number containing the reference is placed into the condition code of the PCB.
15	Attempt to execute a privileged opcode when not in monitor mode.
17	Attempt to reference a non-existent frame.
19	Disc error.
21	Break key activated on the terminal.
23	Return stack format error. There are two conditions that cause this error. Either the end of stack location is less than the current stack position, or the stack size is defined for less than 7 entries.
31	Debug trace mode. This is not an error condition. Trace mode is controlled by bytes 6 and 7 of the PCB.

Trace Mode

The Reality CPU can operate in a special mode called trace mode. When trace mode is in effect the hardware monitors the instruction execution and traps to location 31 of frame 1 on the occurrence of certain conditions. When a trap occurs, bits 0 through 3 of byte 6 of the PCB are set to zero, inhibiting further traps. The conditions that can cause the trap are defined in bytes 6 and 7 of the PCB. The following list shows the conditions:

Byte 6, Bit 1 - This bit indicates that a trace interrupt is to occur on every BSL, ENT, BSL*, or ENT* instruction (modal trace).

Byte 6, Bit 3 - This bit indicates that trace traps can occur on every instruction.

Byte 6, Bit 5 - This bit is set by firmware when an instruction trace trap has occurred.

Byte 6, Bit 6 - This bit is set by firmware when a RTN trace trap has occurred.

Byte 6, Bit 7 - This bit is set by firmware when a BSL, ENT, ENT* or BSL* trace trap has occurred.

Byte 7 - When the system is in the instruction trace mode, this byte is incremented for every instruction executed. A trace trap will not occur until this byte has been incremented to zero.

2.4.3 MONITOR DISC SCHEDULING TABLES

There are two tables that control the disc I/O scheduling: the I/O Queue (IOQ) and the Device Control Table (DCT). The IOQ table may be considered a subset of the Process Identification Block set (PIB); a process that requires disc input must first be allotted a spot in the IOQ before its request can be honored. Since the IOQ can be set to any size between two and eight entries, the IOQ acts as a funnel between the disc input requests from the processes, and the actual disc I/O. By preventing the honoring of requests for too many processes in rotation, the IOQ serves to control "thrashing".

The I/O Queue (IOQ)

Moving on and off the IOQ is controlled by the monitor in the following manner: When a process requests disc input, and it is not on the IOQ, it is moved on to an available IOQ entry if such an entry exists at the time; if not, the process goes into a wait state until an entry

becomes available. A process moves off the IOQ ("deactivates") under the following conditions:

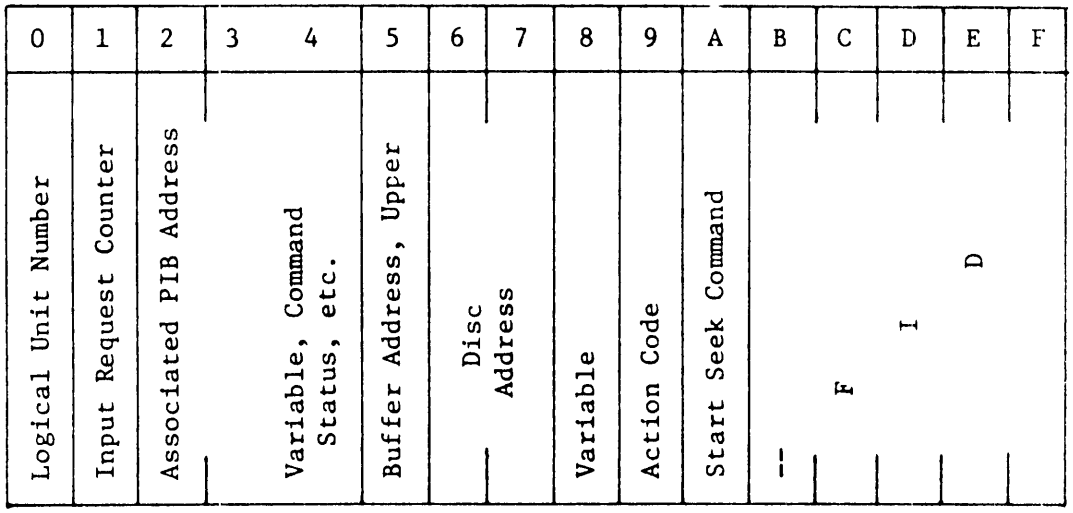
1) The process executes a "Release Quantum", either explicitly (via the RQM instruction) or implicitly due to:

   - Real Time Clock Interrupt
   - Terminal input or output roadblock
   - Concurrent I/O block transfer roadblock

2) The process executes the maximum number of disc input requests, as specified by the monitor parameter IOMAX.

In any of the above cases, that process is taken off the IOQ; the entire PIB table is then searched for other processes that are roadblocked due to a disc input request, and one of them selected to be moved to the ICQ, at which time the disc input request counter (IOCTR) is set to the initial value specified in IOMAX.

IOQ Table Format



Byte	PSYM Name	Description
0	None	Logical unit number; consists of the controller device address, with the high-order bit specifying the drive number:

Controller Address -		14	15	16	17
Drive Number	0	14	15	16	17
	1	94	95	96	97

Byte	PSYM Name	Description
0 (Continued)	None	This byte is set up when the disc address is computed from the FID; is set to zero when the disc request has been processed.
1	IOCTR	Disc input request counter; is set to value specified in IOMAX when a process is moved to the IOQ; is decremented on every input request processed. If zero specifies an available IOQ entry.
2-3	IOQ--PIB	Link from IOQ to PIB; is set up when a process is moved to the IOQ; is never zeroed.
4	IOQCOM1	Scratch location used to communicate with the disc controller may store a RETURN command, a SELECT and QUEUE SEEK command, or the major status.
5	IOQBUF	Buffer address, (upper) to which disc I/O is being done; the low-order bit is first zeroed to output the "buffer start, upper" command, to the disc controller, then set to output the "buffer end, upper" command.
6-7	IOQDA	Disc address computed from the FID.
8	None	Used as a scratch location to output the "buffer start, lower" and "buffer end, lower" bytes (always X'00' and X'FF' respectively).
9	IOQACT	Controller action code: X'00' - Read X'01' - Verify X'02' - Write
A	None	Controller command--always X'90' (start queued seek(s), arm interrupt).
B	-	Unused
C	IOQFIDO	FID, uppermost byte is always X'80'.
D-F	IOQFID	

Selection of a Process to be placed on the IOQ

The following rules are used to select a process to be placed on the IOQ (if more than one process is roadblocked due to a disc input request):

1) If the process had been taken off the IOQ due to a terminal I/O roadblock, it is selected to be placed on the IOQ immediately. This is governed by bit LOPRIOBIT being zero in the PIB.

2) If none of the processes are so roadblocked, the first process with LOPRIOBIT set (therefore lower priority) is selected to be placed on the IOQ.

The effect of this selection criterion is that processes that had moved off the IOQ due to terminal I/O will have a higher priority than processes that do not do any I/O. In order that the latter type of processes do not get into a state where they may never be selected due to the existence of other processes that are in a heavy terminal I/O state, LOPRIOBIT is zeroed during the search described above.

Note that in the event that only one process requires disc input, a full search of the PIB table is completed, before the deactivated process is moved back to the IOQ.

IOQ Setup

When a process is placed on the IOQ due to a disc input request, the following sequence of events occurs:

1) The disc input roadblock (DIOBLK/) in the PIB status is set, thereby indicating that this process is on the IOQ, and is no longer a candidate for IOQ selection.

2) The "waiting for disc" flag (DWAIT/) is zeroed, thereby preventing the Select Next User routine from selecting that process for execution.

3) The requested FID is moved from the PIB to the IOQ, and the disc address is computed.

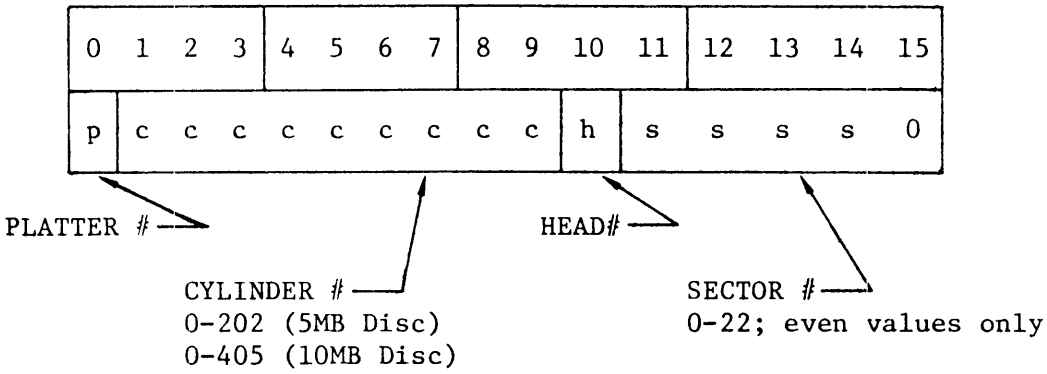
4) a) If the addressed disc is busy, nothing further can be done.

   b) If not, the 'SETUP' routine in the disc interrupt handler is entered.

Disc Address Computation

Subroutine SETIOQ takes the FID specified in the IOQ entry; checks against a maximum FID as specified in the literal FIDMAX; converts the FID to a 16-bit disc address and stores the latter at IOQDA.

Disc Address Format

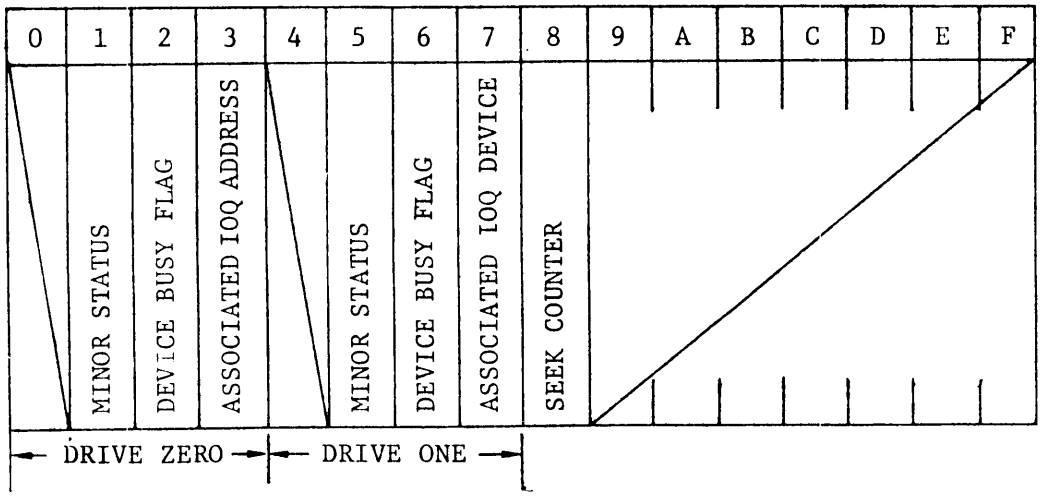


The Device Control Table (DCT)

The DCT entry is uniquely associated with a disc drive; it contains a flag indicating whether the device is busy or not, a link to the IOQ associated with the drive, and in the case of two drives per controller, a seek counter which keeps track of the number of seeks that have been started on the controller.

The DCT location is mapped directly from the device address of the controller, and the disk drive number; its main function is to provide an easy linkage from the interrupt address supplied by the CPU at the time of an interrupt, to the IOQ.

DCT Table Entry (Per Controller)



Byte	PSYM Name	Description
1	None	Minor status from drive if an error has occurred; is not reset if no errors.
2	DCTBSY	High-order bit, if set, indicates drive is busy. Reset on I/O completion.
3	DCT--IOQ	Address link from DCT to associated IOQ Entry; is setup when seek is started; is not reset.

2.4.4 DISC INTERRUPT HANDLING

On receiving an end-of-transfer interrupt, the firmware deactivates the currently executing virtual process (if in virtual mode), and traps to location X'403' in the monitor. The interrupt address of interrupting device (device address times two) is stored at X'0F' - the low-order byte of the monitor accumulator. A virtual process cannot resume execution after completion of disc interrupt handling, since its buffers may be replaced and attached registers may no longer be valid.

The interrupt address is mapped into the DCT address, which leads to the associated IOQ address, which in turn leads to the associated buffer address, FID, and PIB address. The status of the drive is obtained and, if there are no errors, the controller is re-armed (if another seek is pending it is also re-started). If the completed operation was a write, a verify is now started, and interrupt handling terminates.

Selection and setup of next I/O

Since the drive is now ready, the IOQ table is searched for a matching device/drive number, or logical unit number. If a match is found, the setup phase is entered (also entered from frame fault).

1) The FID is picked up from the IOQ, and a FAR instruction executed; this is the only monitor-level instruction that causes the firmware to attach an A/R. If the requested FID is core-resident, the disc read roadblock is removed from the associated PIB, and the IOQ table searched for the next I/O.

2) If the FID is not core-resident, the execution of the FAR instruction has automatically caused an attachment to the "oldest" buffer in core. If this is core-locked, the FAR is repeated. If not, and if the buffer has no write-required flag on its status, an available spot has been found for the disc read, and the read parameters are set up.

3) If a write-required flag exists on the buffer status, that data must be written out before the read request can be processed; therefore, the IOQ entry is overwritten with the FID to be written out, and the write parameters are set up in the IOQ. Also, the flag DWRITE in the PIB status is set, indicating that the read request from the process is yet to be processed. Note that in this case a drive other than the one that just completed a transfer may be started.

Starting I/O

In the case of two drives per controller, a RETURN instruction is issued to the controller if another seek is pending completion (DSCSEEKS non-zero); if the controller status indicates that it did not return, the start I/O sequence is aborted, and the associated process arbitrarily reactivated by clearing its PIB roadblocks. This is because the controller either:

- is transferring data at this time, in which case it cannot be interrupted to queue another seek, or
- it has completed a transfer, in which case an interrupt is pending recognition and another seek cannot be queued.

When reading a frame, the buffer FID is set to zero till the read completes; when writing a frame to the disc, the FID of the buffer remains unchanged. In either case, the buffer status is set core-locked and I/O busy for the duration of the I/O.

Disc Errors

If a disc error is detected at the completion of the transfer, the minor status from the drive is stored in the DCT; the buffer is set non-core-locked, not-I/O-busy, the associated process PIB roadblock is cleared just as if the I/O had completed, and error #9 is set in the PIB error byte. This causes a virtual software trap to the DEBUG State, and an eventual re-stacking of the request. Note that the buffer FID is maintained if the transfer was a write, and is left as zero if the transfer was a read.

If an illegal FID is requested (as determined by comparing against FIDMAX in the monitor PCB), as above, except that error #8 is flagged. The DEBUGGER will abort the process in this case.

The Select Next User (SNU) Routine

The Select Next User routine (SNU) is entered whenever the monitor has completed setting up a disc transfer; has completed processing of a disc interrupt; or is in a "wait" state due to all virtual

processes being quiescent. While the monitor is in the SNU routine, it cycles through the PIB's to see if any of the processes require activation. At this time also, the monitor executes the Test Interrupt instruction, which is the only monitor-executed instruction that allows recognition of external interrupts.

A process may be selected under the following conditions:

- All roadblocks clear; that is, if DIOBLK/, DWAIT/, OBYTEBLK/, IBYTEBLK/, CIOBLK/ are set, and DELAY and DWRITE are zero.

- External activation and disc roadblocks clear; that is, if ACTIVE, CIOBLK/ and DWAIT/ are set. This would happen if the process received a BREAK-key interrupt, or if another process sent this one a message via the MESSAGE processor.

In either case, the monitor sets up to activate the process by loading the PCB-FID from the PIB into R4FID (of the monitor), and executing a Start Virtual Process instruction.

If the process had been roadblocked due to a disc input request, and the monitor had overwritten this request with a disc write, all status bits are as in (1) above, except that DWRITE is set. In this case, the monitor will cause the read request to be re-stacked, by re-entering the Frame Fault entry point.

Programming Notes

The current IOQ entry is addressed by the address register "IOQ" (R7); the current DCT entry by the A/R "DCT" (R8); in the case of two discs per controller, R12 addresses byte zero of the DCT block for the current controller.

Registers DCT and IOQ always work in the same 256-byte block (X'600 - X'6FF'); therefore, the upper bytes of their address words are preset to X'06' (as an initial condition when cold-starting the system), and only their low order address bytes are altered. Also, the IOQ table starting address pointer (IOQSTART) is a half-tally.

The DCT location is from X'640' through X'67F' (see formats later); sixteen bytes are allocated to each of four possible controllers, (with device addresses X'14' through X'17' respectively), with one or two drives per controller.

The IOQ table has a fixed ending address--the last entry is X'6F0' through X'6FF'; the starting address is variable, depending on the

number of entries allowable. The table beginning pointer (IOQSTART) is 16 bytes before the first IOQ entry:

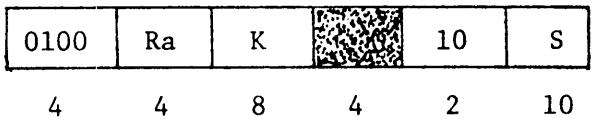
IOQSTART=X'700' - 16* (IOQMAX+1)

where IOQMAX is in the range 2 through 8 inclusive.

2.5 MACHINE INSTRUCTIONS

This section lists all Reality machine instructions and describes their execution. A diagram representing the format is given with each instruction description. Preceding the diagram is the name of the instruction. Enclosed in parentheses is the assembler code for the instruction. It should be noted that the assembler codes are not unique. That is, several of the instructions have the same code. The assembler uses both the code and the operand attributes to determine a particular operation (see Section 3). Below is an example of an instruction description:

Branch Byte Equal to Immediate (BCE)



The numerical operation code bits are shown as binary numbers. Note that the operation code need not occupy consecutive bit positions of an instruction. In the example above the operation code occupies the first and fifth fields of the instruction. The numbers appearing beneath the diagram indicate the number of bit positions occupied by the particular field. The symbols appearing in the diagram indicate the type of information in the field. Shading indicates that the field is not used in the instruction. The table below defines the symbols used in the instruction diagrams:

Symbol	Meaning
R	The field contains an address register number.
D	The field contains a displacement relative to the contents of an address register or relative to the beginning of a frame.
FID	The field contains a frame identification number. If the field is less than 24 bits wide, high order zeroes are assumed.
S	The field contains a signed magnitude skip distance (in bytes) for conditional skipping. A skip distance of zero means no bytes are to be skipped.

Symbol	Meaning
L	The field contains an operand length. L = 0 is a 1 byte operand. L = 1 is a 2 byte operand. L = 2 is a 4 byte operand. L = 3 is a 6 byte operand.
K	The contents of the field itself is an operand.

Terms used in the instruction descriptions are defined as follows:

- Ra or Rb means the contents of the addressing register named by the Ra or Rb field of an instruction.
- C(Ra) or C(Rb) means the contents of the location referenced by the address contained in the named addressing register.
- C(Ra,Da) or C(Rb,Db) means the contents of the storage frame location referenced by adding the D field of the instruction to the contents of the named storage register.
- When a register or part of a register is cleared, the cleared part contains zero bits.
- When the word 'load' is used in a description it means the contents of some frame location replaces the contents of a special register (address register or accumulator).
- When the word 'store' is used in a description it means the contents of a special register or the contents of an instruction field replaces the contents of some frame location.
- When the word 'move' is used in a description it means that the contents of a frame location replaces the contents of another frame location, or the contents of a register replaces another register.

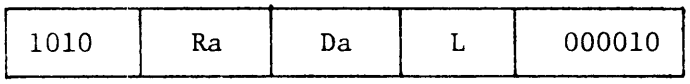
Storage operands are always referenced through one of the 16 addressing registers. An addressing register contains the byte address of the operand. For instructions with a D field, a displacement is added to form an effective address. When the operand is a single byte (L field = 0), the D field of the instruction is the displacement. When the operand is a word, double word or triple word (L = 1, 2 or 3) the D field is doubled to form the displacement.

2.5.1 ARITHMETIC OPERATIONS

The following operations perform arithmetic on binary integers. Negative values are represented in two's complement form. The 'L' field of the instruction specifies the length of the operand in storage. For storage to accumulator operations, triple word operands are not allowed (L field of 3); byte and word operands are sign extended to form a double word value before the operation is performed. The accumulator operand is always a double word. Storage operands must lie entirely in a single frame. The condition codes resulting from an arithmetic operation are placed in byte 1 of the PCB. The condition codes are defined below.

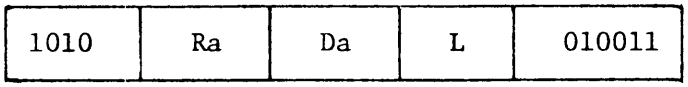
Symbolic Name	Bit Position	Condition Indicated
ZROBIT	5	zero result
NEGBIT	6	negative result
OVFBIT	7	arithmetic overflow

Test and Set Arithmetic Condition Flags (TST)



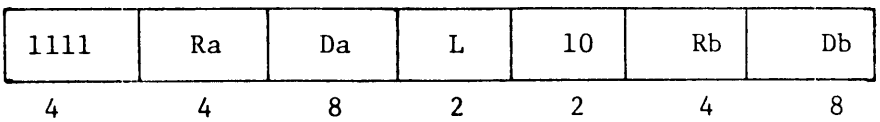
The contents of (Ra,Da) is tested and the arithmetic condition flags (i.e., ZROBIT and NEGBIT) are updated appropriately. The instruction may be used with a half tally, a tally, or a double tally. L is '00', '01', or '10' respectively depending upon the type of operand.

Add to Accumulator (ADD)



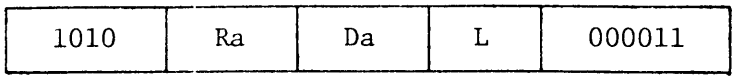
The C(Ra,Da) are added algebraically to the accumulator. The sum is placed in the accumulator. The C(Ra,Da) are unchanged.

Add to Storage (INC)



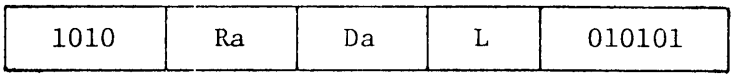
The C(rb,Db) are added algebraically to the C(Ra,Da). The sum is placed in the C(Ra,Da). The C(Rb,Db) are unchanged.

Add a One to Storage (INC)



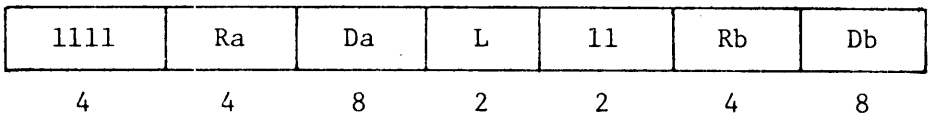
The C(Ra,Da) are algebraically increased by 1.

Subtract from Accumulator (SUB)



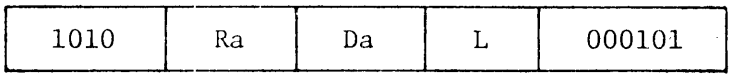
The C(Ra,Da) are algebraically subtracted from the accumulator. The difference is placed into the accumulator. The C(Ra,Da) are not changed.

Subtract from Storage (DEC)



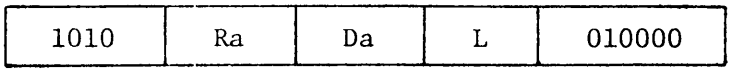
The C(Rb,Db) are algebraically subtracted from the C(Ra,Da). The difference replaces the C(Ra,Da). The C(Rb,Db) are not changed.

Subtract One from Storage (DEC)



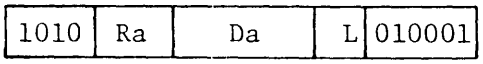
The C(Ra,Da) are algebraically decreased by 1.

Multiply (MUL)



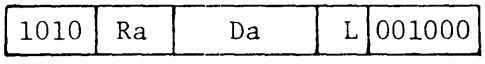
The contents of the accumulator are multiplied by the C(Ra,Da). A 64 bit product replaces the contents of the accumulator and accumulator extension. The sign of the product is determined by the rules of algebra. The C(Ra,Da) are not changed.

Divide (DIV)



The sign of the accumulator is replicated into the accumulator extension to form a 64 bit dividend. The C(Ra,Da) are divided into the dividend to form a 32 bit quotient and a 32 bit remainder. The quotient replaces the contents of the accumulator and the remainder replaces the contents of the accumulator extension. The sign of the quotient is determined by the rules of algebra. The sign of the remainder is the sign of the dividend. The C(Ra,Da) are not changed.

Negate (NEG)



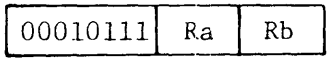
The sign of the C(Ra,Da) is changed.

2.5.2 DATA TRANSMISSION OPERATIONS

The following operations are concerned with the transmission of data between storage locations, between registers, and between registers and storage locations.

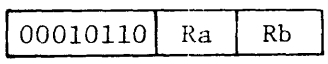
For instructions in this group, operands in storage must be within a single frame.

Exchange Address Registers (XRR)



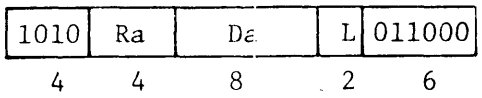
Ra and Rb are exchanged.

Move Address Register to Address Register (MOV)



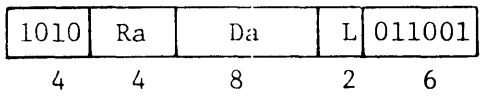
Ra replaces Rb. Ra is not changed.

Load Accumulator (LOAD)



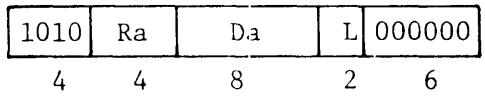
The C(Ra,Da) replace the contents of the accumulator. The C(Ra,Da) are not changed. The accumulator is sign extended to form a double word value.

Store Accumulator (STORE)



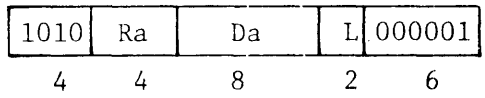
The contents of the accumulator replaces the C(Ra,Da). The contents of the accumulator is not changed.

Store a Zero (ZERO)



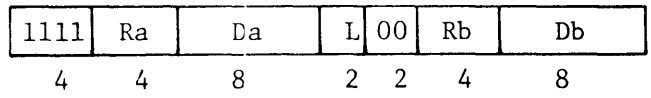
The C(Ra,Da) are replaced with zeros.

Store a One (ONE)



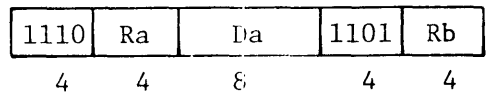
The C(Ra,Da) are replaced with a 1.

Move (MOV)



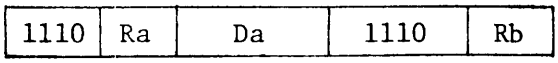
The C(Rb,Db) replace the C(Ra,Da). The C(Rb,Db) are not changed.

Store Address Register (MOV)



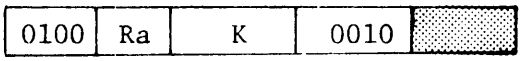
The detached form of Rb is stored into the C(Ra,Da). A triple word is stored. Rb is not changed. Da is doubled to form the effective address.

Load Address Register (MOV)



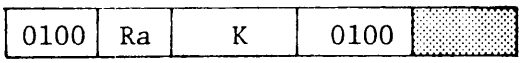
The C(Ra,Da) replace the 6 low order bytes (displacement and FID) of Rb. The high order byte of Rb (buffer location) is set to zero. The C(Ra,Da) are not changed. Da is doubled to form the effective address.

Move Immediate Character (MCC)



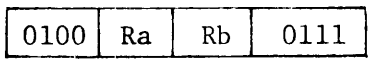
The byte, K, replaces the C(Ra).

Increment and Move Immediate Character (MCI)



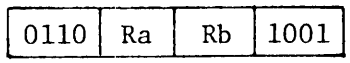
Ra is incremented by 1 and then the byte, K, replaces the C(Ra).

Exchange Characters (XCC)



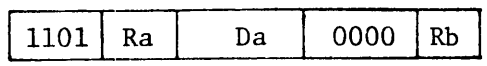
The single byte in C(Ra) is exchanged with the single byte in C(Rb).

Move Character (MCC)



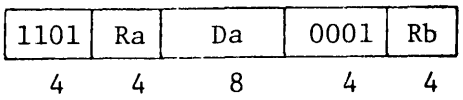
The single byte in C(Rb) replaces the byte in C(Ra). The C(Rb) are not changed.

Move Character to Relative Character (MCC)



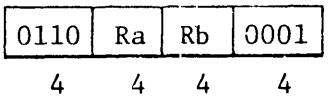
The single byte at the C(Rb) replaces the byte at the C(Ra,Da). The C(Rb) are not changed.

Move Relative Character to Character (MCC)



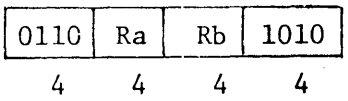
The single byte at the C(Ra,Da) replaces the byte at the C(Rb). The C(Ra,Da) are not changed.

Increment Source Register and Move Character (MIC)



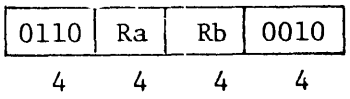
Ra is incremented by 1 and then the single byte in C(Ra) replaces the C(Rb). The C(Ra) are not changed.

Increment Destination Register and Move Character (MCI)



Ra is incremented by 1 and then the single byte at C(Rb) replaces the C(Ra). The C(Rb) are not changed.

Increment Both Registers and Move Character (MII)

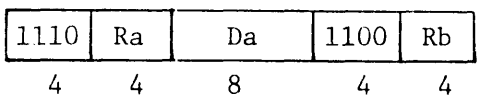


Ra and Rb are each incremented by 1 and then the single byte at C(Ra) replaces the C(Rb). The C(Ra) are not changed.

2.5.3 ADDRESS MODIFICATION OPERATIONS

The following group of instructions are used to modify the displacement portion of an addressing register.

Load Absolute Address Difference (LAD)

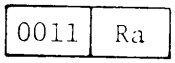


This operation treats the triple word in the C(Ra,Da) as a storage address. The absolute value of the difference between this address and

the address in Rb is computed. The result is a two byte integer. The result replaces the contents of the low-order accumulator.

NOTE: This instruction is valid for unlinked frames only if the frame number in C(Ra,Da) is the same as the frame number in Rb. The instruction is valid for unequal frame numbers only if both frames are in the same group of contiguously linked frames and the difference between the frame numbers is less than 32.

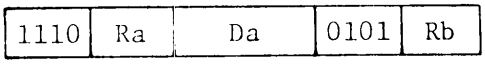
Increment Address Register (INC)



Attachment is forced on Ra and the word address portion of Ra is incremented by one. If the resulting word address is not in the same buffer, then either:

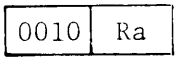
- A crossing frame limits error occurs if Ra is in unlinked format, or
- An attempt is made to attach Ra to the first data byte of the frame pointed to by the forward link of the current frame. In this case, forward link zero and illegal frame id are errors which can be detected if they occur.

Add to Address Register (INC)



The two byte integer at the C(Ra,Da) is added to the displacement portion of Rb. The C(Ra,Da) are not changed. The Da field is doubled to form the effective address.

Decrement Address Register (DEC)



Attachment is forced on Ra and the word address portion of Ra is decremented by one. If Ra is in the unlinked format no error detection is performed. In the linked format, if the resulting word address is XY0B*:

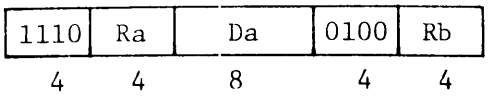
- If the backward link of the current frame is zero, Ra remains attached to data byte zero of the current frame.
- Otherwise, an attempt is made to attach Ra to the last data byte of the frame pointed to by the backward link of the current frame. Illegal frame id is an error which can be detected in this case.

In the linked format, if the resulting word address is XY0A*, a backward link zero error is detected.

---

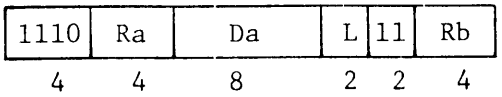
* XY represents any even hex number ($0 \leq XY \leq FE$).

Subtract from Address Register (DEC)



The 16 bit integer at the C(Ra,Da) is subtracted from the address portion of Rb. The C(Ra,Da) are not changed. The Da field is doubled to form the effective address.

Load Effective Address (SRA)

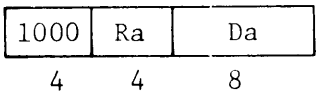


An effective address is computed using the contents of Ra and the Da field. The Da field is doubled if the L field is not zero. The resulting effective address replaces Rb. Ra is not changed.

2.5.4 BIT MANIPULATING OPERATIONS

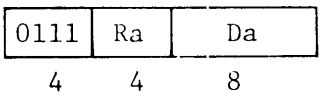
The following two instructions are used to manipulate individual bits.

Set Bit On (SB)



Da is a bit displacement relative to the byte address in Ra. The most significant bit of the byte has a bit number of zero. The least significant bit of the addressed byte has a displacement of seven. The bit in the C(Ra,Da) is set to 1.

Set Bit Off (ZB)



Da is a bit displacement relative to the byte address in Ra. The most significant bit of the byte has a bit number of zero. The least significant bit of the addressed byte has a displacement of seven. The bit in the C(Ra,Da) is set to 0.

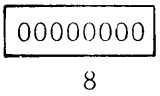
2.5.5 CONTROL OPERATIONS

Instructions that govern the flow of a program, and in particular cause an alteration of the process of taking instructions from sequential locations, are called control instructions.

Branch instructions specify the frame and word displacement relative to the start of the frame from which the computer is to take the next instruction.

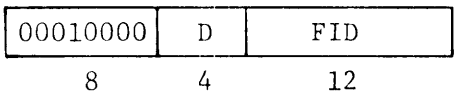
Skip instructions specify the number of bytes to be skipped in order to reach the next instruction. The skip amount is relative to the first byte of the instruction following the skip instruction. The skip amount is a 10 bit field represented in sign magnitude form. For skip instructions, a skip out of the current frame causes a fault trap to location 9 of frame 16.

No Operation (NOP)



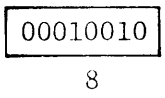
This instruction causes the computer to take the next instruction in sequence.

External Branch (ENT)



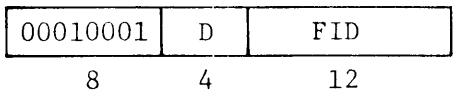
This instruction causes the computer to take its next instruction from the location specified by the FID and D fields. The D field is doubled to determine the branch location relative to byte 1 of the frame. Only the first 16 words of a frame can be specified as branch locations. The first 16 words of a procedure frame normally contain entry vectors.

External Branch Indirect (ENTI)



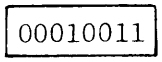
This instruction is similar to the Branch instruction except that the D and FID fields are contained in the two low order bytes of the accumulator.

Branch and Stack Location (BSL)



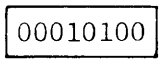
The location following this instruction is stacked in the return stack. The next instruction is taken from the location specified by the FID and D fields. The D field is doubled to determine the location relative to byte 1 of the frame. The location is stored as a two byte FID and a two byte displacement.

Branch and Stack Location Indirect (BSLI)



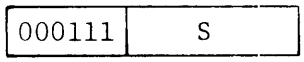
This instruction is similar to Branch and Stack Location except that the D and FID fields are contained in the two low order bytes of the accumulator. The location is stored as a two byte FID and a two byte displacement.

Return (RTN)



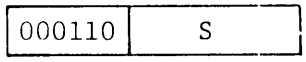
The address stored in the top of the return stack replaces the instruction counter and the return stack is popped. This causes the next instruction executed to be the one following the most recently executed Branch and Stack instruction.

Branch (B)



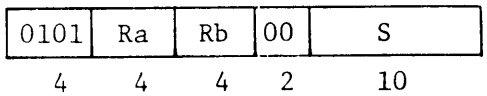
The number of bytes specified by S are skipped.

Branch and Stack Location (BSL)



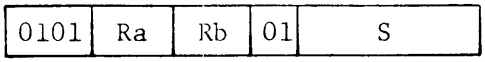
The location following this instruction is stacked in the return stack. Then the number of bytes specified by S are skipped. The location is stored as a two byte zero field and a two byte displacement.

Branch Character Not Equal (BCU)



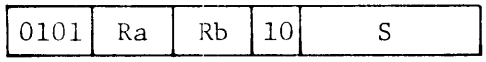
If the byte at the C(Ra) is not equal to the byte at the C(Rb), the number of bytes specified by S are skipped.

Branch Character Less Than (BCL)



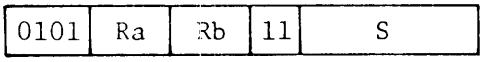
The byte at C(Rb) is subtracted, using 2's complement arithmetic, from the byte at C(Ra) in an internal 8-bit register. If the high order bit of the result is set, the number of bytes specified by S are skipped.

Branch Character Equal (BCE)



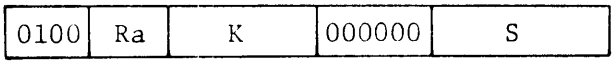
If the byte at the C(Ra) is equal to the byte at the C(Rb), the number of bytes specified by S are skipped.

Branch Character Less Than or Equal (BCLE)



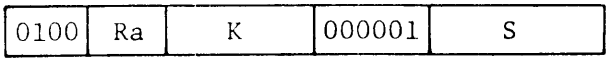
The byte at C(Rb) is subtracted, using 2's complement arithmetic, from the byte at C(Ra) in an internal 8-bit register. If the result is zero or if the high order bit is set, the number of bytes specified by S are skipped.

Branch Character Not Equal to Immediate (BCU)



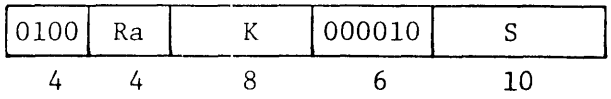
If the byte, K, is not equal to the byte at the C(Ra), the number of bytes specified by S are skipped.

Branch Character Less Than Immediate (BCL)



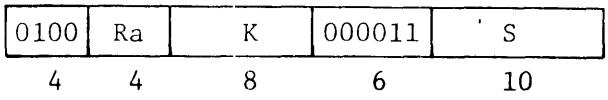
The byte at C(Ra) is subtracted, using 2's complement arithmetic, from the byte K in an internal 8-bit register. If the high order bit of the result is set, the number of bytes specified by S are skipped.

Branch Character Equal to Immediate (BCE)



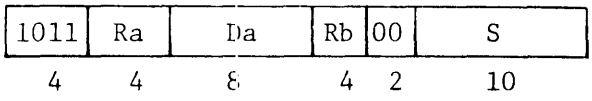
If the byte, K, is equal to the byte at the C(Ra), the number of bytes specified by S are skipped.

Branch Character Less Than or Equal to Immediate (BCLE)



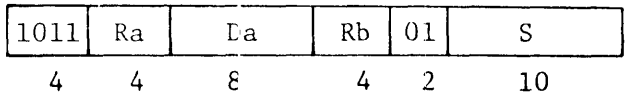
The byte at C(Ra) is subtracted, using 2's complement arithmetic, from the byte K in an internal 8-bit register. If the result is zero or if the high order bit is set, the number of bytes specified by S are skipped.

Branch Relative Character Not Equal (BCU)



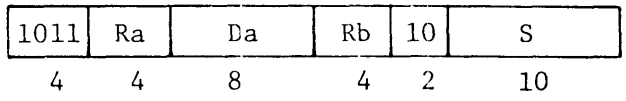
If the byte at the C(Ra,Da) is not equal to the byte at the C(Rb), the number of bytes specified by S are skipped.

Branch Relative Character Less Than (BCL)



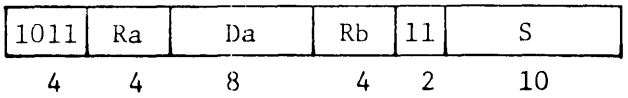
The byte at C(Rb) is subtracted, using 2's complement arithmetic, from the byte at C(Ra, Da) in an internal 8-bit register. If the high order bit of the result is set, the number of bytes specified by S are skipped.

Branch Relative Character Equal (BCE)



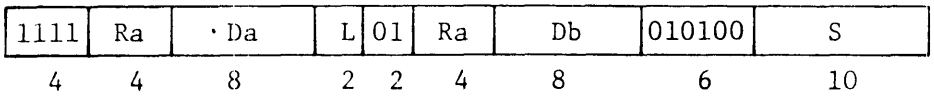
If the byte at the C(Ra,Da) is equal to the byte at the C(Rb), the number of bytes specified by S are skipped.

Branch Relative Character Less Than or Equal (BCLE)



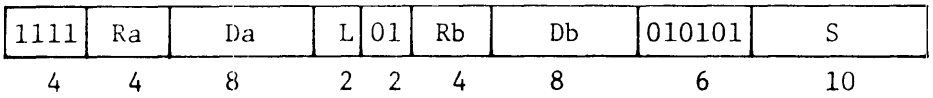
The byte at C(Rb) is subtracted, using 2's complement arithmetic, from the byte at C(Ra, Da) in an internal 8-bit register. If the result is zero or if the high order bit is set, the number of bytes specified by S are skipped.

Compare and Branch Not Equal (BU)



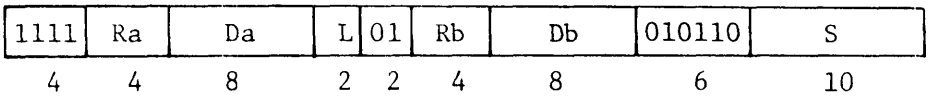
If the C(Ra,Da) are not equal to the C(Rb,Db), the number of bytes specified by S are skipped.

Compare and Branch Less Than (BL)



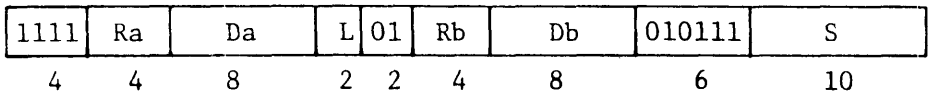
The value at C(Rb, Db) is subtracted, using 2's complement arithmetic, from the byte at C(Ra, Da) in an internal register. If the high order bit of the result is set, the number of bytes specified by S are skipped. The OVFBIT will be set if overflow occurs on the subtraction; otherwise, OVFBIT will be reset. If OVFBIT is set, the condition of the branch should be reversed.

Compare and Branch Equal (BE)



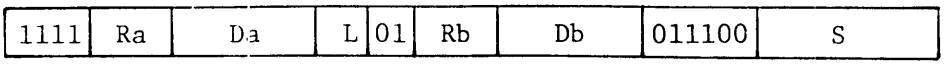
If the C(Ra,Da) are equal to the C(Rb,Db), the number of bytes specified by S are skipped.

Compare and Branch Less Than or Equal (BLE)



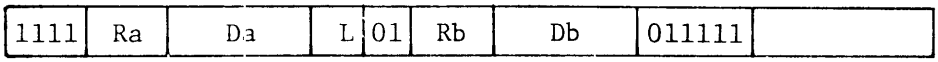
The value at C(Rb, Db) is subtracted, using 2's complement arithmetic, from the byte at C(Ra, Da) in an internal register. If the result is zero or if the high order bit is set, the number of bytes specified by S are skipped. The OVFBIT will be set if overflow occurs on the subtraction; otherwise, OVFBIT will be reset. If OVFBIT is set, the condition of the branch should be reversed.

Subtract and Branch Not Equal (BDNZ)



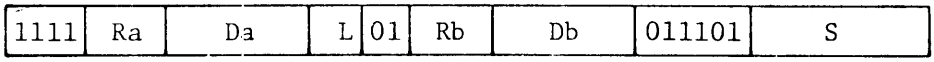
The C(Rb,Db) are subtracted from the C(Ra,Da). The difference replaces the C(Ra,Da). If the original C(Ra,Da) are not equal to the C(Rb,Db), the number of bytes specified by S are skipped. The condition codes are set.

Subtract and Branch Less Than or Equal (BDLEZ)



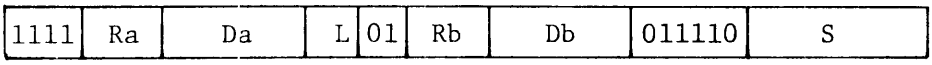
The C(Rb,Db) are subtracted from the C(Ra,Da). The difference replaces the C(Ra,Da). If the original C(Ra,Da) are less than or equal to the C(Ra,Da) the number of bytes specified by S are skipped. The condition codes are set.

Subtract and Branch Less Than (BDLZ)



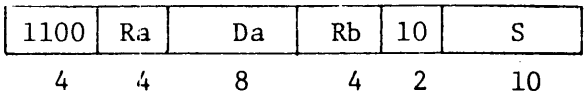
The C(Rb,Db) are subtracted from the C(Ra,Da). The difference replaces the C(Ra,Da). If the original C(Ra,Da) are less than the C(Rb,Db), the number of bytes specified by S are skipped. The condition codes are set.

Subtract and Branch Equal (BDZ)



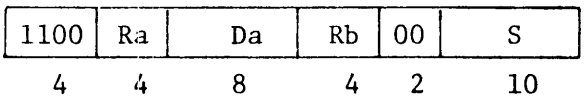
The C(Rb,Db) are subtracted from the C(Ra,Da). The difference replaces the C(Ra,Da). If the original C(Ra,Da) are equal to the C(Rb,Db), the number of bytes specified by S are skipped. The condition codes are set.

Branch Address Equal (BE)



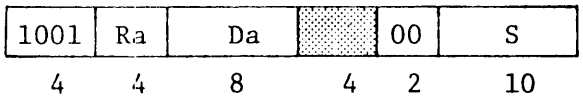
The 6 byte (C(Ra,Da) are compared with the address of Rb. If the values are equal, the number of bytes specified by S are skipped. It is possible for two addresses to compare not equal even though they represent the same storage location. This can occur if the FID in the C(Ra,Da) is not the same as the FID in Rb. See Note under LAD instruction.

Branch Address Not Equal (BU)



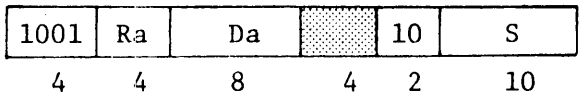
The 6 byte C(Ra,Da) are compared with the address of Rb. If the values are not equal, the number of bytes specified by S are skipped. It is possible for two addresses to compare not equal even though they represent the same storage location. This can occur if the FID in the C(Ra,Da) is not the same as the FID in Rb. See Note under LAD instruction.

Branch Bit Set (BBS)



Da is a bit displacement relative to the byte address in Ra. The most significant bit of the addressed byte has a displacement of zero. The least significant bit of the addressed byte has a displacement of seven. If the bit in the C(Ra,Da) is on (bit = 1), the number of bytes specified by S are skipped.

Branch Bit Zero (BBZ)



Da is a bit displacement relative to the byte address in Ra. The most significant bit of the addressed byte has a displacement of zero. The least significant bit of the addressed byte has a displacement of seven. If the bit in the C(Ra,Da) is off (bit = 0), the number of bytes specified by S are skipped.

2.5.6 LOGICAL OPERATIONS

The following group of instructions perform logical operations between a byte in storage and an immediate operand. The logical operations are AND, OR (sometimes called "inclusive or") and XOR ("exclusive or").

When two bytes are combined by an AND, they are matched bit for bit. If the same bit position in each byte contains a 1, the result is a 1. If a position of either byte (or both bytes) contains a 0, the result is 0.

The following is an example of a logical AND operation:

```
10110001
00110110
00110000   resulting AND
```

When two bytes are combined by an OR they are matched bit for bit. If the same bit position in each byte contains a 0, the result is a 0. If a position of either byte (or both bytes) contain a 1, the result is a 1. The following is an example of logical OR:

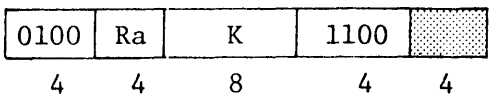
```
11011000
00010001
11011001   resulting OR
```

When two bytes are combined by an XOR they are matched bit for bit. If the corresponding bit positions in each byte are the same (both 0 or both 1) the result is 0. If the same bit position in each byte is not the same (either contains a 1 while the other contains a 0) the result is 1.

The following is an example of a logical XOR operation.

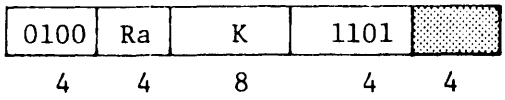
```
10011100
00011011
10000111   resulting XOR
```

AND Character (AND)



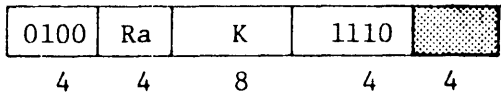
The byte in the C(Ra) and K are logically AND'ed. The result replaces the C(Ra).

OR Character (OR)



The byte in the C(Ra) and K are logically OR'ed. The result replaces the C(Ra).

Exclusive OR Character (XOR)

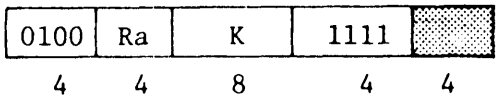


The byte in the C(Ra) and K are logically exclusive OR'ed. The result replaces the C(Ra).

2.5.7 SHIFT OPERATION

The following instruction allows a byte to be shifted one bit to the right.

Shift Character Right (SHIFT)

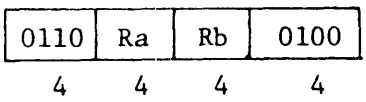


K is shifted right one bit position. A zero bit is inserted on the left and the bit shifted off the right is lost. The result replaces the C(Ra). The value K in the instruction does not change.

2.5.8 STRING OPERATIONS

The following instructions operate on strings. A string is a logically contiguous group of bytes. Strings may extend across frame boundaries provided that the frames are linked.

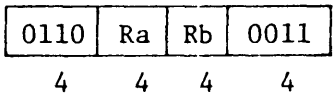
Increment and Move String Under Count Control (MIIT)



The contents of the lower half of the accumulator (T0) is read into internal hardware registers. Ra and Rb are each incremented by one and then the byte at c(Ra) replaces the byte at c(Rb). Next the internal hardware registers are decremented. If the resulting value is not zero the increment and move is repeated. If, during the

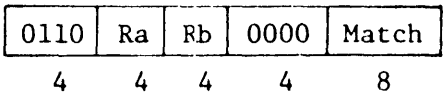
execution of the instruction, an internal interrupt occurs or if the move crosses the frame boundary of a linked frame, the current contents of the internal hardware registers are stored in TO. For this reason the contents of TO at the conclusion of the move are indeterminable. If TO is initially zero no operation is performed.

Increment and Move String Under Address Control (MIIR)



Ra and Rb are each incremented by one and then the byte at C(Ra) replaces the byte at C(Rb). Next Ra is compared with the contents of address register 15. If the values are not equal the operation is repeated. If Ra is initially equal to the contents of address register no operation is performed.

Increment and Move String Under Match Control (MIID)



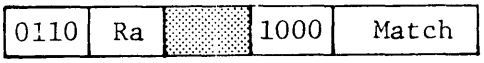
Ra and Rb are each incremented by one and then the byte at C(Ra) replaces the byte at C(Rb). The byte that moved is then tested for a match with one of 7 possible values as defined by the match field. If the match is not successful, the operation is repeated.

The matching is performed as follows. For each of the bit positions one through seven that is a 1, a match test is performed. If bit position zero is a 1, the move stops on any equal match. If bit position zero is a 0, the move stops if none of the bytes tested match. The table below shows the test performed for each bit in the mask.

Bit in Match Field	Test Performed
0	1 = Stop on equal 0 = Stop if unequal
1	Compare with Hexadecimal "FF"
2	Compare with Hexadecimal "FE"
3	Compare with Hexadecimal "FD"
4	Compare with Hexadecimal "FC"
5	Compare with Byte at 003 in PCB
6	Compare with Byte at 004 in PCB
7	Compare with Byte at 005 in PCB

Note: byte 003 of the PCB may not contain a hexadecimal 00 or 01.

Increment and Scan String Under Match Control (SCD)

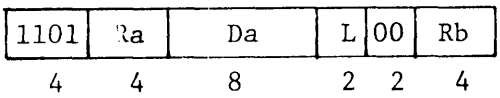


Ra is incremented by one and then the byte at C(Ra) is tested for a match as defined by the match field. If the match is not successful, the operation is repeated. See the description of the Increment and Move String Under Match Control instruction for the matching rules.

2.5.9 CONVERSION OPERATIONS

Conversion operations are provided to convert decimal integers represented by ASCII characters into binary values, and to convert hexadecimal integers into binary values, and binary values to hexadecimal.

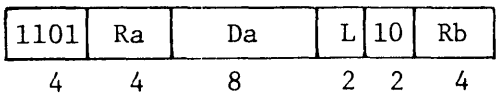
Decimal to Binary (MDB)



The C(Ra,Da) are multiplied by ten. The binary value of the ASCII digit in the C(Rb) is added to the product, and the result replaces the C(Ra,Da). This instruction is not defined for a single byte at C(Ra,Da). (A value of L = 0 represents a different operation.)

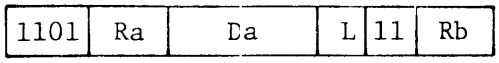
If the C(Ra,Da) are initially zero, repeated use of this instruction (with incrementing of Rb) will convert an ASCII string representing a decimal value into a binary integer.

Hexadecimal to Binary (MXB)



The C(Ra,Da) are multiplied by sixteen. The binary value of the ASCII hexadecimal digit in the C(Rb) is added to the product and the result replaces the C(Ra,Da). If the C(Ra,Da) are initially set equal to zero, repeated use of this instruction will convert an ASCII string representing a hexadecimal value into a binary integer.

Binary to Hexadecimal (MBX)

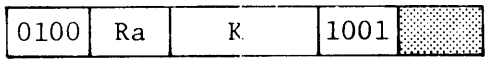


The binary integer at the C(Ra,Da) is converted into an ASCII string starting at the C(Rb) +1. Bits 28 through 31 of the accumulator contain a count of the maximum number of ASCII bytes to be generated. If bit 24 of the accumulator is a zero, the leading zeros of the hexadecimal string are suppressed and the C(Rb) +1 will contain the most significant non-zero hexadecimal digit. If bit 24 of the accumulator is a 1, zero suppression will not take place. The contents of the accumulator is unpredictable after this instruction is executed.

2.5.10 INPUT OUTPUT OPERATIONS

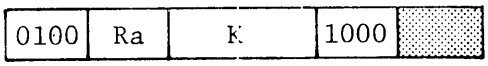
The input output operations provide for communication with the terminal associated with a process and for input and output with peripheral devices.

Input a Byte (IB)



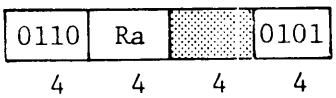
The K field specifies a 3 bit function code and a 5 bit device address. The byte from the selected device replaces the C(Ra). This instruction can be executed only in monitor mode.

Output a Byte (OB)



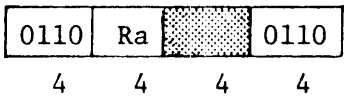
The K field specifies a 3 bit function code and a 5 bit device address. The single byte in the C(Ra) is transmitted to the selected device. This instruction can be executed only in monitor mode.

Read Input Queue (READ)



The next character from the terminal input queue replaces the C(Ra). If the input queue is empty the process is suspended until a character is received from the terminal. Characters transmitted by the terminal are automatically queued in the PIB for the terminal.

Write to Output Queue (WRITE)

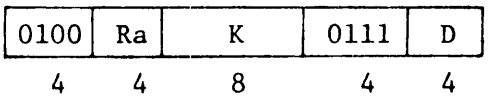


The byte in the C(Ra) is placed into the terminal output queue. If the queue is full, the process is suspended until the terminal has printed all but four characters from the queue. If there are any characters in the input queue before this instruction is executed, they are lost.

2.5.11 MONITOR OPERATIONS

The following operations are used to communicate with the monitor.

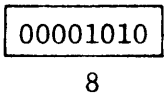
Monitor Call (MCAL)



This operation generates an interrupt into the monitor. The four bits of the D field are doubled to determine the location relative to byte 1 of the monitor frame for transfer of control.

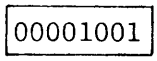
The address contained in Ra, the address of the PCB of the current process, and the K field from this instruction are all placed into the PIB for the current process, and then control is passed to the monitor.

Resume Virtual Process (RVP)



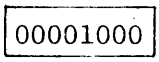
This operation returns control to a process that has previously been interrupted. The status of the interrupted process is restored and execution of the process resumes from the point of the interrupt. This instruction can be executed only in monitor mode.

Start Virtual Process (SVP)



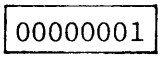
The FID portion of register 4 (of the monitor) is treated as the location of the PCB for a dependent process. The buffer map is searched for the PCB frame. If the PCB is present, registers 0 and 1 of the PCB are attached and execution of the process begins. If the frame is not present, the referenced FID is placed into the PIB and the monitor is reentered at the absent frame entry point (location 1). This instruction can be executed only in monitor mode.

HALT



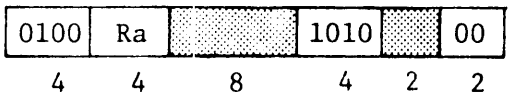
The CPU is halted; this instruction can be executed only in monitor mode.

Test Interrupts



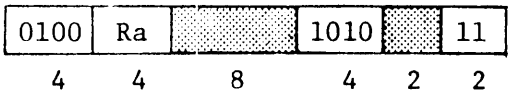
This instruction has meaning only in the monitor mode; it is a NOP in the virtual mode. Internal and External Interrupts are tested for, and, if any are pending, a fault trap to the appropriate monitor location is taken.

Halt and Display (HLD)



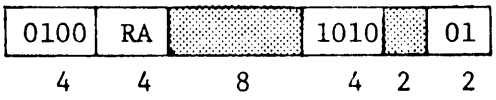
Halts the CPU and gates the eight-bit literal addressed by register RA to the A bus where it can be displayed in the eight least significant indicator lamps of the system panel by depressing the Data select switch. This instruction is restricted to Monitor level code.

Enter Console Command Switches (ECS)



The status of the eight low-order console command switches is placed in the eight bit byte addressed by register RA. If the switch is on, the corresponding bit in the byte addressed by register RA is set to

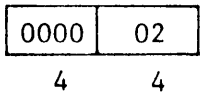
one. This instruction is restricted to Monitor level code. If a switch is not set, the corresponding bit will be set to zero.



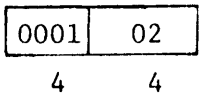
The status of the four console sense switches is placed in the four most significant bits of the eight bit byte addressed by register RA. The status of a switch is one when the switch is set. The four low order bits are set to one. This instruction is restricted to Monitor level code.

2.6 INSTRUCTION SUMMARY

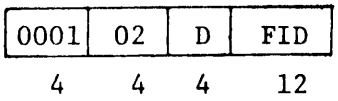
The following diagrams show the formats for each of the instructions. The diagrams are listed in order of increasing primary operation code (first four bits). The operation code is shown as a binary value. The second and third portions of the operation code (if they appear) are labeled 02 and 03 respectively.



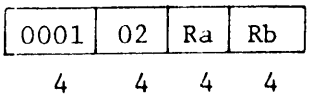
02	INSTRUCTION
0000	No Operation
0001	Test Interrupts
1000	Halt the CPU
1010	Resume Virtual Process
1011	Start Virtual Process
1100	Branch to Absolute Address
1101	Branch and Stack Location, to Absolute Address



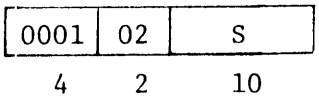
02	INSTRUCTION
0010	Branch Indirect (External)
0011	Branch and Stack Location Indirect (External)
0100	Return
0101	Return without Trace



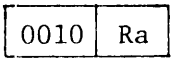
02	INSTRUCTION
0000	Branch (External)
0001	Branch and Stack Location (External)



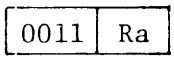
02	INSTRUCTION
0110	Move Address Register to Address Register
0111	Exchange Address Registers



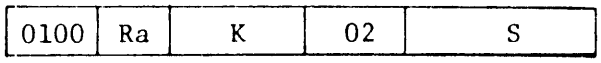
02	INSTRUCTION
10	Branch and Stack Location (Internal)
11	Branch (Internal)



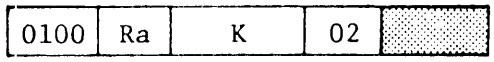
INSTRUCTION
Decrement Address Register



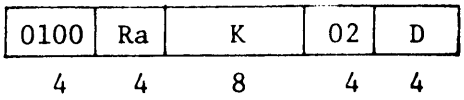
INSTRUCTION
Increment Address Register



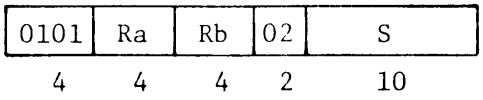
02	INSTRUCTION
000000	Branch Character Not Equal to Immediate
000001	Branch Character Less Than Immediate
000010	Branch Character Equal to Immediate
000011	Branch Character Less Than or Equal to Immediate



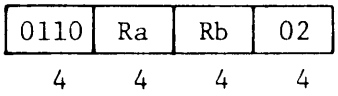
02	INSTRUCTION
0010	Move Immediate Character
0100	Increment and Store Immediate Character
0110	Flag the Address Register
0000	Output Byte
1001	Input Byte
1100	Or
1101	Exclusive Or
1110	And
1111	Shift



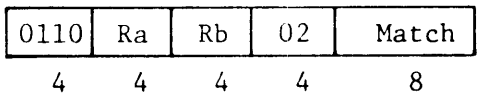
02	INSTRUCTION
0111	Monitor Call



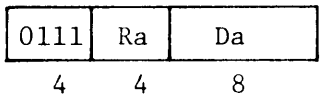
02	INSTRUCTION
00	Branch Character Not Equal
01	Branch Character Less Than
10	Branch Character Equal
11	Branch Character Less Than or Equal



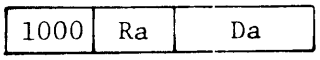
02	INSTRUCTION
0001	Increment Source Register and Move Character
0010	Increment Both Registers and Move Character
0011	Increment and Move String Under Address Control
0100	Increment and Move String Under Count Control
0101	Read Terminal Queue
0111	Exchange Bytes
1001	Move Byte
1010	Increment Destination Register and Move Character
1101	Write Terminal Queue



02	INSTRUCTION
0000	Increment and Move String Under Match Control
1000	Increment and Scan String Under Match Control

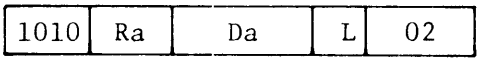


INSTRUCTION
Set Bit Off

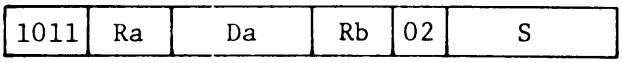


INSTRUCTION
Set Bit On

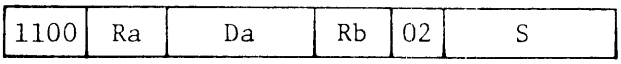
02	INSTRUCTION
00	Branch Bit On
01	Branch Bit Off



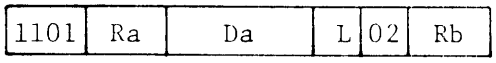
02	INSTRUCTION
000000	Store a Zero
000001	Store a One
000011	Add One to Storage
000101	Subtract One from Storage
001000	Negate Storage
010000	Multiply
010001	Divide
010011	Add to Accumulator
010101	Subtract from Accumulator
011000	Load Accumulator
011001	Store Accumulator



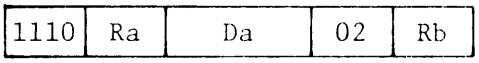
02	INSTRUCTION
00	Branch Relative Character Not Equal
01	Branch Relative Character Less Than
10	Branch Relative Character Equal
11	Branch Relative Character Less Than or Equal



02	INSTRUCTION
00	Branch Addresses Equal
10	Branch Addresses Not Equal

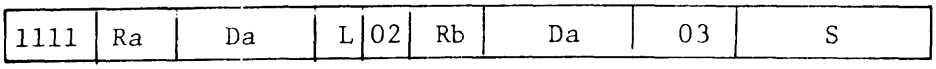


02	INSTRUCTION
00	Move Byte to Relative Byte (L equal to zero)
00	Decimal to Binary (L not equal to zero)
01	Move Offset Byte to Byte
10	Hexadecimal to Binary
11	Binary to Hexadecimal



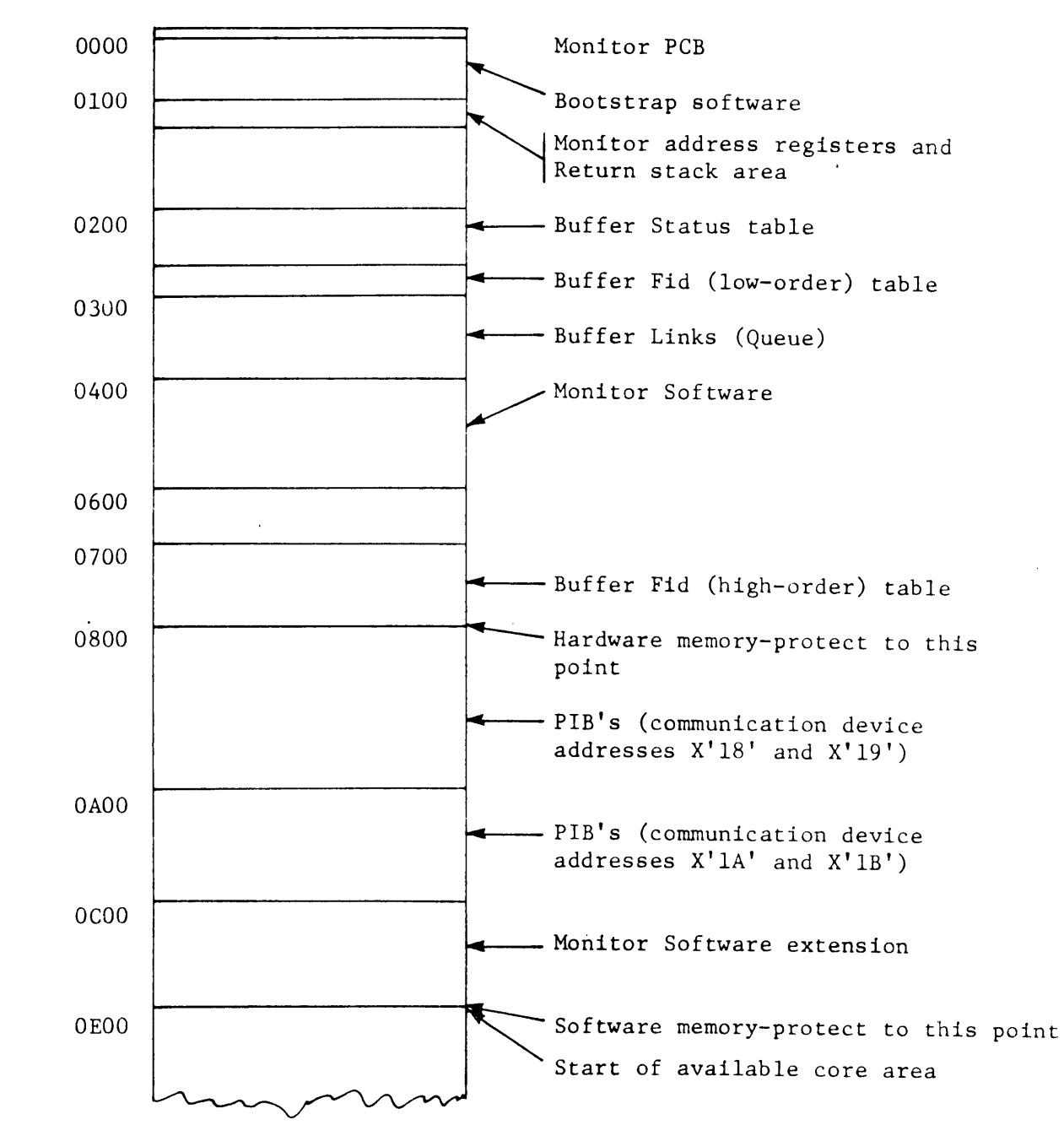
02	INSTRUCTION
0011	Load Effective Address (half word)
0100	Subtract from Address Register
0101	Add to Address Register
0111	Load Effective Address (full word)
1011	Load Effective Address (double word)
1100	Load Absolute Address Difference
1101	Store Address Register
1110	Load Address Register
1111	Load Effective Address (triple word)

02	INSTRUCTION
00	Move Storage to Storage
10	Add Storage to Storage
11	Subtract Storage from Storage



02	03	INSTRUCTION
01	010100	Compare and Branch Not Equal
01	010101	Compare and Branch Less Than
01	010110	Compare and Branch Equal
01	010111	Compare and Branch Less Than or Equal
01	011100	Subtract and Branch Not Equal
01	011111	Subtract and Branch Less Than Equal or Equal
01	011101	Subtract and Branch Less Than Equal
01	011110	Subtract and Branch Equal

2.7 CORE MAP


Core Address
0000
0100
0200
0300
0400
0600
0700
0800
0A00
0C00
0E00
Monitor PCB
Bootstrap software
Monitor address registers and Return stack area
Buffer Status table
Buffer Fid (low-order) table
Buffer Links (Queue)
Monitor Software
Buffer Fid (high-order) table
Hardware memory-protect to this point
PIB's (communication device addresses X'18' and X'19')
PIB's (communication device addresses X'1A' and X'1B')
Monitor Software extension
Software memory-protect to this point
Start of available core area

2.8 PERIPHERAL I/O: DEVICE ORDERS

Order Number	Operation	Description
0	Data Transfer	A data byte will be transferred between the addressed device and the processor. Direction of the transfer will depend on whether the instruction is an input or an output.
1	Status/Function	A status byte will be input from the addressed device or a function byte will be output to the addressed device, depending on whether the instruction is an input or an output.
2	Block Input/INT	The addressed device will start a concurrent block input to memory and will generate an external interrupt at the conclusion of the transfer unless the interrupt has been subsequently disarmed. This order should be sent by an output instruction.
3	Arm Interrupt	Permits the addressed device to make an external interrupt request upon the satisfaction of an interrupt condition. This order should be sent by an output instruction.
4	Disconnect	The block transfer in progress by the addressed device is stop ed and end of block interrupt will occur unless the interrupt has been disarmed. This order should be sent by an output instruction.
5	Disarm Interrupt	Inhibits the addressed device from marking an external interrupt request under any condition. This order should be sent by an output instruction.
6	Block Output/INT	The addressed device will start a concurrent block output from memory and will generate an external interrupt at the conclusion of the transfer unless the interrupt has been subsequently disarmed. This order should be sent by an output instruction.
7	Unassigned	This order, if assigned, may perform any required function as interpreted by the individual interface. If a byte transfer is desired the order may be sent by an input or an output instruction.

SECTION 3

REALITY ASSEMBLY LANGUAGE (REAL)

The Reality Assembler (REAL) translates source statements into Reality CPU machine language equivalents. The source file, or "mode" is an item in any file defined on the database. The mode is assembled in place; that is, at the conclusion of the assembly process, the item contains both the original source statements, as well as the generated object code. The same mode can then be used to generate a formatted listing (using the MLIST verb) or can be loaded for execution (using the MLOAD verb).

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 op-code) 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 op-code. Op-codes are pre-defined in the permanent op-code symbol file OSYM and consist of one or more alpha characters. Op-codes may be mnemonics for Reality machine language instruction (eg., B for BRANCH) macros, which may assemble into several Reality machine language instructions (eg., MBD for MOVE BINARY to DECIMAL), or assembler pseudo-ops (eg., ORG for ORIGIN). Additionally, users may define new mnemonics or "macros" which expand into several Reality machine instructions. This may be done by creating new entries in the OSYM file.

3.1.3 OPERAND FIELD

Operand field entries are optional, and vary in number according to the needs of the associated REAL op-code. Entries are separated by commas and cannot contain embedded blanks (except for character string literals enclosed by single quotes). The operand field is terminated by the first blank encountered. The characters + - ' * have special meaning in this field.

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 is an entry in the pre-defined permanent symbol file (PSYM). A constant may be one of the following forms:

*	-	Defines current value of the assembler location counter.
n	-	(n decimal) - A decimal constant.
X'h'	-	(h hexadecimal) - A hexadecimal constant.
C'text'	-	Character string; any characters, including blanks and commas, may appear as part of "text"; a sequence of two single quotes ('') is used to represent one single quote in the text.

Arithmetic operators (+,-) may be used to combine two or more constants.

3.1.5 COMMENT FIELD

Any commentary information preceded by a blank may follow the operand field entries.

3.1.6 "ARGUMENT" FIELD

For the purposes of the remainder of this documentation, the label field entry, op-code field entry, and operand field entries will be referred to as "argument field" (AF) 0, 1, 2, etc.

3.2 CALLING THE ASSEMBLER

The assembler is called by the statement:

```
AS file-name item-name {(Q)}
```

which will assemble the item in the file specified. The optional specification "(Q)" specifies that error lines are not to be listed at the end of the assembly.

As the assembler processes, it will output an asterisk (*) as every ten source statements are assembled. At the end of pass-1 a new line is started and an asterisk is printed for each ten statements reassembled.

3.3 LISTING OUTPUT

The listing processor may be called by the statement:

```
MLIST file-name item-name {(options)}
```

Options are separated by commas:

P Routes output to the line-printer.

M Prints macro-expansions of source statements.

E Prints error lines only; also suppresses the pagination, and enters EDIT at the end of the listing.

Z Inhibits EDIT entry when E option is specified.

n-m Restricts listing to line numbers n through m inclusive.

The listing is output with a statement number, location counter, object code and source code, with the label, op-code, operand and comment fields aligned. A page heading is output at the top of each new page.

Errors, if any, appear in the location counter/object code area; macro expansions appear as source code if not suppressed, with the operation codes prefixed by a plus sign (+).

A sample listing output is shown on the next page.

```
:MLIST SYSGEN-FILE ABSL (1-25,M) (CR)

MODE: ABSL                                    PAGE 1        17:26  26 OCT 1973

001                                 FRAME 047
    001 7FF0002F                    +FRM: 047
                                    +ORG  1
002                 *SYSTEM MODE
003                                 B     !ABSD             0
    001 1C04                        +B:   !ABSD
004                                 B     !ABSL             1
    003 1C3C                        +B:   !ABSL
005                                 B     !SEGMNT           2
    005 1D80                        +B:   !SEGMNT
006                 DETACH  DEFM    14,TAPEIO
007                 GETWS   DEFM    3,WSPACES
008                 *
009 007 100031      !ABSD   ENT     ABSD
010 00A 0000002F    ABSLFID DTLY    47                      FID OF THIS PROGRAMME
011 00E 00000022    OF1FID  DTLY    34                      FID OF OF1 PROGRAMME
012                 XUSER   DEFT    15,USER                 TALLY USER RELATIVE TO REGISTER 15
013 012 0000        PIBADDR ADDR    0,X'80FFFFFC'
    014 80FFFFC
014                 *
015                 * ERROR ENTRY POINT *
016 018 E062E4      TPERR   MOV     JSBAG,IS
017 01B 111006              BSL     CRLFPRINT
018 01E 07                  TEXT    X'07',C'TAPE FORMAT ERROR'.X'FDFF'
    01F 54415045
    023 20464F52
    027 4D415420
    02B 4552524F
    02F 52
    030 FDFF
019                         MII     IS,OB,OBSIZE        COPY TAPE DATA & PRINT
    032 A05B53              +LOADT  OBSIZE
    035 64B4                +MIITRP IS,OB
020 037 112006              BSL     WRTLIN
021 03A 08                  TEXT    X'08'               BLOW-UP HERE
022                         DEC     RSCWA,4             BACKUP STACK
    03B F0C171E7            +DEC    RSCWA,=-4
023                         B       NXTAB               CONTINUE TO NEXT TAPE SEGMENT
    03F 1CAC                +B:     NXTAB
024                 *
025                 ********************************************
EC
```

3.4 LOADING

The assembled mode may be loaded into the frame specified by the FRAME op-code by using the statement:

```
MLOAD file-name item-name
```

If the load is successful, the message:

```
[216] 'item-name' LOADED ON FRAME # n
size = m (DEC), h (HEX)
```

is returned.

The mode will not load correctly if its size exceeds 512 bytes, or if a FRAME statement is not the first statement assembled in the mode. In either case, a message will be returned indicating the error.

Note: MLDAD can not load itself. It must be loaded by the XLOAD verb by the following message:

```
XLOAD SYSTEM-MODES LOADER
```

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:

```
: MVERIFY SM EXAMPL1 (CR)

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

: MVERIFY SM EXAMPL2 (CR)

014 OC 18

[218] MODE 'EXAMPL2' HAS 1 BYTES OBJECT CODE 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 file which mismatches will be listed, followed by the value in the executable frame.

Example:

```
:MVERIFY SM EXAMPL3 (A) (CR)

LOC  XX  YY   LOC  XX  YY   LOC  XX  YY   LOC  XX  YY
014  OC  18   015  13  17   016  OE  OD   017  3A  3C
--   --            --            --            --

[218] MODE 'EXAMPL3' HAS 78 BYTES OBJECT CODE MIS-MATCHES
```

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 by item with the external references of that item. The CROSS-INDEX Verb requires the following format:

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

Example:

```
: CROSS-INDEX MODES * (CR)
```

Would cross index all items of the modes file. An example of what a portion of the CSYM file might look like after using the CROSS-INDEX Verb follows. Notice that the item called DLOAD has one external reference to LISTFLAG, two external references to RMBIT, etc.

```
    DLOAD
ØØ1 LISTFLAG Ø1 RMBIT Ø2
ØØ2 CH8 Ø1
ØØ3 NNCF Ø2
ØØ4 CTR1 Ø2 CTR2 Ø1 MODULO Ø7 OBSIZE Ø1 RSCWA Ø1 SEPAR 1Ø TØ Ø1 T4 Ø3
ØØ5 BASE Ø8 DØ Ø1 OVRFLW Ø1 R15FID Ø1 RECORD Ø5
ØØ6 BMSBEG Ø1 CSBEG Ø1 ISBEG Ø2 OBBEG Ø1 S2 Ø2
ØØ7 CS Ø6 IS 21 OB Ø5 R14 Ø3 R15 Ø6 TS Ø1
ØØ8 ABSL Ø2 CRLFPRINT Ø1 CVDR15 Ø3 CVTNIS Ø2 GETBLK Ø1 LINK Ø1 MBDNSUB Ø3
    UPDITM Ø1 WRTLIN Ø2
ØØ9 AM Ø2
Ø1Ø

    WRAPUP-III
ØØ1 INHIBIT Ø4 RMBIT Ø4 SB6Ø Ø3 SB61 Ø6
ØØ2
ØØ3
ØØ4 CTRØ 13 CTR1 Ø3 EMOD Ø1 MMOD Ø1 MODULO Ø1 OBSIZE Ø2 TØ Ø2 T6 Ø3 USER Ø4
ØØ5 BASE Ø3 EBASE Ø3 MBASE Ø1
ØØ6 BMSBEG Ø3 OBBEG Ø2 OBEND Ø6 SR4 Ø2 SYSR1 Ø2
ØØ7 AF Ø8 BMS Ø4 IR 25 OB 3Ø R14 Ø4 R15 Ø3 TS 14
ØØ8 DATE Ø1 GBMS Ø1 RETIX Ø2 TIME Ø1 WRITOB Ø1 WRTLIN Ø8
ØØ9 AM Ø5 MMOD Ø1 SM Ø4
Ø1Ø
```

```
    DUMP-I
ØØ1 GBIT Ø6
ØØ2
ØØ3 NNCF Ø1 NPCF Ø1
ØØ4 C1 Ø8 C2 Ø2 C6 Ø7 C7 Ø6 REJCTR Ø1 TØ Ø1 T4 Ø1
ØØ5 D3 Ø7 D4 Ø5 FRMP Ø1 LINQUE Ø7 RECORD Ø7
ØØ6 IREND Ø6 S1 Ø3 S3 Ø8
ØØ7 IR 18 IS 15 OB 17 R15 14 IS Ø5
ØØ8 CVTNIS Ø4 DUMP-II Ø1 MBDNSUB Ø1 MBDSUB Ø7 MD99 Ø1 MD999 Ø1 RDREC Ø4
    WRTLIN Ø5
ØØ9 LF Ø2
Ø1Ø

    PROC-I
ØØ1 PQFLG Ø4 SB1 Ø2 SB2 Ø1 SBIT Ø7 STKFLG Ø1
ØØ2 PRMPC Ø1
ØØ3
ØØ4 C1 Ø1 TØ Ø2
ØØ5
ØØ6 BMSBEG Ø1 IBBEG Ø1 IBEND Ø1 ISBEG Ø1 OSBEG Ø1 PBUFBEG Ø4 PQBEG Ø2
    PQCUR Ø2 PQEND Ø3 SR35 Ø1 SR4 Ø1 Ø2
ØØ7 CS Ø2 TB Ø7 IR 81 IS 11 OB Ø5 OS Ø1 R14 12 TS Ø4 UPD 12
ØØ8 CVTHIR Ø1 GETIBX Ø1 MD18 Ø1 MD995 Ø1 MD999 Ø1 PROC-II Ø4 PROC-III Ø4
    WRAPUP-I Ø1 WRITOB Ø1 WRTLIN
ØØ9 AM 12 CR Ø1 SM Ø2 VM Ø1
Ø1Ø
```

3.6.2 XREF VERB

The TCL-II XREF Verb uses the CSYM file as updated by the Cross-Index Verb for input. XREF then updates the XSYM file in the opposite order of the CSYM file. The XREF Verb requires the following format:

```
XREF file-name item-list {(options)}
```

Example:

```
: XREF CSYM * (CR)
```

Would cross reference all items of the CSYM file. An example of what a portion of the XSYM file might look like after using the XREF Verb follows. Notice that the item called T4 was externally referenced by DLOAD, DUMP-I, and SYSTEM-SUBS-II.

```
    MBIT
ØØ1 ASTAT

    GEN
ØØ1 ASTAT

    CVDR15
ØØ1 DLOAD
```

```
    LFDLY
ØØ1 SYSTEM-SUBS--II

    PAGSIZE
ØØ1 SYSTEM-SUBS--II

    T4
ØØ1 DLOAD DUMP-I SYSTEM-SUBS-II

    D5
ØØ1 SYSTEM-SUBS--II

    CTR2
ØØ1 DLOAD AID1

    UPDITM
ØØ1 XREF ILOAD T-LOAD DLOAD EDIT-IV/P&A

    C1
ØØ1 XREF ILOAD ASTAT T-LOAD DUMP-I PROC-I EDIT-IV/P&A EDIT-I

    SR13
ØØ1 EDIT-I

    SR7
ØØ1 EDIT-I

    MBASE
ØØ1 XREF WRAPUP--III TI WII

    DB1
ØØ1 WII

    B15
ØØ1 MONITOR/295Ø,1ØMB

    R7WA
ØØ1 DISK-DIAGNOSTIC/2314

    T5
```

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

Example:

```
:SORT XSYM REFERENCES NONCOL (P)CR
```

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

The following is an example of a partial listing.

```
XSYM : ABIT
REFERENCES   EDIT-I

XSYM : ABSL
REFERENCES   DLOAD

XSYM : ACF
REFERENCES   WII

XSYM : ADDLAB
REFERENCES   ASTAT

XSYM : AF
REFERENCES   ASTAT          WRAP-III          EDIT-I

XSYM : AFBEG
REFERENCES   ASTAT          EDIT-I

XSYM : AFEND
REFERENCES   ASTAT

XSYM : ALIGN
REFERENCES   ASTAT

XSYM : AM
REFERENCES   XREF           ASTAT             T-LOAD
             DLOAD          WRAPUP-III        PROC-I
             AID1           TI                WII
```

3.6.3 XREF PROC

The XREF Proc will perform the following functions:

1. Clear the XSYM file.
2. Use the XREF verb to update the XSYM file.
3. Alphabetically sort the XSYM file and output the results to either the user's terminal or to the system line printer.

The XREF Proc requires the following format:

```
XREF file-name item-list {(options)}
```

Example:

```
: XREF CSYM * (P)(CR)
```

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

The following is an example of a partial listing.

```
                                                    11:2Ø  2Ø DEC 1973

XSYM.....REFERENCES.....

ABIT        EDIT-1

ABSL        DLOAD

ACF         WII

ADDLAB      ASTAT

AF          ASTAT
            WRAPUP-III
            EDIT-1

AFBEG       ASTAT
            EDIT-1

AFEND       ASTAT

ALIGN       ASTAT

AM          XREF
            ASTAT
            T-LOAD
            DLOAD
            WRAPUP-III
            PROC-1
            AID1
            TI
            WII

ATTOVF      WII

B15         MONITOR/295Ø,1ØMB

B4          MONITOR/295Ø,1ØMB

BASE        XREF
            ILOAD
            ASTAT
            T-LOAD
            DLOAD
            WRAPUP-III
            TI
            WII
```

3.7 THE REAL INSTRUCTION REPERTOIRE

In defining the REAL op-codes the following set of symbolic operands are used.

SYMBOL	OPERAND	DESCRIPTION
a	ABS	An absolute core-address reference
b	BIT	A bit addressed relatively via a base address register and a bit displacement.
c	CHARACTER	A byte addressed relatively via a base address register and an 8-bit byte displacement.
d	DOUBLE-WORD	A 4-byte field addressed relatively via a base register and a 16-bit word displacement.
h	HALF-WORD	A 1-byte field addressed relatively via a base register and an 8-bit byte displacement.
l	LABEL	A label definition local to the current program frame.
m	MODE ID	A 16-bit modal identification, comprised of a 4-bit entry point and a 12-bit frame number. The implied location is in the frame defined by the low-order 12-bits of "m", offset from the frame-beginning by twice the entry-point value.
n	LITERAL	A literal or immediate value. The size of the assembled literal or value is dependent on the instruction in which the "n" is used.
r	ADDRESS-REGISTER	One of the sixteen Reality address registers (A/R's).
s	STORAGE-REGISTER	A 6-byte field (usually a storage-register, or S/R) relatively addressed via a base register and a 16-bit word displacement.
t	WORD	A 2-byte field relatively addressed via a base register and a 16-bit word displacement.

Note: The parenthesized footnotes in the following sub-sections are defined in Section 3.7.12.

3.7.1 CHARACTER INSTRUCTIONS (MOVES)

MCC	n,c	(1)	Move Character to Character; the byte (character) defined or addressed by operand-1 is moved to the location addressed by operand-2.
	n,r		
	n,s	(1)	
	c,c		
	c,r		
	c,s	(1)	
	r,c		
	r,r		
	r,s	(1)	
	s,c	(2)	
	s,r	(3)	
	s,s	(2)	
MCI	n,r		Move Character to Incrementing character; the byte (character) pointer operand-2 is incremented by one and the byte defined or addressed by operand-1 is moved to the location then addressed by operand-2.
	n,s	(1)	
	c,r		
	c,s	(1)	
	r,r		
	r,s		
	s,r	(3)	
	s,s	(2)	
MCI	n,r,n	(4)	Move Character Incrementing; the byte (character) pointer operand-2 is incremented by one and the byte defined by operand-1 is moved to the location then addressed by operand-2. This process continues until the number of bytes specified by operand-3 have been moved. At least one byte is always moved and if initially operand-3 = 0, 65,536 bytes will be moved.
	n,r,h	(4)	
	n,r,t	(4)	
	n,r,d	(4)	
MIC	r,c		Move Incrementing character to Character; the byte (character) pointer operand-1 is incremented by one and the byte then addressed by operand-1 is moved to the location addressed by operand-2.
	r,r		
	r,s	(1)	
	s,c	(2)	
	s,r	(3)	
	s,s	(2)	

MII	r,r		Move Incrementing character to Incrementing character; both byte pointers are incremented by one and the byte then addressed by operand-1 is moved to the location addressed by operand-2.
	r,s	(1)	
	s,r	(3)	
	s,s	(2)	
MII	r,r,n	(5)	Move Incrementing character to Incrementing character; both byte pointers are incremented by one and the byte addressed by operand-1 is moved to the location addressed by operand-2. This process is repeated until the number of bytes specified by n,h,t or d have been moved. h,t or d are not destroyed and if initially zero, no bytes are moved.
	r,r,h	(5)	
	r,r,t	(5)	
	r,r,d	(5)	
MII	r,r,r	(3)	Move Incrementing character to Incrementing character; both addressing-registers operand-1 and operand-2 are incremented by one and the byte then addressed by operand-1 is moved to the location addressed by operand-2. This process is repeated until the first addressing-register operand-1 matches the byte-pointer operand-3. If operand-1 = operand-3 on entry no movement takes place.
	r,r,s	(3)	
MIID	r,r,n		Both addressing-registers are incremented by one, and the byte addressed by addressing-register-1 is moved to the location addressed by addressing-register-2. The byte moved is then tested under the following masking condition where "n" is an 8-bit mask field:

Bit	Meaning	
0	True/False	
1	Match on:	X'FF'
2		X'FE'
3		X'FD'
4		X'FC'
5		SC0
6		SC1
7		SC2

Bit 0 is a true/false flag; is set, the move stops on a "match" condition (as defined by bits 1 through 7); if zero, the move stops on a "non-match". Bits 1 through 7 represent one character each; if any bit is set, the byte moved is compared to the

		character represented by the bit for a match. Bits 1 through 4 represent the special system delimiters SM (X'FF'), AM (X'FE'), VM (X'FD'), and SVM (X'FC') respectively. Bits 5, 6, and 7 represent the contents of the scan character-registers SC0, SC1, and SC2 respectively. (Thus only three of the delimiters are variable. NOTE: Character-register SC0 may not contain the hex patterns X'00' or X'01'.
SCD	r,n	Scan characters to delimiter(s). The addressing-register is incremented till a "match" condition (see MIID instruction) as defined by the 8-bit mask field "n" is found.
MIIT	r,r	This instruction assumes that the lower half of the accumulator (T0) has an absolute byte count (up to 65535) defining the number of bytes to be moved (see MII op-code). If T0 is zero when the instruction is executed, no operation is performed. Otherwise, the addressing-registers are incremented by one, and the byte addressed by addressing-register-1 is moved to the location addressed by addressing-register-2, and T0 is decremented by one. This sequence is repeated till T0 reaches zero.
MIIR	r,r	This instruction assumes that address register R15 is setup to a location equal to or greater than that of addressing-register-1. (See MII op-code). If the addresses of addressing-register-1 and register R15 are equal, no operation is performed. Otherwise, the addressing-registers are incremented by one, and the byte addressed by addressing-register-1 is moved to the location addressed by addressing-register-2. This sequence is repeated till the addresses of addressing-register-1 and register R15 are equal.

XCC	c,c	(2)	Exchange Character with Character; the byte
	c,r	(3)	(addressed) by operand-1 is interchanged
	c,s	(2)	with the byte defined by operand-2.
	r,c	(1)	
	r,r		
	r,s	(1)	
	s,c	(2)	
	s,r	(3)	
	s,s	(2)	
OR	c,n	(3)	OR character; the byte (character) addressed
	r,n		by operand-1 is logically or'd with the
	s,n	(3)	8-bit immediate operand-2.
XOR	c,n	(3)	Exclusive OR character; the byte (character)
	r,n		addressed by operand-1 is exclusively or'd
	s,n	(3)	with the 8-bit immediate operand-2.
AND	c,n	(3)	AND character; the byte (character)
	r,n		addressed by operand-1 is logically and'd
	s,n	(3)	with the 8-bit immediate operand-2.

3.7.2 CHARACTER INSTRUCTIONS (TESTS)

BCE	n,c,1	(1)	Branch Character Equal; the byte (character)
	n,r,1		defined or addressed by operand-1 is compared
	c,n,1	(3)	to the byte defined or addressed by operand-2.
	c,c,1		If the two bytes are equal, instruction
	c,r,1		execution branches to the location as defined
	r,n,1		by operand-3. Neither operand-1 nor
	r,c,1		operand-2 are altered. The arithmetic
	r,r,1		condition flag (ACF) is set on c,c,1 only.
BCU	(see BCE)		Branch Character Unequal; branch if characters are not equal.
BCL	(see BCE)		Branch Character Low; branch if operand-1 is less than operand-2.
BCLE	(see BCE)		Branch Character Less than or Equal; branch if operand-1 is less than or equal to operand-2.
BCH	(refer to BCE)		Branch Character High; branch if operand-1 is greater than operand-2.
BCHE	(refer to BCE)		Branch Character High or Equal; branch if operand-1 is greater than or equal to operand-2.

BCN	r,1	Branch if Character is Numeric; branch if the character addressed by the first operand is in the range 0-9, inclusive.
BCX	r,1	Branch if Character is hexadecimal; branch if the character addressed by the first operand is in the range 0-9 or A-F, inclusive.
BCA	r,1	Branch if Character is Alphabetic; branch if the character addressed by the first operand is in the range A-Z, inclusive.

3.7.3 BIT INSTRUCTIONS

SB	b	Set Bit; the bit addressed by the operand is set to an on condition (one).
ZB	b	Zero Bit; the bit addressed by the operand is set to an off condition (zero).
BBS	b,1	Branch Bit Set; the bit addressed by operand-1 is tested and if set (one) instruction execution branches to the location defined by operand-2.
BBZ	b,1	Branch Bit Zero; the bit addressed by operand-1 is tested and if not set (zero) instruction execution branches to the location defined by operand-2.

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

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

STORE	h t d		STORE from accumulator; the contents of the 32-bit accumulator (D0) are stored into the location defined by operand-1. For half-word and word operands, the high order bits are lost.
ADD	n h t d		ADD to accumulator; the integer addressed by operand-1 is added to the 32-bit accumulator (D0) with sign extension.
SUB	n h t d		SUB from accumulator; the integer addressed by operand-1 is subtracted from the 32-bit accumulator (D0) with sign extension.
MUL	n h t d		MULtiply to accumulator; the integer addressed by operand-1 is multiplied by the contents of the 32-bit accumulator (D0). The resulting product is stored in the 64-bit accumulator extension (D1,D0), as a 63 bit number and a duplicated sign bit.
DIV	n h t d		DIVide into the accumulator; the integer addressed by operand-1 is divided into the 32-bit accumulator (D0). The answer is stored in D0 and the integer remainder is stored into the accumulator extension (D1).

3.7.5 REGISTER INSTRUCTIONS

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

DEC	r s		DECrement register; the address or storage register operand-1 is decremented by one.
DEC	r,n r,h r,t r,d s,n s,h s,t s,d		DECrement register by count; the address or storage register operand-1 is decremented by the integer stored at the location addressed by operand-2.
LAD	r,r r,s s,r s,s	(7) (1)	Load Absolute Difference; the absolute difference in bytes (characters) between the byte pointer operand-1 and the byte pointer operand-2 is computed and stored into the lower half of the accumulator (T0). Please see special note following Branch Register Equal/Unequal instructions.
SRA	r,c r,h r,t r,d r,s r,l		Set Register to Address; the byte pointer operand-1 is set pointing to the first byte of the functional element at the location addressed by operand-2.
FAR	r,n		Flag and Attach Register; the address-register operand-1 is attached and secondary processing as defined by the 8-bit literal n is performed: 0 0 - I/O busy for buffer 1 not used 2 0 - buffer core-locked 3 0 - write-required 4 not used 5 Set-up R15 to 1st byte unlinked; old buffer status in R15DSP 6 Change buffer FID 7 0 - OR n with status, 1 - AND n with status In normal execution only n2 is effective; the remainder of the functions can only be evoked in the "monitor mode."

BE BU	r,r,1 r,s,1 s,r,1	(7) (10)	Branch Register Equal/Unequal; the address of the byte pointer operand-1 is compared to the address of the byte pointer operand-2. The branch is taken appropriately. If the FID's of the registers are unequal, it is assumed that the affected frames are contiguously linked and the address computation is made on that basis; further, the difference in the FID's is assumed to be less than or equal to thirty-two (32); therefore the instruction execution may prove incorrect if 1) The FID's are unequal and not contiguously linked or 2) One of the registers is in an unlinked format, and the other is not. (For special information regarding the BE and BU instructions, refer to Section 3.14.)
BE BU	s,s,1		Branch Register Equal/Unequal; the 6-byte storage register operand-1 is arithmetically compared to the storage register operand-2 and the branch is made accordingly.

3.7.6 DATA COMPARISON INSTRUCTIONS

BE	n,h,1 n,t,1 n,d,1 h,n,1 h,h,1 h,t,1 h,d,1 t,n,1 t,h,1 t,t,1 t,d,1 d,n,1 d,h,1 d,t,1 d,d,1	(6) (6) (6) (6) (6) (6) (6) (6) 	Branch word Equal; the integer stored in the word addressed by operand-1 is compared arithmetically (2's complement) to the integer stored in the word addressed by operand-2. If an equal comparison is made, instruction branches to the location defined by operand-3.
BU	(see BE)		Branch word Unequal; branch if words are unequal.
BL	(see BE)		Branch word Low; branch if operand-1 is less than operand-2.
BLE	(see BE)		Branch word Low or Equal; branch if operand-1 is less than or equal to operand-2.

BH	(see BE)	Branch word High; branch if operand-1 is greater than operand-2.
BHE	(see BE)	Branch word High Equal; branch if operand-1 is greater than or equal to operand-2.
BDZ	h,h,l t,n,l t,t,l d,n,l d,d,l	Branch on Decrementing word Zero; the word at the location addressed by operand-1 is decremented by the integer at the location addressed by operand-2. If the result is zero, instruction branches to the location defined by operand-3.
BDNZ	(see BDZ)	Branch on Decrementing word not Zero; same as BDZ but branch on result not zero.
BDLZ	(see BDZ)	Branch on Decrementing word Less than Zero; same as BDZ but branch on result less than zero.
BDLEZ	(see BDZ)	Branch on Decrementing word Less than or Equal to Zero; same as BDZ but branch on result less than or equal to zero.
BDZ	t,l d,l	Branch on Decrementing word Zero; same as BDZ above but decrement by one.
BDNZ	t,l d,l	Branch on Decrementing word not Zero; same as BDNZ above but decrement by one.
BDLZ	t,l d,l	Branch on Decrementing word Less than Zero; same as BDLZ above but decrement by one.
BDLEZ	t,l d,l	Branch on Decrementing word Less than or Equal to Zero; same as BDLEZ above but decrement by one.

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

3.7.7 TRANSLATE INSTRUCTIONS

MBD MBDN	h,r t,r d,r n,h,r n,t,r n,d,r	Move Binary word to Decimal characters; This macro generates a call to the subroutine MBDSUB (MBD) or MBDNSUB (MBDN), which converts from a binary integer at the location addressed by operand-1 to a string of decimal ASCII characters, stored beginning from the location addressed by the byte-pointer operand-2 plus one.

			The following elements are used by the subroutine and macro D0; D1; D2; T4; T5; R14; R15. A minus sign will precede the converted value if it was negative; at the conclusion of the instruction, the byte pointer operand-2 addresses the last converted byte. MBD deletes leading zeros, but converts at least one character; MBDN converts at least "n" characters, padded with leading zeros if necessary.
MDB	r,t r,d		Move Decimal character to Binary word; ASCII decimal to binary conversion. The word at the location addressed by operand-2 is multiplied by 10, and a value (as defined above for the MXT instruction) from the byte addressed by the addressing register is added to it. The arithmetic condition flags are not reset, and arithmetic overflow cannot be detected.
MBX	h,r t,r d,r		Move Binary word to heXadecimal characters; Binary to ASCII hex conversion. This instruction assumes that the least significant byte of the accumulator (H0) has a parameter (see MBX/MBXN macro). Bits 3-0 contain a digit count, specifying the maximum number of ASCII digits to be converted. As each digit is converted, the addressing register is incremented by one, and the converted ASCII character is stored in the location addressed by the addressing register. The format of H0 at the conclusion of this instruction is unpredictable. If the digit count in H0 exceeds the field defined by operand-1, no operation is performed.
MBX MBXN	n,h,r n,t,r n,d,r	(9)	Move Binary word to heXadecimal characters; This macro expands as a LOAD of the first operand (MBX) or the first operand +X'80' (MBXN), and an primitive. The MBX macro, therefore, causes conversion from binary to ASCII hex, with only significant digits (to a maximum of "n") converted. The MBXN macro causes conversion as above, but always converts "n" digits, with leading zeros if necessary. The addressing register defined by the third operand is incremented before each byte converted.

MXB	r,h r,t r,d	Move heXadecimal characters to Binary word; ASCII hex to binary conversion The field defined by operand-2 is shifted left 4 bits, and the value defined below, from the byte addressed by the addressing register, is added to the field: The 4-bit value from bits 3-0 of the byte (bits numbered right to left), plus nine times bit 5. The arithmetic condition flags are not reset by this instruction, and arithmetic overflow cannot be detected.

3.7.8 EXECUTION TRANSFER INSTRUCTIONS

B	l a	Branch; branch to location defined, in the current frame, defined by label "l", or the absolute core address "a". The branch to an absolute core-address is a privileged instruction executable only at the monitor-level.
BSL	l m a	Branch and Stack Location; Subroutine call to mode defined by mode-ID "m", local label "l", or absolute core-address "a". (The BSL to an absolute core-address is a privileged instruction executable at the monitor-level only). The location-1 of the instruction following the BSL is saved in the return stack, and the next instruction executed is that defined by the operand. The return stack level is increased by one; if the call causes the return stack level to exceed its maximum value, the stack pointers are reset to the beginning and a trap to the DEBUG mode is executed.
BSLI		Branch and Stack Location Indirect; Subroutine call indirect; this instruction assumes that the lower half of the accumulator, T0 contains a mode-ID (see BSL* macro). The 16-bit mode-ID contained in T0 defines the location of the next instruction that is to be executed, after the location-1 of the instruction following the TCI is saved in the return stack.

RTN			ReTurN; Return to subroutine called. The last entry in the return stack defines the location of the next instruction to be executed; the return stack level is decremented by one. If the return stack is empty, a trap to the DEBUG mode is executed. A return instruction from a subroutine called via a local call, or an absolute core-address call, will return within the current 512-byte frame only.
ENT	m		ExterNal Transfer; Branch to location defined by mode-ID"m".
ENTI			ExterNal Transfer Indirect; Enter mode indirect: this instruction assumes that T0 contains a 16-bit mode-ID (see ENT* macro), which defines the next instruction to be executed.
BSL*	h t d	(8)	Branch and Stack Location indirect; subroutine call to mode defined by the mode-ID contained in the word addressed by operand-1. The 16 bit mode-ID is loaded into the accumulator, and a BSLI instruction is executed.
ENT*	h t d	(8)	ExterNal Transfer indirect; branch to external location defined by the mode-ID contained in the word addressed by operand-1. The 16 bit mode-ID is loaded into the accumulator, and an ENTI instruction is executed.

3.7.9 I/O AND CONTROL INSTRUCTION

IOI	r,n_1,n_2	I/O Instruction Input; this instruction is used to control input from peripheral devices whose device addresses are in the range 0 through X'F' (15). This instruction causes an MCAL instruction to entry point 8 in the Monitor. Register r points to the start of the input buffer; n_1 is a 3-bit order code; n_2 is a 4-bit device address. Refer to Reality CPU, peripheral I/O for details.

IOO	r,n_1,n_2	I/O instruction Output; as above this instruction output to peripheral devices.
READ	r	A byte from the byte-I/O buffer in the PIB is stored at the location addressed by the addressing register. If the buffer is empty, or if there is data in the byte I/O) buffer yet to be output to the byte I/O device, the process executing the READ instruction will enter a quiescent state till data from the byte input device causes a re-activation.
WRITE	r	The byte addressed by the addressing register is moved into the byte I/O buffer of the PIB. If the buffer is empty, the byte is also output immediately to the byte I/O device. If the buffer is full, the process executing the write will enter a quiescent state till the byte output device has accepted the data from the buffer, and causes a re-activation. Execution of this instruction causes a loss of any input data in the byte I/O buffer, and inhibits any further data input from the byte I/O device.
MCAL	r,n_1,n_2	Monitor call to entry point "n_2" ($7 < n_2 < 16$). The word address of the addressing register; the 8-bit address of the addressing register; the 8-bit mask n_1; and the location of the PCB are passed, as parameters to the monitor, in the PIB. MCAL r,5,11 sets CORELOCK and CORELOCK1 bits in the buffer status byte indicated by register "r". MCAL r,6,11 resets these same bits.
RQM		Process releases the remainder of its time quantum to the monitor. Equivalent to: MCAL 0,0,9.
IB OB	r,n	Input/Output byte instruction. Refer to Reality CPU, peripheral I/O for details. The byte defined by the mask "n" is output as a control byte, and a data byte is input (IB) and stored at the location addressed by the addressing register, or output (OB) from the location addressed by the addressing register. These instructions are allowable in monitor mode only.

NOP		No OPeration is performed by this instruction.
HALT		This instruction halts the CPU and is executable in the Monitor mode only.
HLD	r	Halts the CPU and gates the eight-bit literal addressed by register r to the A bus, where it can be displayed in the least significant indicator lamps of the system panel by depressing the Data select switch. Executable in monitor mode only.
TEXT X'01'		Tests internal and external interrupts and traps to the appropriate monitor location if any interrupts are pending. Executable in monitor mode only.
ECS	r	The status of the eight low-order console command switches is placed in the byte addressed by register r. Executable in monitor mode only.
ESS	r	The status of the four console sense switches is placed in the four most significant bits of the byte addressed by register r.
SVP		Start virtual Process (Monitor level only) The 2-byte FID located at absolute core address X'127', X'128' (R4-FID of monitor) is treated as a PCB-FID, and the buffer-pool searched for a match. If found, register zero in the PCB is setup in an "attached" format, and the attachment process for register one (user program-mode register) is started. If not found, a frame fault request on the PCB-FID is stacked, and the monitor is re-entered.
RVP		Return to Virtual Process (monitor level only) should be executed when a trap to the monitor due to an external interrupt by devices 0-X'15' has caused a monitor trap. Selects primary file registers of the 1600 computer and resumes execution of the virtual Process. If this instruction is executed when the system is not in an interrupt-handling mode, no operation takes place.

5.7.10 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 op code, which ignores the operand field.
	CMNT		Comment; the contents of this statement are treated as commentary, and ignored by the assembler. Note: A label field entry is allowable.
l	DEFA	a	Defines the local symbol "l" to be of type a.
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	EQU	c h t d s l n	Equates the local label "1" to the symbol or literal value of the operand.
	FRAME	n	Must be the first assembled statement in a mode that is to be loaded; "n" defines the frame on which the object code is to be loaded.
	ORG	1 n	Resets the location counter to value defined by the operand. This statement may have a label field entry.
	SETAR	r	Causes all literals encountered from this point in the assembly to be defined as a displacement relative to register r. If no SETAR occurs, SETAR 1 is assumed.

TEXT	X'...' C'...'	Assembles binary equivalent of character strings (enclosed in quotes and preceded by a 'C') or hexadecimal values. Any number and combination of C and X literals separated by commas is permitted.

3.7.11 ADDRESS REGISTER USAGE

A storage operand is always referenced through an address register containing the byte address of the operand. For instructions with a D field, a displacement is added to the contents of the address register to form an effective address. The length of the operand is encoded in the L field of the instruction. (Refer to Section 2.5.)

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
Rn;Sm	6*m bytes	6 bytes

Example:

MCC	R0;C15,R15	Move low order byte of the Accumulator to the byte addressed by R15.

Example:

SB	R5;B0	Set bit 0 of the byte addressed by R5.

Example:

MOV	MBASE,R10;D4	Move double-word MBASE to the double-word starting 16 bytes past the byte addressed by R10.

3.7.12 REAL INSTRUCTION SIDE EFFECTS

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

(10) See Appendix A.

3.7.13 EXAMPLES

The following listing presents examples of the REAL instructions.

```
001                ************************************************************************
002                *
003                * REAL INSTRUCTION REPERTOIRE
004                *
005                ************************************************************************
006                *
007                * DEFINE SYMBOLIC OPERANDS USED IN DEFINITIONS
008                *
009                B1        DEFB  1,11
010                B2        DEFB  2,22
011                C1        DEFC  1,11
012                C2        DEFC  2,22
013                H1        DEFH  1,11
014                H2        DEFH  2,22
015                T1        DEFT  1,11
016                T2        DEFT  2,22
017                D1        DEFD  1,11
018                D2        DEFD  2,22
019                S1        DEFS  1,11
020                S2        DEFS  2,22
021                *
022                * DEFINE FUNCTIONAL ELEMENTS USED IN MACRO EXPANSIONS
023                *
024                T0        DEFT  0,7
025                D0        DEFD  0,6
026                A1        DEFA  X'1234'
027                M1        DEFM  1,2
028                *
029                *
030                *
031                ************************************************************************
032                *
033                * CHARACTER OPERATIONS
034                *
035                ************************************************************************
```

```
036                 * MOVE CHARACTER TO CHARACTER
037                 LØØ       EQU   *
038 ØØ1 E2163F               MCC   C'A',C2          ONE CHARACTER MOVED
    ØØ4 4F4120
039 ØØ7 424120               MCC   C'A',R2
040 ØØA E216EF               MCC   C'A',S2
    ØØD 4F4120
041                 *
042 Ø1Ø F216Ø1ØB             MCC   C1,C2
043 Ø14 D1ØB12               MCC   C1,R2
044 Ø17 E216EF               MCC   C1,S2
    Ø1A D1ØB1F
045                 *
046 Ø1D D216Ø1               MCC   R1,C2
047 Ø2Ø 6219                 MCC   R1,R2
048 Ø22 E216EF               MCC   R1,S2
    Ø25 6F19
049                 *
050 Ø27 E2163E               MCC   S1,C2
    Ø2A E1ØBEF
    Ø2D 6EF9
051 Ø2F E1ØBEF               MCC   S1,R2
    Ø32 62F9
052 Ø34 E1ØBEE               MCC   S1,S2
    Ø37 E216EF
    Ø3A 6F39
053                 *
054                 * MOVE CHARACTER TO CHARACTER, INCREMENTING DESTINATION
055 Ø3C 424140               MCI   C'A',R2          ONE CHARACTER MOVED
056 Ø3F A21643               MCI   C'A',S2
    Ø42 E216EF
    Ø45 4F4120
057                 *
058 Ø48 32                   MCI   C1,R2
    Ø49 D1ØB12
```

```
059 04C E216EF          MCI   C1,S2
    04F 3F
    050 D10B1F
    053 E216DF
060                 *
061 056 621A            MCI   R1,R2
062 058 E216EF          MCI   R1,S2
    05B 6F1A
    05D E216DF
063                 *
064 060 E10BEF          MCI   S1,R2
    063 62FA
065 06C E10BEE          MCI   S1,S2
    068 E216EE
    068 6FEA
    06D E216DF
066                 *
067                 *
056 070 162F            MCI   C'A',R2,7       MOVE # CHARACTERS IN OPERAND 3
    072 424140
    075 A72E58
    078 6F24
069 07A A10B18          MCI   C'A',R2,H1
    07D A00685
    080 162F
    082 424140
    085 6F24
070 087 A10B58          MCI   C'A',R2,T1
    08A A00685
    08D 162F
    08F 424140
    092 6F24
071 094 A10B98          MCI   C'A',R2,D1
    097 A00685
    09A 162F
```

```
    09C 424140
    09F 6F24
072                 *
073                 * MOVE CHARACTER TO CHARACTER INCREMENTING SOURCE
074 0A1 31                  MIC   R1,C2             MOVE 1 CHARACTER
    0A2 D21601
075 0A5 6121                MIC   R1,R2
076 0A7 E216EF              MIC   R1,S2
    0AA 61F1
077                 *
078 0AC A10B43              MIC   S1,C2
    0AF E2163E
    0B2 E10BEF
    0B5 6EF9
079 0B7 E10BEF              MIC   S1,R2
    0BA 6F21
    0BC E10BDF
080 0BF E10BEF              MIC   S1,S2
    0C2 E216EE
    0C5 6FE1
    0C7 E10BDF
081                 *
082                 * MOVE CHARACTER TO CHARACTER, INC SOURCE AND DESTINATION
083 0CA 6122                MII   R1,R2             MOVE 1 CHARACTER
084 0CC E216EF              MII   R1,S2
    0CF 61F2
    0D1 E216DF
085                 *
086 0D4 E10BEF              MII   S1,R2
    0D7 6F22
    0D9 E10BDF
087 0DC E10BEE              MII   S1,S2
    0DF E216EF
    0E2 6EF2
    0E4 E10BDE
```

```
    0E7 E216DF
088                 *
089 0EA A72F58              MII   R1,R2,80          MOVE # CHARACTERS IN OPERAND 3
    0ED 6124
090 0EF A10B18              MII   R1,R2,H1
    0F2 6124
091 0F4 A10B58              MII   R1,R2,T1
    0F7 6124
092 0F9 A10B98              MII   R1,R2,D1
    0FC 6124
093                 *
094 0FE 163F                MII   R1,R2,R3          MOVE UNTIL 2ND OPERAND = 3RD OPERAND
    100 6123
095 102 E24FEF              MII   R1,R2,S3
    105 6123
096                 *
097                 * INSTRUCTIONS INCREMENTING SOURCE & DESTINATION REPEATEDLY
098 107 6120E0              MIID  R1,R2,X'E0'       MOVE CHAR TO CHAR THROUGH DELIMITER
099                 *
100 10A 6108A0              SCD   R1,X'A0'          SCAN CHARACTERS TO DELIMETER
101                 *
102 10D 6124                MIIT  R1,R2             MOVE NUMBER OF CHARS.  IN ACCUMULATOR
103                 *
104 10F 6123                MIIR  R1,R2             MOVE UNTIL R1 AND R15 ARE EQUAL
105                 *
106                 *EXCHANGE CHARACTER WITH CHARACTER
107 111 E10B3E              XCC   C1,C2
    114 E2163F
    117 6EF7
108 119 E10B3F              XCC   C1,R2
    11C 6F27
109 11E E10B3E              XCC   C1,S2
    121 E216EF
    124 6EF7
110                 *
```

```
111 126 E2163F                XCC   R1,C2
    129 6F17
112 12B 6127                  XCC   R1,R2
113 12D E216EF                XCC   R1,S2
    130 61F7
114                 *
115 132 E10BEE                XCC   S1,C2
    135 E2163F
    138 6EF7
116 13A E10BEF                XCC   S1,R2
    13D 6F27
117 13F E10BEE                XCC   S1,S2
    142 E216EF
    145 6EF7
118                 *
119                 *LOGICAL OR CHARACTER WITH MASK
120 147 E10B3F                OR    C1,X'AB'
    14A 4FABC0
121 14D 41ABC0                OR    R1,X'AB'
122 150 E10BEF                OR    S1,X'AB'
    153 4FABC0
123                 *
124                 *EXCLUSIVE OR WITH MASK
125 156 E10B3F                XOR   C1,X'AB'
    159 4FABD0
126 15C 41ABD0                XOR   R1,X'AB'
127 15F E10BEF                XOR   S1,X'AB'
    162 4FABD0
128                 *
129                 *LOGICAL AND CHARACTER WITH MASK
130 165 E10B3F                AND   C1,X'AB'
    168 4FABE0
131 168 41ABE0                AND   R1,X'AB'
132 16E E10BEF                AND   S1,X'AB'
    171 4FABE0
```

```
133                     *
134                     *
135 174 41FEF0                 SHIFT X'FE',R1
136                     *
137                     *
138                     *
139                     *
140                     ****************************************
141                     *
142                     * CHARACTER INSTRUCTIONS (TESTS)
143                     *
144                     ****************************************
145                     *
146                     *
147                     *BRANCH CHARACTER EQUAL
148 177 E2163F                 BCE   C'A',C2,L1
    17A 4F410847
149 17E 42410843                BCE   C'A',R2,L1
150                     *
151 182 E10B3F
    185 4F41083C
152 189 F10B1216                BCE   C1,C2,L1
    18D 5836
153 18F B10B2832                BCE   C1,R2,L1
154                     *
155 193 4141082E                BCE   41,C'A',L1
156 197 B216182A                BCE   R1,C2,L1
157 19B 512827                  BCE   R1,R2,L1
158                     *
159                     *BRANCH CHARACTER UNEQUAL
160 19E E2163F                  BCU   C'A',C2,L1
    1A1 4F410020
161 1A5 4241001C                BCU   C'A',R2,L1
162 1A9 E10B3F                  BCU   C1,C'A',L1
    1AC 4F410015
```

```
163                     *
164 1B0 F10B1216                BCU   C1,C2,L1
    1B4 500F
165 1B6 B10B200B                BCU   C1,R2,L1
166                     *
167 1BA 41410007                BCU   R1,C'A',L1
168 1BE B2161003                BCU   R1,C2,L1
169 1C2 512000                  BCU   R1,R2,L1
170                     *
171                     *BRANCH CHARACTER LOW
172                     L1      EQU   *
173 105 E2163F                  BCL   C'A',C2,L1
    1C8 4F410607
174 1CC 4241060B                BCL   C'A',R2,L1
175 1D0 E10B3F                  BCL   C1,C'A',L1
    1D3 4F410C02
    1D7 1E14
176 1D9 F10B1216                BCL   C1,C2,L1
    1DD 561A
177 1DF B10B261E                BCL   C1,R2,L1
178 1E3 41410C02                BCL   R1,C'A',L1
    1E7 1E24
179 1E9 B2161C19                BCL   R1,C2,L1
    1ED 1E2A
180 1EF 51262D                  BCL   R1,R2,L1
181                     *
182                     *BRANCH CHARACTER LOW OR EQUAL
183 1F2 E2163F                  BCLE  C'A',C2,L1
    1F5 4F410E34
184 1F9 42410E38                BCLE  C'A',R2,L1
185 1FD E10B3F                  BCLE  C1,C'A',L1
    200 4F410402
    204 1E41
186 206 F10B1216                BCLE  C1,C2,L1
    20A 5247
```

```
187 20C B10B2E4B              BCLE  C1,R2,L1
188 210 41410402              BCLE  R1,C'A',L1
    214 1E51
189 216 B2161402              BCLE  R1,C2,L1
    21A 1E57
190 21C 512E5A                BCLE  R1,R2,L1
191                     *
192                     *BRANCH CHARACTER GREATER
193 21F E2163F                BCH   C'A',C2,L1
    222 4F410C02
    226 1E63
194 228 42410C02              BCH   C'A',R2,L1
    22C 1E69
195 22E 310B3F                BCH   C1,C'A',L1
    231 4F410670
196 235 F216110B              BCH   C1,C2,L1
    239 5676
197 23B B10B2C16              BCH   C1,R2,L1
    23F 1E7C
198 241 41410680              BCH   R1,C'A',L1
199 245 B2161684              BCH   R1,C2,L1
200 249 521687                BCH   R1,R2,L1
201                     *
202                     *BRANCH CHARACTER GREATER OR EQUAL
203 24C E2163F                BCHE  C'A',C2,L1
    24F 4F410402
    253 1E90
204 255 42410402              BCHE  C'A',R2,L1
    259 1E96
205 25B E10B3F                BCHE  C1,C'A',L1
    25E 4F410E9D
206 262 F216110B              BCHE  C1,C2,L1
    266 5EA3
207 268 B10B2402              BCHE  C1,R2,L1
    26C 1EA9
```

```
208 26E 41410EAD              BCHE  R1,C'A',L1
209 272 B2161EB1              BCHE  R1,C2,L1
210 276 521EB4                BCHE  R1,R2,L1
211                  *CHARACTER TYPE TESTS
212 279 413A0C04              BCN   R1,L1             BRANCH CHARACTER NUMERIC
    27D 41300EBC
213 281 41470C0C              BCX   R1,L1             BRANCH CHARACTER HEXADECIMAL
    285 41410EC4
    289 413A0C04
    28D 41300ECC
214 291 415B0C04              BCA   R1,L1             BRANCH CHARACTER ALPHABETIC
    295 41410ED4
215                  *
216                  *
217                  ***********************************************
218                  *
219                  * BIT INSTRUCTIONS
220                  *
221                  ***********************************************
222                  *
223                  *
224                  *
225                  *
226 299 810B                  SB    B1                SET BIT
227 298 710B                  ZB    B1                ZERO BIT
228 29D 910B0000              BBS   B1,L4             BRANCH BIT SET
229 2A1 910B0A04     L4       BBZ   B1,L4             BRANCH BIT ZERO
230                  *
231                           CMNT  *                 SEE MOV BIT TO BIT BELOW
232                  ***********************************************
233                  *
234                  * DATA MOVEMENT AND ARITHMETIC INSTRUCTIONS
235                  *
236                  ***********************************************
237                  *
```

```
238                     *MOVE DATA TO DATA BY AREA
239 2A5 A73058                  MOV   32,H2
    2A8 A21619
240 2AB F2164730                MOV   32,T2
241 2AF F2168744                MOV   32,D2
242                     *
243 2B3 F216Ø1ØB                MOV   H1,H2
244 2B7 A1ØB18                  MOV   H1,T2
    2BA A21659
245 2BD A1ØB18                  MOV   H1,D2
    2CØ A21699
246                     *
247 2C3 A1ØB58                  MOV   T1,H2
    2C6 A21619
248 2C9 F216410B                MOV   T1,T2
249 2CD A1ØB58                  MOV   T1,D2
    2DØ A21699
25Ø                     *
251 2D3 A1ØB98                  MOV   D1,H2
    2D6 A21619
252 2D9 A1ØB98                  MOV   D1,T2
    2DC A21659
253 2DF F216810B                MOV   D1,D2
254                     *
255 2E3 7216                    MOV   B1,B2             MOVE BIT TO BIT
    2E5 91ØBØ8Ø2
    2E9 8216
256                     *
257                     *TEST AND SET ARITHMETIC FLAGS
258 2EB A1ØBØ2                  TST   H1
259 2EE A1ØB42                  TST   T1
26Ø 2F1 A1ØB82                  TST   D1
261                     *
262                     *
263                     *INCREMENT INSTRUCTIONS
```

```
264 2F4 A10B03              INC   H1              INCREMENT DATA BY 1
265 2F7 A10B43              INC   T1
266 2FA A10B83              INC   D1
267                 *
268 2FD A10B18              INC   H1,32           INCREMENT DATA AREA BY DATA
    300 A73053
    303 A10B19
269 306 F10B2216            INC   H1,H2
270 30A A10B18              INC   H1,T2
    30D A21653
    310 A10B19
271 313 A10B18              INC   H1,D2
    316 A21693
    319 A10B19
272                 *
273 31C F10B6730            INC   T1,32
274 320 A21618              INC   T1,H2
    323 F10B6007
275 327 F10B6216            INC   T1,T2
276 32B A21698              INC   T1,D2
    32E F10B6007
277                 *
278 332 F10BA744            INC   D1,32
279 336 A21618              INC   D1,H2
    339 F10BA006
280 33D A2A658              INC   D1,T2
    340 F10BA006
281 344 F10BA216            INC   D1,D2
282                 *
283                 *DECREMENT INSTRUCTIONS
284 348 A10B05              DEC   H1              DECREMENT DATA AREA BY 1
285 348 A10B45              DEC   T1
286 34E A10B85              DEC   D1
287
288 351 A10B18              DEC   H1,32           DECREMENT DATA AREA BY DATA
```

```
    354 A73055
    357 A10B19
289 35A F10B3216            DEC   H1,H2
290 35E A10B18              DEC   H1,T2
    361 A21655
    364 A10819
291 367 A10B18              DEC   H1,D2
    36A A21695
    36D A10B19
292                 *
293 370 F10B7730            DEC   T1,32
294 374 A21618              DEC   T1,H2
    377 F10B7007
295 378 B10B7216            DEC   T1,T2
296 37F A21698              DEC   T1,D2
    382 F10B7007
297                 *
298 386 F10BB744            DEC   D1,32
299 38A A21618              DEC   D1,H2
    38D F10BB006
300 391 A21658              DEC   D1,T2
    394 F10BB006
301 398 F10BB216            DEC   D1,D2
302                 *
303                 *ZERO OUT DATA AREA
304 39C A10B00              ZERO  H1
305 39F A10B40              ZERO  T1
306 3A2 A10B80              ZERO  D1
307                 *REPLACE DATA AREA WITH NUMBER 1
308 3A5 A10B01              ONE   H1
309 3A8 A10B41              ONE   T1
310 3AB A10B81              ONE   D1
311                 *NEGATE DATA AREA
312 3AE A10B08              NEG   H1
313 3B1 A10B48              NEG   T1
```

```
314 3B4 A10B88                NEG   D1
315                  *LOAD DATA INTO ACCUMULATOR
316 3B7 A73C58                LOAD  876
317 3BA A10B18                LOAD  H1
318 3BD A10B58                LOAD  T1
319 3C0 A10B98                LOAD  D1
320                  *STORE ACCUMULATOR INTO DATA AREA
321 3C3 A10B19                STORE H1
322 3C6 A10B59                STORE T1
323 3C9 A10B99                STORE D1
324                  *ADD DATA TO ACCUMULATOR
325 3CC A74653                ADD   6547
326 3CF A10B13                ADD   H1
327 3D2 A10B53                ADD   T1
328 3D5 A10B93                ADD   D1
329                  *SUBTRACT DATA FROM ACCUMULATOR
330 3D8 A74755                SUB   643
331 3DB A10B15                SUB   H1
332 3DE A10B55                SUB   T1
333 3E1 A10B95                SUB   D1
334                  *MULTIPLY ACCUMULATOR BY DATA
335 3E4 A73A50                MUL   23
336 3E7 A10B10                MUL   H1
337 3EA A10B50                MUL   T1
338 3ED A10B90                MUL   D1
339                  *DIVIDE ACCUMULATOR BY DATA
340 3F0 A73151                DIV   23
341 3F3 A10B11                DIV   H1
342 3F6 A10B51                DIV   T1
343 3F9 A10B91                DIV   D1
344                  *
345                  ******************************************
346                  *
347                  * REGISTER INSTRUCTIONS
348                  *
```

```
349                     ****************************************
350                     *
351                     *MOV REGISTER TO REGISTER
352 3FC 1612                       MOV   R1,R2
353 3FE E216D1                     MOV   R1,S2
354 401 E10BE2                     MOV   S1,R2
355 404 F216C10B                   MOV   S1,S2
356                     *EXCHANGE REGISTER CONTENTS
357 408 1712                       XRR   R1,R2
358 40A E10BEF                     XRR   R1,S1
    40D E10BD1
    410 16F1
359 412 161F                       XRR   S1,R1
    414 E10BE1
    417 E10BDF
360 41A E216EF                     XRR   S1,S2
    41D F216C10B
    421 E10BDF
361                     *INCREMENT REGISTER
362 424 31                         INC   R1              BY 1
363 R25 A10B43                     INC   S1
364 428 E73251                     INC   R1,20           BY 2ND OPERAND
365 42B A10B18                     INC   R1,H1
    42E E00751
366 431 E10B51                     INC   R1,T1
367 434 A10B98                     INC   R1,D1
    437 E00751
368                     *
369 43A F10B6732                   INC   S1,20
370 43E A10B18                     INC   S1,H1
    441 F10B6007
371 445 F10B610B                   INC   S1,T1
372 449 A10B98                     INC   S1,D1
    44C F10B6007
373                     *DECREMENT REGISTER
```

```
374 450 21                  DEC   R1              BY 1
375 451 A10B45              DEC   S1
376                 *
377 454 E74841              DEC   R1,25           BY 2ND OPERAND
378 457 A10B18              DEC   R1,H1
    45A E00741
379 45D E10B41              DEC   R1,T1
380 460 A10B98              DEC   R1,D1
    463 E00741
381 466 F10B7749            DEC   S1,1000
382 46A A10B18              DEC   S1,H1
    46D F10B7007
383 471 F10B710B            DEC   S1,T1
384 475 A10B98              DEC   S1,D1
    478 F10B7007
385                 *
386                 *LOAD ABSOLUTE ADDRESS DIFFERENCE
387                 L2      EQU   *
388 47C E0F6D1              LAD   R1,R2
    47F E0F6C2
389 482 E216C1              LAD   R1,S2
390 485 E10BC2              LAD   S1,R2
391 488 E216EF              LAD   S1,S2
    48B E10BCF
392                 *SET REGISTER TO ADDRESS
393 48E E21631              SRA   R1,C2
394 491 E21631              SRA   R1,H2
395 494 E21671              SRA   R1,T2
396 497 E216B1              SRA   R1,D2
397 49A E216F1              SRA   R1,S2
398 49D E10131              SRA   R1,L00
399                 *FLAG AND ATTACH ADDRESS REGISTER
400 4A0 423460              FAR   R2,X'34'
401                 *BRANCH REGISTER EQUAL (UNEQUAL)
402                 L3      EQU   *
```

```
403 4A3 E0F9D1              BE    R1,R2,L3
    4A6 C0F92A07
404 4AA C2161A0B            BE    R1,S2,L3
405 4AE F216D10B            BE    S1,S2,L3
    4B2 5A11
406 4B4 E0F9D1              BU    R1,R2,L3
    4B7 C0F92218
407 4BB C216121C            BU    R1,S2,L3
408 4BF C10B2220            BU    S1,R2,L3
409 4C3 C10B2A24            BE    S1,R2,L3
410 4C7 F216D10B            BU    S1,S2,L3
    4CB 522A
411                 *
412                 *
413                 *
414                 ***********************************************
415                 *
416                 * DATA COMPARISON INSTRUCTIONS
417                 *
418                 ***********************************************
419                 *
420                 *BRANCH 1ST OPERAND = 2ND OPERAND
421 4CD E10B3F              BE    25,H1,L6
    4D0 4F190949
422 4D4 F733510B            BE    26,T1,L6
    4D8 5943
423 4DA F734910B            BE    27,D1,L6
    4DE 593D
424                 *
425 4E0 A10B18              BE    H1,X'25',L6
    423 F0075736
    4E7 5934
426 4E9 F10B110B            BE    H1,H1,L6
    4ED 592E
427 4EF A10B18              BE    H1,T1,L6
```

```
    4F2 F00751ØB
    4F6 5925
428 4F8 A1ØB18              BE    H1,D1,L6
    4FB F00691ØB
    4FF 591C
429                 *
430 501 F1ØB573D            BE    T1,12345,L6
    505 3916
431 507 A1ØB18              BE    T1,H1,L6
    50A F1ØB500?
    50E 590D
432 510 F1ØB5216            BE    T1,T2,L6
    514 5907
433 516 A1ØB58              BE    T1,D1,L6
    519 F00691ØB
    51D 58FE
434                 *
435 51F F1ØB974A            BE    D1,X'1234',L6
    523 58F8
436 525 A1ØB18              BE    D1,H1,L6
    528 F1ØB9ØØ6
    52C 58EF
437 52E A1ØB58              BE    D1,51,L6
    531 F1ØB9ØØ6
    535 58E6
438 537 F1ØB9216            BE    D1,D2,L6
    53B 58EØ
439                 *
44Ø                 *BRANCH 1ST OPERAND NOT EQUAL SECOND OPERAND
441 53D E1ØB3F              BU    25,H1,L6
    54Ø 4F19ØØD9
442 544 F733510B            BU    26,T1,L6
    548 5ØD3
443 54A F734910B            BU    27,D1,L6
    54E 5ØCD
```

```
444                     *
445 550 A10B18                  BU      H1,X'25',L6
    553 F0075736
    557 50C4
446 559 F10B110B                BU      H1,H1,L6
    55D 50BE
447 55F A10B18                  BU      H1,T1,L6
    562 F007510B
    566 50B5
448 568 A10B18                  BU      H1,D1,L6
    56B F006910B
    56F 50AC
449                     *
450 571 F10B573D                BU      T1,12345,L6
    575 50A6
451 577 A10B18                  BU      T1,H1,L6
    57A F10B5007
    57E 509D
452 580 F10B5216                BU      T1,T2,L6
    584 5097
453 586 A10B58                  BU      T1,D1,L6
    589 5006910B
    58D 508E
454                     *
455 58F F10B974A                BU      D1,X'1234',L6
    593 5088
456 595 A10B18                  BU      D1,H1,L6
    598 F10B9006
    59C 507F
457 59E A10B58                  BU      D1,T1,L6
    5A1 F10B9006
    5A5 5076
458 5A7 F10B9216                BU      D1,D2,L6
    5AB 5070
459                     *
```

```
460                  *BRANCH 1ST OPERAND          2ND OPERAND
461 5AD E10B3F                BL    25,H1
    5B0 4F190469
462 5B4 F733510B              BL    26,[illegible]
    5B8 5463
463 5BA F734910B              BL    27,D1,L6
    4BE 545D
464                  *
465 5C0 A10B18                BL    H1,X'35',L6
    5C3 F0075736
    5C7 5454
466 5C9 F10B110B              BL    H1,H1,L6
    5CD 544E
467 5CF A10B18                BL    H1,T1,L6
    5D2 F007510B
    5D6 5445
468 5D8 A10B18                BL    H1,D1,L6
    5D8 F006910B
    5DF 543C
469                  *
470 5E1 F10B573D              BL    T1,12345,L6
    5E5 5436
471 5E7 A10B18                BL    T1,H1,L6
    5EA F10B5007
    5EE 542D
472 5F0 F10B5216              BL    T1,T2,L6
    5F4 5427
473 5F6 A10B58                BL    T1,D1,L6
    5F9 F006910B
    5FD 541E
474                  *
475 5FF F10B974A              BL    D1,X'1234',L6
    603 5418
476 605 A10B18                BL    D1,H1,L6
    608 F10B9006
```

```
    60C 540F
477 60E A10B58              BL    D1,T1,L6
    611 F10B9006
    615 5406
478 617 F10B9216            BL    D1,D2,L6
    61B 5400
479                 *
480                 L6      EQU   *
481                 *BRANCH 1ST OPERAND LESS OR EQUAL 2ND OPERAND
482 61D 310B3F              BLE   25,H1,L6
    620 4F190E07
483 624 F733510B            BLE   26,T1,L6
    628 5E0D
484 62A F734910B            BLE   27,D1,L6
    62E 5E13
485                 *
486 630 A10B18              BLE   H1,X'25',L6
    633 F0075736
    637 5E1C
487 639 F10B110B            BLE   H1,H1,L6
    63D 5E22
488 63F A10B18              BLE   H1,T1,L6
    642 F007510B
    646 5E2B
489 648 A10B18              BLE   H1,D1,L6
    64B F006910B
    64F 5E34
490                 *
491 651 F10B573D            BLE   T1,12345,L6
    655 5E3A
492 657 A10B18              BLE   51,H1,L6
    65A F10B5007
    65E 5E43
493 650 F10B5216            BLE   T1,T2,L6
    664 5E49
```

```
494 666 A10B58              BLE   T1,D1,L6
    669 F00691OB
    66D 5E52
495                 *
496 66F F10B974A            BLE   D1,X'1234',L6
    673 5E58
497 675 A10B18              BLE   D1,H1,L6
    678 F10B900G
    67C 5E61
498 67E A10B58              BLE   D1,T1,L6
    681 F10B9006
    685 5E6A
499 687 F10B9216            BLE   D1,D2,L6
    68B 5E70
500                 *
501                 *BRANCH 1ST OPERAND GREATER THAN 2ND OPERAND
502 68D A10B18              BH    25,H1,L6
    690 F0075748
    694 5679
503 697 F10B5733            BH    26,T1,L6
    69A 567F
504 69C F10B9734            BH    27,D1,L6
    6A0 5685
505                 *
506 6A2 E10B3F              BH    H1,X'25',L6
    6A5 4F25068C
507 6A9 F10B110B            BG    H1,H1,L6
    6AD 5692
508 6AF A10B18              BH    H1,T1,L6
    6B2 F10B5007
    6B6 569B
509 6B8 A10B18              BH    H1,D1,L6
    6BB F10B9006
    6BF 56A4
510                 *
```

```
511 6C1 F73D510B              BH    R1,12345,L6
    6C5 566AA
512 6C7 A10B18                BH    T1,H1,L6
    6CA F00751ØB
    6CE 56B3
513 6D0 F216510B              BH    T1,T2,L6
    6D4 56V9
514 6D6 A10B58                BH    R1,D1,L6
    6D9 F10B9006
    6DD 56C2
515                 *
516 6DF F74A910B              BH    D1,X'1234',L6
    6E3 56C8
517 6E5 A10B18                BH    D1,H1,L6
    6E8 F0069ØB
    6EC 56D1
518 6EE A10B58                BH    D1,T1,L6
    6F1 F0069ØB
    6F5 56DA
519 6F7 F216910B              BH    D1,D2,L6
    6FB 56E0
520                 *
521                 *BRANCH 1ST OPERAND GREATER OR EQUAL 2ND OPERAND
522 6FD A10B18                BHE   25,H1,L6
    700 F0075748
    704 5EE9
523 706 F10B5733              BHE   26,T1,L6
    70A 5EEF
524 70C F10B9734              BHE   27,D1,L6
    710 5EF5
525                 *
526 712 E10B3F                BHE   H1,X'25',L6
    715 4F250EFC
527 7L9 F10B110B              BHE   H1,H1,L6
    71D 5F02
```

```
528 71F A10B18              BHE   H1,T1,L6
    722 F10B5007
    726 5F0B
529 728 A10B18              BHE   H1,D1,L6
    72B F10B9006
    72F 5F14
530                 *
531 731 F73D510B            BHE   T1,12345,L6
    735 5F1A
532 737 A10B18              BHE   T1,H1,L6
    73A F0075108
    73E 5F23
533 740 F216510B            BHE   T1,T2,L6
    744 5F29
534 746 A10B58              BHE   T1,D1,L6
    749 F20B9006
    74D 5F32
535                 *
536 74F F74A910B            BHE   D1,X'1234',L6
    753 5F38
537 755 AQ0B18              BHE   D1,H1,L6
    758 F0069108
    75C 5F41
538 75E A10B58              BHE   D1,T1,L6
    761 F0069108
    765 5F4A
539 767 F216910B            BHE   D1,D2,L6
    76B 5F50
540                 *
541                 *DECREMENT OPERAND 1 BY OPERAND 2 AND BRANCH IF RESULT IS ZERO
542 76D F10B1216            BDZ   H1,H2,L7
    771 7872
543 773 F10B5737            BDZ   T1,5,L7
    777 786C
544 779 F10B5216            BDZ   T1,T2,L7
```

```
    77D 7866
545 77F F10B973E            BDZ   D1,10,L7
    783 7860
546 785 F10B9216            BDZ   D1,D2,L7
    789 785A
547                 *DECREMENT OPERAND 1 BY OPERAND 2 AND BRANCH RESULT NOT ZERO
548 78B F10B1216            BDNZ  H1,H2,L7
    78F 7054
549 791 F10B5737            BDNZ  T1,5,L7
    795 704E
550 797 F10B5216            BDNZ  T1,T2,L7
    79B 7048
551 79D F10B973E            BDNZ  D1,10,L7
    7A1 7042
552 7A3 F10B9216            BDNZ  D1,D2,L7
    7A7 703C
553                 *DECREMENT OPERAND 1 BY OPERAND 2 AND BRANCH RESULT NEGATIVE
554 7A9 F10B1216            BDLZ  H1,H2,L7
    7AD 7436
555 7AF F10B5738            BDLZ  T1,5,L7
    7B3 7430
556 7B5 F10B5216            BDLZ  T1,T2,L7
    7B9 742A
557 7BB F10B973E            BDLZ  D1,10,L7
    7BF 7424
558 7C1 F10B9216            BDLZ  D1,D2,L6
    7C5 741E
559                 *DECREMENT OPERAND 1 BY OPERAND 2 AND BRANCH RESULT ZERO OR NEGATIVE
560 7C7 F10B1216            BDLEZ H1,H2,L7
    7CB 7C18
561 7CD F10B5737            BDLEZ T1,5,L7
    7D1 7C12
562 7D3 F10B5216            BDLEZ T1,T2,L7
    7D7 7C0C
563 7D9 F10B973E            BDLEZ D1,10,L7
```

```
    7DD 7C06
564 7DF F10B9216            BDLEZ D1,D2,L7
    7E3 7C00
565                 *DECREMENT OPERAND 1 BY ONE AND BRANCH RESULT ZERO
566                 L7      EQU   *
567 6E5 F10B574C            BDZ   T1,L7
    7F9 7A06
568 7EB F10B9740            BDZ   D1,L7
    7EF 7A0C
569                 *DECREMENT OPERAND 1 BY ONE AND BRANCH RESULT NOT ZERO
570 7F1 F10B574C            BDNZ  T1,L7
    7F5 7212
571 7F7 F10B9740            BDNZ  D1,L7
    7FB 72A8
572                 *DECREMENT OPERAND 1 BY ONE AND BRANCH RESULT NEGATIVE
573 7FD F10B574C            BDLZ  T1,L7
    801 761E
574 803 F10B9740            BDLZ  D1,L7
    807 7624
575                 *DECREMENT OPERAND 1 BY ONE AND BRANCH IF RESULT NEGATIVE OR ZERO
576 809 F10B574C            BDLEZ T1,L7
    80D 7E2A
577 80F F10B9740            BDLEZ D1,L7
    813 7E30
578                 *
579                 *
580                 ******************************************
581                 *
582                 * CONVERSION INSTRUCTIONS
583                 *
584                 ******************************************
585                 *
586                 *MOVE BINARY TO DECIMAL
587 815 A10B18              MBD   H1,R2
    818 162F
```

```
    81A 110008
    81D 16F2
588 81F A10B58              MBD   T1,R2
    822 162F
    824 110008
    827 16F2
589 829 1A0B98              MBD   D1,R2
    82C 162F
    82E 110008
    831 16F2
590 833 F0144739            MBD   8,H1,R2
    837 A10B18
    83A 162F
    83C 111008
    83F 16F2
591 841 F0144739            MBD   8,T1,R2
    845 A10B58
    848 162F
    84A 111008
    84D 16F2
592 84F F0144739            MBD   4,D1,R2
    853 A10B98
    856 162F
    858 111008
    85B 16F2
593                 MOVE DECIMAL TO BINARY
594 85D D21641              MDB   R1,T2
595 860 D21681              MDB   R1,D2
596                 MOVE BINARY TO HEX
597 863 D10B32              MBX   H1,R2
598 866 D10B72              MBX   T1,R2
599 869 D10BB2              MBX   D1,R2
600
601 86C A73A58              MBX   2,H1,R2       MAXIMUM DIGITS CONVERTED = OPERAND 1
    86F D10B32
```

```
602 872 A74D58              MBX   4,T1,R2
    875 D10B72
603 878 A73958              MBX   8,D1,R2
    87B D10BB2
604                 *
605 87E A74258              MBXN  2,H1,R2          DIGITS CONVERTED = OPERAND 1
    881 D10B32
606 884 A73B58              MBXN  4,T1,R2
    887 D10B72
607 88A A74358              MBXN  8,D1,R2
    88D D10BB2
608                 *
609 890 D21621              MXB   R1,H2            MOVE HEX TO BINARY
610 893 D21661              MXB   R1,T2
611 896 D216A1              MXB   R1,D2
612                 *
613                 *
614                 ********************************************
615                 *
616                 * EXECUTION TRANSFER INSTRUCTIONS
617                 *
618                 ********************************************
619                 *
620                 *
621 899 1C03                B     L5               BRANCH LOCAL
622 89B 0C1234              B     A1
623                 *
624 89E 1A02        L5      BSL   L5               BRANCH AND STACK LOCATION LOCAL
625 8A0 111002              BSL   M1               BRANCH AND STACK LOCATION EXTERNAL
626 8A3 0D1234              BSL   A1               BRANCH AND STACK LOCATION ABSOLUTE
627                 *
628 8A6 13                  BSLI  *                BRANCH AND STACK LOCATION INDIRECT THROUGH ACCUMULAT
629                 *
630 8A7 14                  RTN   *                RETURN
631 8A8 15                  TEXT  X'15'            RETURN WITHOUT TRACE
```

```
632                     *
633 8A9 101002          ENT   M1          BRANCH EXTERNAL
634                     *
635 8AC 12              ENTI  *           BRANCH EXTERNAL INDIRECT THROUGH ACCUMULATOR
636                     *
637 8AD A10B18          BSL*  H1          BRANCH AND STACK LOCATION THROUGH HALF TALLY
    8B0 13
638 8B1 A10B58          BSL*  T1          BRANCH AND STACK LOCATION INDIRECT THROUGH TALLY
    8B4 13
639 8B5 A10B98          BSL*  D1          BRANCH AND STACK LOCATION INDIRECT THROUGH DOUBLE TA
    8B8 13
640 8B9 A10B18          ENT*  H1          BRANCH EXTERNAL INDIRECT THROUGH HALF TALLY/TALLY/D
    8BC 12
641 8BD A10B58          ENT*  T1
    8C0 12
642 8C1 A10B98          ENT*  D1
    8C4 12
643                     *
644                     *
645                     ***********************************************
646                     *
647                     * I/O AND CONTROL INSTRUCTIONS
648                     *
649                     ***********************************************
650                     *
651                     *
652 8C5 426778          IOI   R2,3,7      CALL MONITOR TO INPUT A BYTE
653 8C8 413578          IOO   R1,1,5      CALL MONITOR TO OUTPUT A BYTE
654                     *
655 8CB 6115            READ  R1          READ ONE BYTE FROM TERMINAL BUFFER
656 8CD 622D            WRITE R2          WRITE ONE BYTE TO TERMINAL BUFFER
657                     *
658 8CF 410274          MCAL  R1,2,4      MONITOR CALL
659                     *
660 8D2 400079          RTM   *           RELEASE QUANTUM
```

```
661                     *
662 8D5 411290                  IB      R1,X'12'        I/O INSTRUCTIONS
663 8D8 413480                  OB      R1,X'34'
664                     *
665 8DB 00                      NOP     *               NO OPERATION
666                     *
667 8DC 08                      HALT    *               HALT
668 8DD 4100A0                  HLD     R1              HALT AND DISPLAY
669                     *
670 830 01                      TEXT    X'01'           TEST INTERRUPTS
671                     *
672 8E1 4100A3                  ECS     R1              ENTER CONSOLE COMMAND SWITCHES
673 8E4 4100A1                  ESS     R1              ENTER SENSE SWITCHES
674                     *
675 8E7 0B                      SVP     *               START VIRTUAL PROCESS
676 8E8 0A                      RVP     *               RETURN TO VIRTUAL PROCESS
677                     *
678                     *
679                     ************************************************************
680                     *
681                     * ASSEMBLER DIRECTIVES
682                     *
683                     ************************************************************
684                     *
685 8E9 7FF00064                FRAME 100
686                     *
687                     LABEL1  EQU     *               EQUATES
688                     LABEL2  EQU     C1
689                     LABEL3  EQU     H1
690                     LABEL4  EQU     T1
691                     LABEL5  EQU     D1
692                     LABEL6  EQU     S1
693                     LABEL7  EQU     R1
694                     LABEL8  EQU     LABEL1
695                     LABEL9  EQU     X'600'
```

```
696                     *
697                     LABEL10   ORG    LØØ              ORG'S
698 ØØ1 534F4D45                  TEXT   C'SOME CODE'
    ØØ5 2Ø434F44
    ØØ9 45
699                               ORG    1
7ØØ                     *
7Ø1                     ABS       DEFA   X'Ø8ØD'          DEFINE ABSOLUTE ADDRESS
7Ø2                     *
7Ø3                               SETAR  7                SET PROGRAM ADDRESS REGISTER
7Ø4 ØØ1 A74358                    LOAD   1234
7Ø5                     *
7Ø6                     SPEC1     DEFB   R1,*1+7          SPECIAL DEF(K) FORMS
7Ø7                     SPEC2     DEFB   1,*1+11
7Ø8 ØØ4 712B                      MOV    SPEC1,SPEC2
    ØØ6 91270802
    ØØA 812B
7Ø9                     *
71Ø                     SPEC3     DEFC   R1,*+3
711                     SPEC4     DEFC   1,*+11
712 ØØC F117Ø1ØF                  MCC    SPEC3,SPEC4
713                     *
714                     SPEC5     DEFH   41,*+3
715                     SPEC6     DEFH   1,*+11
716 Ø1Ø F11BØ113                  MOV    SPEC5,SPEC6
717                     *
718                     SPEC7     DEFT   R1,*16+3
719                     SPEC8     DEFT   1,*16+11
72Ø Ø14 F115410D                   MOV    SPEC7,SPEC8
721                     *
722                     SPEC9     DEFD   R1,*16+3
723                     SPEC1Ø    DEFD   1,*16+11
724 Ø18 F117810F                  MOV    SPEC9,SPEC1Ø
725                     *
726                     SPEC11    DEFS   R1,*16+3
```

```
727                     SPEC12    DEFS  1,*16+11
728 01C F119C111                  MOV   SPEC11,SPEC12
729                     *
730                     * PLEASE SEE ABOVE FOR DEF(K)
731                     *
732 020 41              LABEL12   CHR   C'A'
733 021 19                        HTLY  25
734 022 0200                      TLY   X'200'
735 024 0001E240                  DTLY  123456
736 028 00000000                  SR    0
    02C 0000
737                     *
738                     REG1      AR    R1                DEFINE TYPE ADDRESS REGISTER
739                     REG2      AR    2
740                     REG3      AR    HS
741                     *
742                     LABEL13   DEFM  0,77              DEFINE MODAL ENTRY
743                     LABEL14   DEFM  1,LABEL13
744 02E 11004D                    BSL   LABEL13
745 031 11104D                    BSL   LABEL14
746                     *
747 034 54484953                  TEXT  C'THIS IS A TEXT MESSAGE',C'THIS IS SOME MORE',X'FF'
    038 20495320
    03C 41205445
    040 5854204D
    044 45535341
    048 4745
    04A 54484953
    04E 20495320
    052 534F4D46
    056 204D4F52
    05A 45
    05B FF
748                               CMNT  *                 THIS ALLOWS COMMENTS TO BE STARTED IN THE COMMENTS
749 05C 0006            *
```

```
    05E 0050
    050 0020
    062 0017
    064 0014
    066 001A
    068 0000001B
    06C 0025
    06E 0005
    070 0005
    072 0008
    074 0002
    076 0084
    078 0360
    07A 3039
    07C 00000005
    080 00000001
    084 0082
    086 0088
    088 00000020
    08C 1093
    08E 0283
    090 0019
    092 03F8
    094 00001234
    098 0001
    09A 0004
    09C 04D2
EOF:
```

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 Master Dictionary (M/DICT) of the data-base for the following file definitions:

PSYM - Permanent symbol table.

TSYM - Temporary symbol table.

OSYM - Operation-code symbol table.

The assembly will abort if any of these file-definitions are missing, with a message indicating the one that was not found. The temporary symbol table is inititalized before the assembly starts. Since the TSYM is actually a permanently defined file on a user's account (M/DICT) it must be pre-defined and can be examined at the conclusion of the assembly. Furthermore, only one person may be using the REAL assembler per account.

3.8.1 TSYM/PSYM TABLE ENTRY FORMATS

The item format of the entries in the PSYM & TSYM files is as follows: (Entries are in character form):

Item - id: Symbol-name

Line 1 : Symbol-code (single character - see below).

Line 2 : Symbol-value (hexadecimal location or displacement)

Line 3 : Base-register value (single hexadecimal digit)

Symbol-Codes

The symbol code is a single character code that defines the type of the symbol, it is used in the operation code lookup to determine legal operands, and to flag undefined or multi-defined labels, etc.

Code	Description - Symbol Type	Unit of Displacement
B	Bit	Bits
C	Character Register	Bytes
D	Double-Word (4-byte)	Words
H	Half-Word (2-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 op-code), or (2) the REAL mnemonic suffixed by the symbol type-codes of all the operand field entries. The purpose of the suffixing is (1) to provide for the separate handling of REAL mnemonics with variable operand field entries; (2) to provide for a check on the number and type of operand field entries. As an example, the basic REAL mnemonic for "move register to register" is MOV, but it has four different object code expansions, depending on whether the registers involved are address (R), or storage-type (S). To allow for all cases, there are four entries in the OSYM file: MOVRR, MOVRS, MOVSR and MOVSS. The assembler will attempt to look up the basic op-code first, and, if it is not found, a second attempt will be made with the basic op-code suffixed as described above.

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 op-code. If no entry is found in the TSYM & PSYM files, the assembler then sets up an entry in the TSYM file with type "U" (undefined), and location zero. This has an important ramification with regard to literal generation.

3.8.4 OSYM TABLE ENTRY FORMAT

Line one of the OSYM table entry may be one of the following:

M - Defines a macro; all further lines are macro substitution lines.

P - Defines "primitive" which calls one of the lower-level assem-ler functions.

Q - Equates this entry to another OSYM table entry specified in the next line.

3.8.5 MACRO DEFINITION FORMAT

Each substitution line in a macro definition will generate a new source statement, to which parameters may be passed from the original source statement. This newly generated source statement will, in turn, be assembled as any other source statement. Thus a macro may cause the original statement to expand into an unlimited set of new statements; however, if any generated statement calls another macro, control is passed immediately to the new macro, and the previous one cannot regain control.

Data in a macro substitution line is transferred, as it is, to generate the new source statement, except for the substitution codes, which cause a specific parameter to be substituted. Substitution codes are enclosed by parentheses:

CODE	ACTION
AF Substitution: (n)	(n decimal) causes insertion of the n-th Argument-field entry of the original source statement; if such an entry is non-existene, no substitution takes place.
Label Substitution: (L ± n)	(The ±n is optional). Causes insertion of label internal to the macro; the label is created using the macro label counter (MLC), which is initialized by the assembler at the start of an assembly. If the label substitution is in the label-field of the generated source statement, it is replaced with a label of the form "=Lm" where m=MLC + 1, and the MLC is incremented by one. If the label substitution code is in the operand field, it is replaced with a symbol of the form "=Lm" where m=MLC ± n, the MLC being unaltered. (See Section 3.13.)

3.8.6 "PRIMITIVE" DEFINITION FORMATS

Each line in a primitive is an assembler-directive that calls a specific assembler process. The first character in each line is a code defining the process:

CODE	PROCESS
A	Align location counter on word boundary.
E	Exit to explicitly defined process.
G	Generate object-code (GEN)
R	Reset entry in TSYM file (RESET)

Exit Format

E:mode-ID

Where "mode-ID" is the hexadecimal mode-ID of the processor which is to be entered.

Gen Format

G, a_1, a_2.... b_1, b_2,.... (A & B-fields separated by one blank)

A-field B-field

The G-primitive causes the actual generation of object code. There should be a one-to-one correspondence between entries in the A- & B- fields. Each A-field entry is a decimal number, and specifies the number of bits of code to be generated using the corresponding B-field entry. The sum of the A-field values must be a multiple of 8, and must be less than or equal to 32.

ENTRY	VALUE RETURNED
*	Current location counter value, in bytes.
*n	As above, modulo "n" bits (n decimal)
B	Current base register value.
n	Decimal literal.
X'h'	Hexadecimal literal.

ENTRY	VALUE RETURNED
C'k'	Character literal (one character only).
An;m	Value returned is: (1) AFn if AFn is a literal; or (2) From line "m" of AFn if AFn is a symbol. (Zero is symbol undefined).
Jn;k	Used to compute relative displacement from current location (*), to the location defined by AFn. Value returned is signed magnitude, 10-bits: (Value (AFn)-*)=k +k if backward reference -k if forward reference

Reset Format

R,n $b_1,b_2,....b_m$

Resets values in TSYM entry for AFn. B-field entries refer to lines 1 through m, and entries are as above. This is used to re-define TSYM table entries: when the assembler finds a label-definition, it inserts the entry in the TSYM as type 'L', location as current location in bytes, and base register field from the current base register. To redefine the entry, a R-primitive is used. For example, the opcode 'TLY' is used to define a local tally, as in:

```
LABEL     TLY     123
```

The OSYM entry for 'TLY' is:

```
Line 1  (type)   :  P

Line 2           :  R,0 C'T',*16,B     Redefines type as T;
                                       location module 16.

Line 3           :  G,16 A2;2          Generates 2-byte object-code
                                       value
```

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) loc. obj. code (VM)... (AM).

Note that regardless of what the assembler appends to the original source statement, the delimiters surrounding the entire statement remain unchanged; this ensures proper source statement input on subsequent assemblies.

3.10 LITERAL GENERATION

REAL statements that require assembly of literal should setup entries in the TSYM of the following format:

Item-ID	:	=k value	k=	S storage register
Symbol-type	:	U		D double word
Location	:	0		T word

This will be done simply by the reference of the symbol as an operand in a source statement. However, the OSYM table entry that the source statement references must be of the "suffixed" type rather than being the basic op-code. For example, the statement:

MOV =T23,CTRI

which references the OSYM table entry 'MOVTT', and, in so doing, sets up the TSYM entry '=T23' as type U to be assembled as a literal.

At the end of pass I, the assembler searches the TSYM file for undefined entries; if they are of the format shown above, a dummy source statement of the form:

LABEL	OPCODE	OPERAND
=k value	:k	value

is generated and assembled. Thus the entries ":S", ":D", ":T" in the OSYM are reserved and cause generation of 6-byte (storage register, type S), 4-byte (double-word, type D) and 2-byte (word, type T) literals, respectively.

3.11 REASSEMBLY IN PASS II

During the assembly process, statements which have a forward reference are flagged for re-assembly by prefixing the character "X" to the location counter and object code data that are appended to the source statement. The REAL assembler is not a true two-pass assembler; pass II consists of scanning the mode for statements that have been flagged for re-assembly, and re-assembling those statements exclusively. If they contain references to undefined symbols, the object code output will still have the "reassemble" flag stored with it, after pass II.

3.12 ASSEMBLER ERROR MESSAGES

Message	Explanation
*UNDEF: $Symbol_1$ $Symbol_2$....	Undefined symbols at end of pass I (Message at end-of-mode).
*LABEL TYPE	Label-field format error.
*MULTIDEF	Label-field entry was previously defined.
*REF-UNDEF	Reference to undefined symbol.
*LABEL ?	Required label-field missing.
*OPCODE ?	Op-code-field entry missing.
*OPERAND ?	Required operand-field entry missing.
*OPCD ILLGL:opcode	Either the opcode was illegal, or the operand types were illegal for the opcode.

*OPRND TYPE	The operand-field entry was an illegal type; eg: ORG statement with undefined symbol, SETAR with non-numeric operand, etc.
*RANGE ERR	The range of the operand-field entry is illegal; eg: SETAR with n not $0 \leqslant n < 16$.
*TRUNCATION	Object code truncation may be due to: branch out-of-range; TSYM/PSYM table entry error; specification error in the GEN primitive.

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

*A-FIELD ? *B-FIELD ?	Error in A- or B-field specification.
*OPCODE-TYPE ERR	Opcode type not a P/Q/M, or primitive type was illegal.
*MACRO-SPEC ERR	Error in the macro specification

3.13 EXAMPLE OF REAL MACRO EXPANSION

```
Location Counter  =  X'012'
        MLC       =  0

Source Statement      :     BCN        R4,LABX

(Branch if character
addressed by R4 is
numeric)
```

		Symbol	Type	Displacement
PSYM table entry	:	R4	R	0004
TSYM table entry	:	LABX	L	0024
OSYM table lookup	:	BCNRL		
Line 1 (type)	:	M		
Line 2	:	BCL (2),X'30',(L + 1)		If 0, skip next
Line 3	:	BCLE (2),X'39',(3)		Branch is < 9
Line 4	:	(L) EQU *		Define Internal label

Generated Source Statements

From Line 1 : BCL R4,X'30',=L01

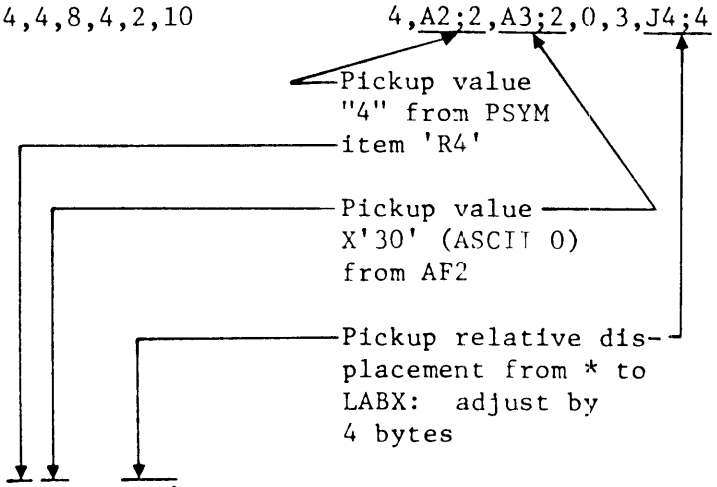
From Line 2 : BCLE R4,X'39',LABX

From Line 3 : =L01 EQU *

OSYM table lookup : BCLERNL

Line 1 (type) : P

Line 2 : G,4,4,8,4,2,10 4,A2;2,A3;2,0,3,J4;4



Object Code : X'4430 0 C0E'

3.14 CORRECT USE OF REGISTER TO STORAGE REGISTER COMPARE OPERATION

The BE and BU instructions with R and S type operands (see Section 3.7.5) operate by multiplying the FID's of the registers by 500, adding the displacements, and then comparing the low order n bits of the result. This allows a register and storage register to compare equal in certain cases when they are in fact not equal. This will happen when the registers are in the linked format pointing to frames whose FID's are a multiple of 128 apart (e.g., 12340 and 12468) with the storage register containing the lower FID, and the relative displacements of the registers are equal within their respective frames.

To correctly perform a register to storage register compare operation, one of the three forms discussed below must be used. Forms 2 and 3 use the BRE and BRU macros; these macros expand into a comparison of the double tallies which are the FID's of the registers, followed by the comparison of the registers in the normal manner.

FORM 1

Form 1 uses the BE and BU instructions. Form 1 may be used when one of the following occur:

- The R and SR are in linked format and are known to point within the same frame. (This is a rare circumstance.)
- The R and SR are in unlinked format. (For example, IB and IBEND.)
- The R and SR are in linked format and are known to point within the same contiguously linked block of frames, and point to frames which are less than 32 frames apart.
- The R and SR are in linked format, the displacement of each is less than 500, and the difference between the registers is less than 500 bytes.

The following illustrates an example of this form:

```
            L0 EQU *
001 C0E7AA04  BE  IB,IBEND,L0
005 C0E7AA08  BE  IB,IBEND,L0
009 C0E7A20C  BU  IB,IBEND,L0
00D C0E7A210  BU  IBEND,IB,L0
```

FORM 2

Form 2 uses the BRE and BRU macros. This form must be used when the R and SR are in linked format, the displacement of each is less than 500, and nothing else is known about the registers (typically in the case of IR and SR4). The following illustrates an example of this form:

```
            L0 EQU *
011 F271909A  BRE IR,SR4,L0
015 5004
017 C2706A1A

01B F271909A  BRU IR,SR4,L0
01F 5220
021 C2706224
```

FORM 3

The FAR instruction must be used in conjunction with BRE and BRU macros under the following circumstances: When the R and SR are in linked format, the displacement of the SR is less than 500, and nothing else is known about the registers. The following illustrates the use of this form:

```
            L0 EQU *
025 440060    FAR IS,0            FORCES DISPLACEMENT OF REGISTER
028 F0669092  BRE IS,ISEND,L0     TO BE LESS THAN 500
02C 5004
02E C0654A31

032 440060    FAR IS,0            FORCES DISPLACEMENT OF REGISTER
035 F0669092  BRU  IS,ISEND,L0    TO BE LESS THAN 500
038 C065423E
```

NOTE

None of the forms will work when either of the following is true:

- Both registers are not in the same format (linked or unlinked).
- The displacement of the SR is greater than 499 (unless the R and SR are in linked format and are known to point within the same contiguously linked block of frames and point to frames which are less than 32 frames apart). If this occurs, the SR must be moved to a scratch register; the displacement of the register must be forced to be less than 500 (i.e., by a FAR instruction); and the register must be moved back to the storage register for the compare.

3.15 REAL INSTRUCTION SUMMARY

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 check-out 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.
f.d	FID in decimal; displacement in hexadecimal.
.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 $2 \leq r \leq 15$).
R.r	Specifies Address Register "r" (where "r" is a hexadecimal value in the range $2 \leq r \leq F$).

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
B address	This command adds the "address" to the Break Table.
D	This command displays the Break Table and Trace Table.
E n	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" turns off the Execution Counter.
END	This command terminates execution and returns to TCL.
G	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".

General Form	Description
K address	This command "kills" the break-point (i.e., deletes "address" from Break Table).
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 "Modal-Break" on; the message "OFF" is displayed when "Modal-Break" is switched off.
N n	This command sets the Break-Point Counter to "n" (i.e., inhibits traps until "n" breaks have occurred).
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.
T address	This command adds the "address" to the Trace Table.
T indirect-address	This command adds the "indirect-address" to the Indirect Trace Table.
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.

Examples of these commands are provided in Section 4.5.

4.3 THE DEBUG DATA DISPLAY COMMANDS

4.3.1 WINDOWS

Before descriging 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.
;.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. (If "window" is not 1, 2 or 4, only one byte of data will be displayed.)

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 this case, the window must have been 1, 2 or 4.

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-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.

4.4 BREAK MESSAGES

DEGUG 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.

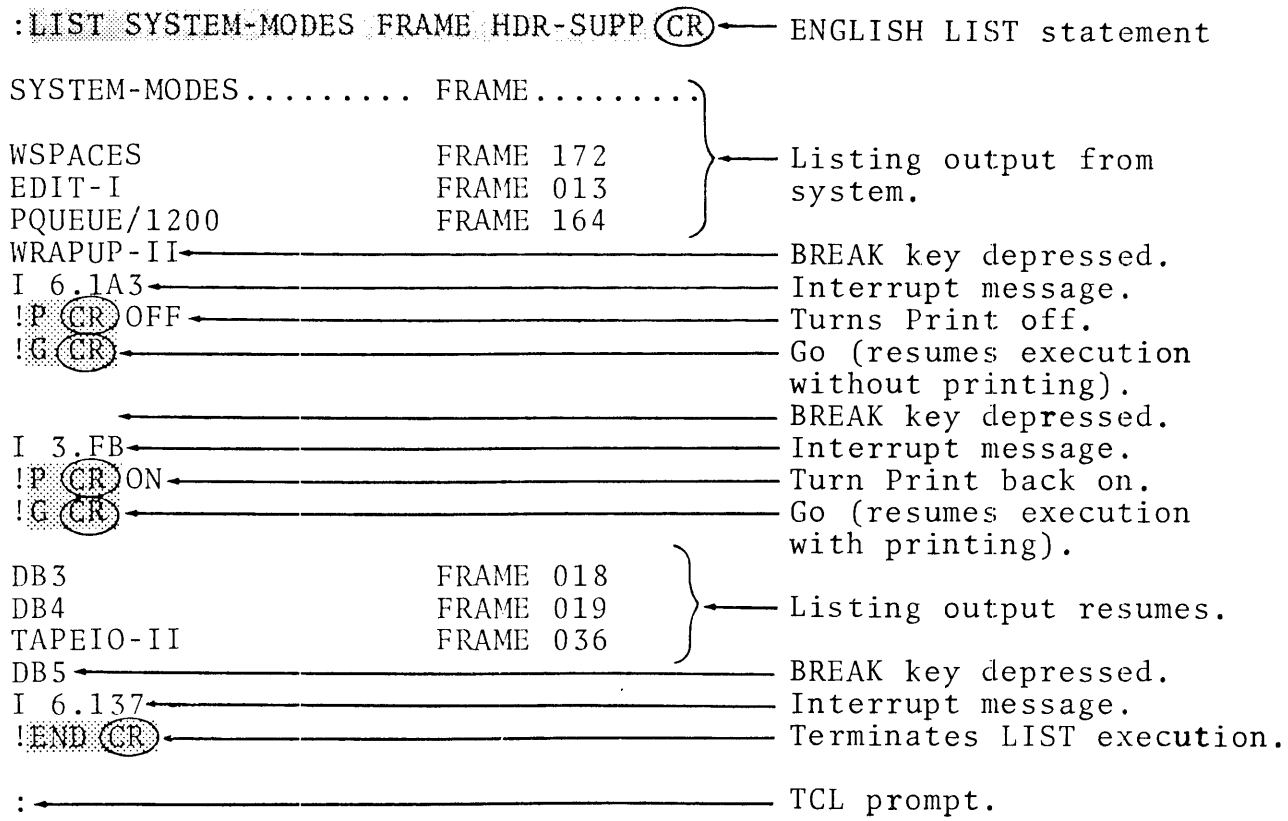
where "f" is the decimal FID and "d" the hexadecimal displacement, representing the location of the execution interruption point.

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.

4.5 EXAMPLES

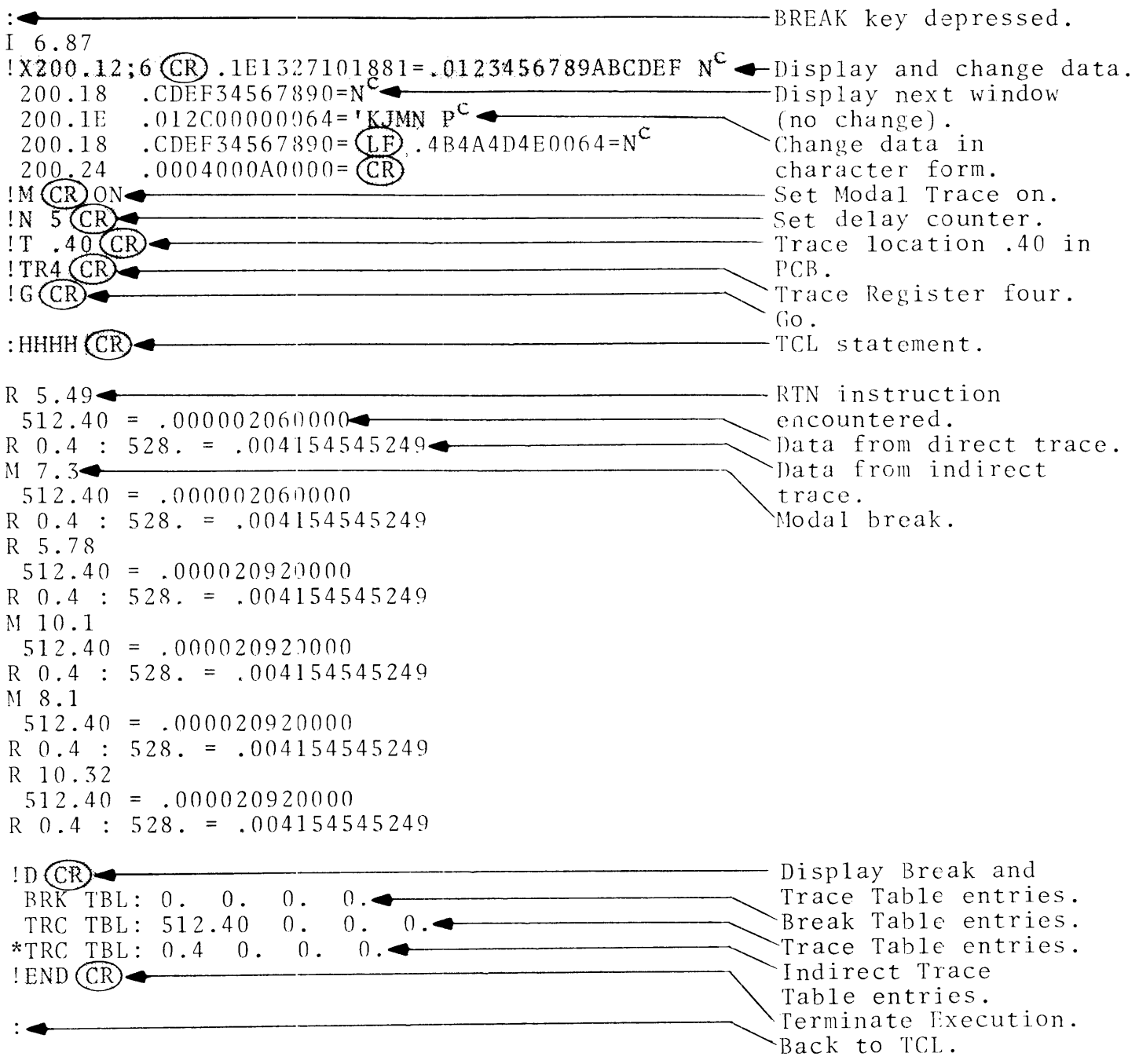
4.5.1 SIMPLE EXAMPLE

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



4.5.2 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.



4.6 HARDWARE TRAP CONDITIONS

Certain error conditions cause the CPU to execute a trap to the DEBUG state; processing of the current program will be aborted, and a message indicating the nature of the trap (and the location at which it occurred) will be displayed. The following list shows these error conditions.

Error No.	Message	Description
0	ILLGL OPCODE	An illegal (undefined) operation code has been found.
1	RTN STK EMPTY	An RTN (return) instruction was executed when the return-stack was empty (current pointer was at X'0184').
2	RTN STK FULL	A BSL or BSLI (subroutine call) instruction was executed when the return-stack was full (current pointer was at X'01B0'); the return-stack has been reset to an "empty" condition before the trap.
3*	FRM-ID ZERO	An address register has an FID of zero.
4*	CROSSING FRM LIMIT	An address register in the "unlinked" format has been incremented or decremented off the boundary of a frame, or has been used in a relative address computation that causes the generated relative address to cross a frame boundary.
5*	FORW LNK ZERO	An address register in the "linked" format has been incremented past the last frame in the linked frame set.
6*	BACKW LNK ZERO	An address register in the "linked" format has been decremented prior to the first frame in the linked frame set.

Error No.	Message	Description
7	PRIV OPCODE	A Privileged operation code (one executable only in the Monitor mode of operation), has been found while executing in the Virtual mode.
8	ILLGL. FRAME-ID	An address register has an FID that exceeds the maximum value allowable in the current disc configuration.
11	STK FRMT ERR	The Return-stack pointers are in an illegal format. Either the ending address is less than X'0184', or the current address is less than X'0184'. The pointers have been reset to an initial condition of X'01B0' and X'0184', respectively.
12	REGISTER ZERO DETACHED	Register zero has been detached by a user-program.

In the case of the traps marked with an asterisk (*) in the list above, the following message will also be returned:

```
REG = 0.x
```

where "x" is the hexadecimal Address Register number of the register causing the trap condition.

In all cases, the following message will also be returned:

```
ABORT @ f.d
```

where "f" is the decimal FID of the frame and "d" is the hexadecimal displacement within it (of the location where the trap occurred). This corresponds to the location counter in the assembly listing of the corresponding program.

Note that the G command, without an address specification, cannot be used after a trap to the DEBUG state.

SECTION 5

SYSTEM SOFTWARE

5.1 INTRODUCTION

Assembly level programming in the Reality system is facilitated by a set of system subroutines that allow easy interaction with the disc file structure, terminal I/O, and other routines. These subroutines work with a standard set of addressing registers, storage registers, tallys, character registers, bits, and buffer pointers, collectively called functional elements. In order to use any of these routines, therefore, it is essential that the calling routine set up the appropriate functional elements as required by the called routine's Input Interface.

The standard set of functional elements is pre-defined in the permanent symbol file (PSYM), and is therefore always available to the programmer. Included in the PSYM are all the mode-id's (program entry points) for the standard system subroutines. There is no reason that a symbol internal to an assembly program cannot have the same name as a PSYM-file symbol, if the PSYM-file symbol is not also referenced in that program; such symbolic usage cannot be a "forward" reference in the assembly program. To avoid confusion, however, it is best to treat the entire set of PSYM symbols as reserved symbols.

5.1.1 ADDRESS REGISTERS

All data reference in the system is made indirectly through one of the sixteen address registers (A/R). Registers zero and one have special, firmware-defined meaning; the other fourteen may be considered general-purpose registers, with the limitation that system software conventions determine the usage of most A/R's. Registers zero and one should never be changed in any way by assembly programs. Register two always points to the SCB after the debugger has been entered.

Register zero always addresses byte zero of the process' PCB; register one always addresses byte zero of the frame in which the process is currently executing. Thus all elements in the PCB may be relatively addressed using register zero as a base register; this includes the individual segments of the address registers themselves (e.g., R15WA, referencing the word-address segment of R15). Address registers can thus be setup explicitly by setting up their segments appropriately; the more conventional way of setting up an A/R is to move a S/R into it. For example, the sequences below are functionally identical.

```
FRM100    ADDR    0,X'100'          DEFINE A LITERAL S/R
          .                         REFERENCING FRAME X'100'
          .
          .
          MOV     FRM100,R15
```

and

```
          .
          .
          ZERO    R15WA
          ZERO    R15DSP
          MOV     =DX'80000100' R15FID
```

It is important to note that, in the first sequence, the address register is automatically set to the "detached" format when the "MOV" instruction executes; in the second sequence, the address register is explicitly set to the "detached" format by the "ZERO R15WA" instruction. The word=address of an A/R must be zeroed before other segments of the A/R are modified.

5.1.2 ATTACHMENT AND DETACHMENT OF ADDRESS REGISTERS

All instructions that reference data force "attachment" of the A/R(s) used in the reference. Not all instructions do the same; for example, the "increment A/R by tally" instruction will not cause a "detached" A/R to attach before execution.

This point may lead to programming errors; consider the following sequence:

```
          .
          .
          .
L1   BCU   AM,R6,NXT     R6 "ATTACHED" AT THIS POINT"
L2   INC   R6, SIZE      R6 MAY "DETACH" DUE TO THIS INSTRUCTION
L3   MOV   R6, SR4       SAVE R6
          .
          .
          .
```

The instruction at L2 may force R6 to "detach" (if the contents of SIZE are such that the resultant address is beyond the limits of the current frame); storing R6 in SR4 will then cause SR4 to have a large positive displacement, and a FID equal to that in R6 at the time of execution of the instruction at L1. Subsequently, a register comparison instruction of the form:

```
          BE      R15,SR4,L20
```

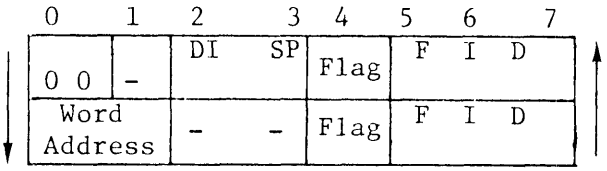
may execute incorrectly due to the fact that if the FID's of R15 and SR4 are unequal at the time of execution, it is assumed that the two frames are contiguously linked (See Section 3.14). Therefore, it is best to force "attachment" of R6 before L3; a convenient way of doing so is to execute the instruction:

```
L3A    FAR    R6,0
```

though any data reference instruction would serve as well.

The following table summarizes the attachment/detachment process:

ATTACHMENT & DETACHMENT OF ADDRESS REGISTERS



A/R is Attached when:

(1) Any instruction that references data via the A/R is executed.

(2) Execution of
INC r
DEC r
instructions.

(3) Execution of
FAR r,n
instruction.

A/R is Detached when:

(1) Process is deactivated due to: terminal I/O; disk I/O; peripheral I/O; timer run-out; monitor call.

(2) A S/R is moved to the A/R.

(3) Execution of
INC r,t
DEC r,t

if a frame boundary is crossed.

5.1.3 RE-ENTRANCY

In practically all cases, the system software is re-entrant, that is, the same copy of the object code may be used simultaneously by more than one process. For this reason, no storage internal to the program

is utilized; instead the storage space directly associated with a process is used; this is part of the process' Primary, Secondary, Debug (or Tertiary) and Quadrenary Control blocks. The Primary Control Block (PCB) is addressed via address register zero; the SCB via register two. The Debug Control Block is used solely by the Debug Processor as a scratch area, and should not be used by any other programs. The Quadrenary Control Block has no register addressing it; it is used by some system software (magnetic tape routines, for example) which temporarily setup a register pointing to it; its use is reserved for future software extensions.

A user program may utilize storage internal to the program if it is to be used in a non-re-entrant fashion; however in most cases it will be found that the functional elements defined in the PSYM will be sufficient.

In some cases it may be required to setup a program to be executable by only one process at a time; that is, the code is "locked" while a process is using it, and any other process attempting to execute the same code waits for the first process to "unlock" it. The following sequence is typical:

```
        ORG     0
        TEXT    X'01'                   INITIAL CONDITION FOR LOCK BYTE
        CMNT                            (NOTE USAGE OF STORAGE INTERNAL TO
        .                               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
```

5.1.4 WORK-SPACES OR BUFFERS

There is a set of work-spaces, or buffer areas, that is pre-defined and available to each process. If the system conventions with regard to these buffers are maintained, they should prove adequate for the majority of assembly programming. There are three "linked" buffers, or work-spaces, of equal size, symbolically called the IS, the OS, and the HS. These are at least 3000 bytes in length each; more space for each area can be assigned to a process at LOGON time. There are five other work-spaces, the BMS, CS, AF, IB and the OB which may vary between 50 and 140 bytes in length, and are all in one frame. There is the TS, a one-frame unlinked work-space of 512 bytes, and the PROC work-space, 2000 bytes in length which is used normally by the PROC processor alone;

finally there are four additional frames (PCB+28 through PCB+31) that are unused by the system, and are freely available. PCB+28 is used internally by the RPG processor.

Each work-space is defined by a beginning pointer and an ending pointer, both of which are storage registers (S/R's). When the process is at the TCL level, all these pointers have been set to an initial condition. At other levels of processing, the beginning pointers should normally be maintained; the ending pointers may be moved by system or user routines. The address registers (A/R's) that are named after these work-spaces (IS,OS,AF,etc.) need not necessarily be maintained within their associated work-spaces; however, specific system routines may reset the A/R to its associated work-space. The table below discusses these points for each work space. Note that, conventionally, a buffer beginning pointer addresses one byte before the actual location where the data starts. This is because data is usually moved into a buffer using one of the "move incrementing" type of instructions, which increment the A/R before the data movement.

Work-space	Location (offset from PCB)	Size	Linked?	Remarks
BMS	4 (disp.=0)	50	No	Normally contains an item-id when communicating with the disc file i/o routines. Typically, the item-id is copied to the BMS area, starting at BMSBEG+1. BMSBEG may be moved to point within any scratch area. BMSEND normally points to the last byte of the item-id. BMS (A/R) is freely usable except when explicitly or implicitly calling a disc file i/o routine.
AF	4 (disp.=50)	50	No	This work-space is not used by any system subroutine, though the AF A/R is used as a scratch register.
CS	4 (disp.=100)	100	No	As above.
IB	4 (disp.=200)	≤140	No	Is used by the terminal input routines to read data. IBBEG may be moved to point within any scratch area before use. IBEND conventionally points to the logical end of data. IB A/R is freely usable except when explicitly or implicitly calling a terminal input routine.

Work-space	Location (offset from PCB)	Size	Linked?	Remarks
OB	4 (disp.=201 + IBSIZE)	140	No	Is used by the terminal output routines to write data. OBBEG & OBEND should not be altered; they always point to the beginning and end of the OB area. OB A/R conventionally points one before the next available location in the OB buffer.
TS	5	511	No	This work-space is used by the RPG compiler.
PROC	6-9	2000	Yes	Used exclusively by the PROC Processor for working storage. User-exits from Proc's may change pointers in this area.
HS	10-15	3000+	Yes	Used as a means of passing messages to the WRAPUP processor at the conclusion of a TCL statement. May be used as a scratch area if there is no conflict with the WRAPUP history-string formats. HSBEG should not be altered; HSEND conventionally points one byte before the next available location in the buffer (initial condition is HSBEG=HSEND).

Work-space	Location (offset from PCB)	Size	Linked?	Remarks
IS OS	16-21 22-27	3000+	Yes	These work-spaces are used interchangeably by some system routines since they are of the same size (and are equal in size to the HS). Specific usage is noted under the various system routines. ISBEG and OSBEG should not be altered, but may be inter-changed if necessary. Initial condition is that ISEND and OSEND point 3000 bytes past ISBEG & OSBEG respectively (not at the true end if additional work-space is assigned at LOGON time). IS & OS A/R's are freely usable except when calling system sub-routines that use them.
	28-31			Used for compilation and execution of RPG programs, but are otherwise available.

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

5.1.6 USAGE OF XMODE

In several cases, the multiple-byte move instructions can be used (say, when building a table) even when it is not known whether there is enough room in the current linked frame set to hold the data. Normally, if the end of a linked frame set is reached, DEBUG is entered with a "forward link zero" abort condition. However, the tally XMODE may be setup to contain a mode-id of a user-written subroutine that will gain control under such a condition. This subroutine can then process the end-of-frame condition, and, by executing a 'RTN' instruction, normal processing will continue. Instructions then can be handled by this scheme are: INC register; MCI; MIC; MII; MIID; SCD; MIIR. Care should be taken in the case of MIIR to save register R15 in the subroutine. MIIT cannot be handled because DEBUG destroys the accumulator in the process of transferring control via XMODE.

For example:

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

!XXX      EQU     %                 ENTRY POINT FOR SUBROUTINE
          MOV     R15, SR1          SAVE R15
          SRA     R15, ACF          SET TO SAVE REGISTER NUMBER
          BCE     X'0D',R15,OK      ENSURE TRAP WAS DUE TO R13
          MOV     0,XMODE           PREVENT DEBUG RE-ENTRY
          ENT     5,DB1             NO! : REENTER DEBUG TO PRINT
          CMNT                      "FORWARD LINK ZERO" MESSAGE

%
OK        MOV     500,R13DSP        RESET DISPLACEMENT FIELD OF R13, SINCE
          CMNT                      FIRMWARE HAS LEFT IT IN A STRANGE STATE.

%         HANDLE END-OF-FRAME CONDITION HERE

          MOV     R13FID,RECORD     SETUP INTERFACE FOR ATTOVF
          BSL     ATTOVF            GET ANOTHER FRAME FROM OVERFLOW

          MOV     SR1,R15           RESTORE R15
          RTN                       RETURN TO CONTINUE EXECUTION OF
                                    MII INSTRUCTION.
```

5.1.7 INITIAL CONDITIONS

At any level in the system, the following elements are assumed to be setup; they should not be altered by any programs:

MBASE	D	Contain base-FID, modulo and separation of the M/DICT associated with the process.
MMOD	T	
MSEP	T	
USER	T	Contains the low-order 16 bits of the base-FID of the M/DICT.

5.1.8 SPECIAL PSYM ELEMENTS

Certain elements have a "global" significance to the system; in addition to those described above they are:

Functional Element	Description
Arithmetic condition flags:	These are altered by any arithmetic instruction, as well as the branch instructions that compare two relatively addressed fields.
ZROBIT	Set if result of operation is zero (equal).
NEGBIT	Set if result of operation is negative.
OVFBIT	Set if arithmetic overflow resulted.
H0 through H7	Overlays accumulator and extension; H7 is high-order byte of D1; H0 is low-order byte of D0.
INHIBIT	If set, the "BREAK" key on the terminal is inhibited; used by processes that should not be interrupted.
OVRFLCTR	See WRAPUP for usage.
RSCWA	Return-stack current word address; contains address one byte past current entry in stack; stack is null if RSCWA=X'184'.
RSEND	Return-stack ending address; contains address one byte past last allowable entry in stack; for a stack depth of 11 entries, RSEND=X'1B0'.
SYSPRIV1	If set indicates system privileges, level one.
SYSPRIV2	If set in addition to SYSPRIV1, indicates system privileges, level two.

Functional Element	Description
T0 through T3	Overlays accumulator and extension.
XMODE	This tally may be setup to a mode-id of a subroutine that is to gain control when a "forward link zero" condition occurs.
WMODE	If WMODE is non-zero on any entry to WRAPUP, a BSL* thorugh 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 loged 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.

5.1.9 PROGRAM DOCUMENTATION CONVENTIONS

In the following documentation, the functional description briefly describes the action taken by the routine. Unless otherwise specified, the program described is called as a subroutine, using the BSL instruction, and it returns to the calling program via a RTN (return) instruction.

The Input Interface, Internal Usage, and Output Interface sections describe the elements used by the subroutine. The single letter following the element name describes its type (C=character, D=double word, H=half word, R=address register, S=storage register, T=word). Unless otherwise specified, it should be assumed that the following elements may be internally destroyed by the routine:

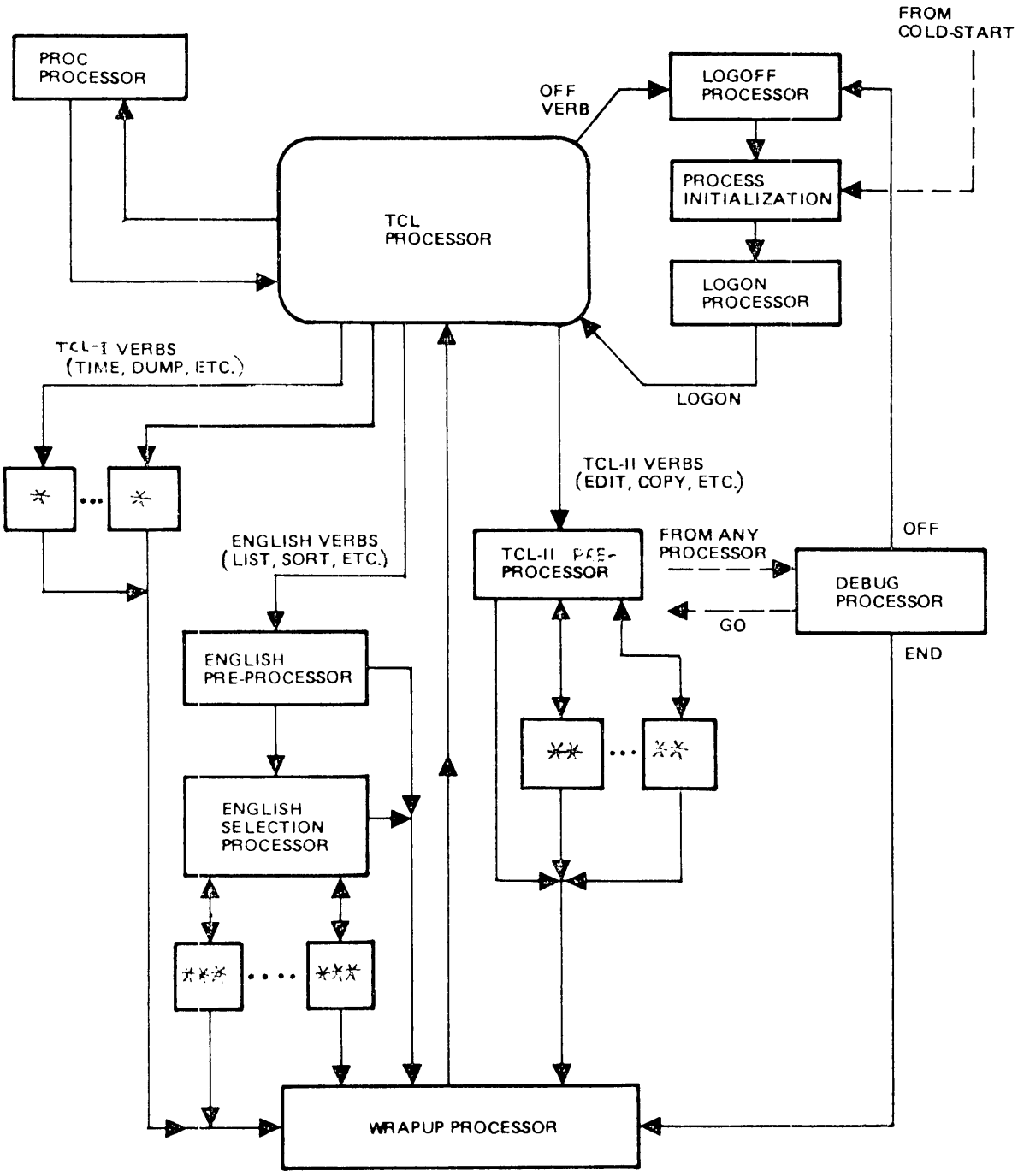
Registers	:	R14 and R15.
Storage Registers	:	SYSR0, SYSR1, SYSR2.
Tallies	:	Accumulator (D0, D1), D2, T4, T5.
Bits	:	Arithmetic condition flags, SB60, SB61.

If no description follows the element name, it indicates that the element is used as a scratch element.

The system delimiters are symbolically referred to as:

Hex. value	Name and description	
FF	SM	Segment Mark.
FE	AM	Attribute Mark.
FD	VM	Value Mark.
FC	SVM	Secondary Value Mark.
FB	SB	Start Buffer.

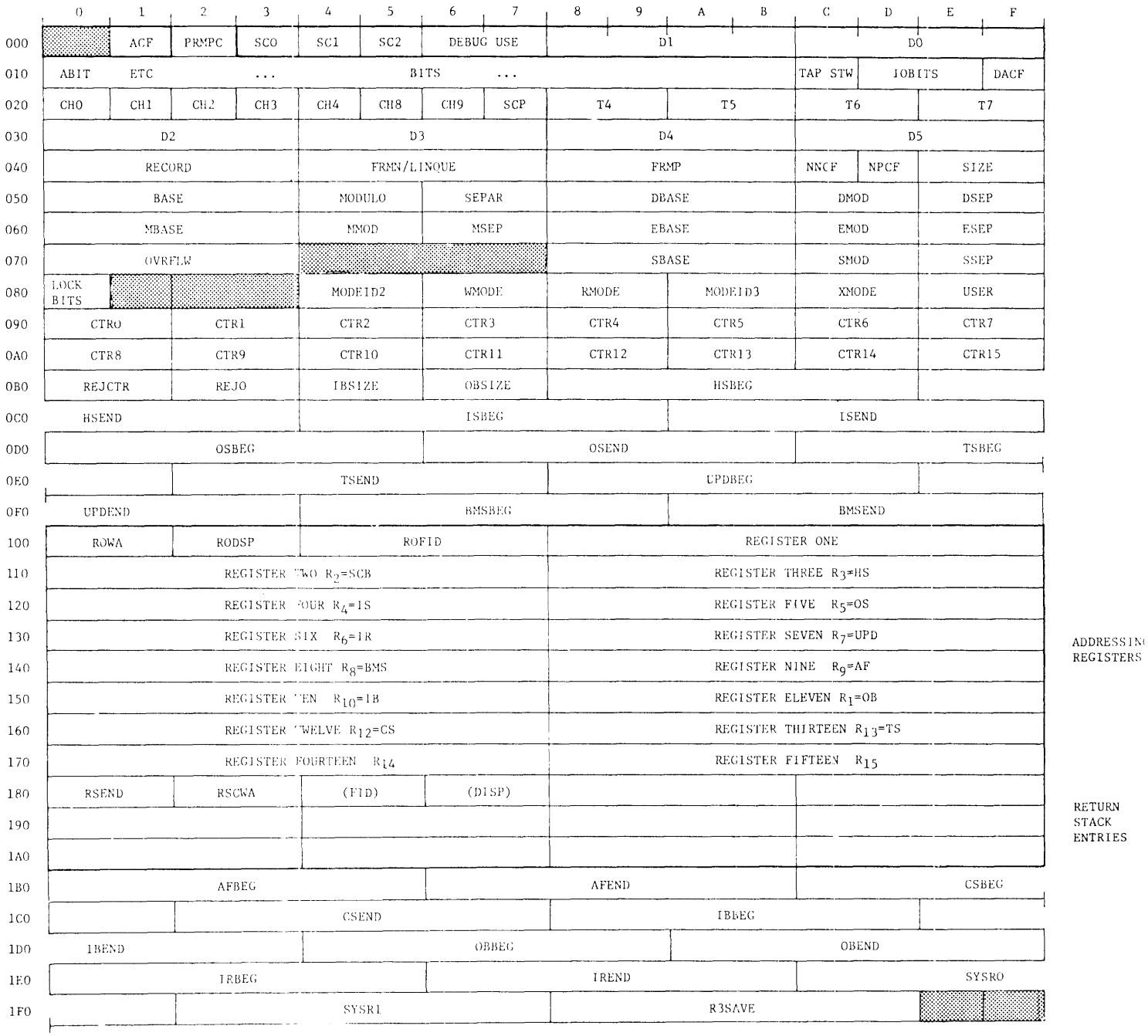
5.1.10 OVERALL VIEW OF SYSTEM SOFTWARE LINKAGE



*TCL-I processors
**TCL-II processors
***ENGLISH processors

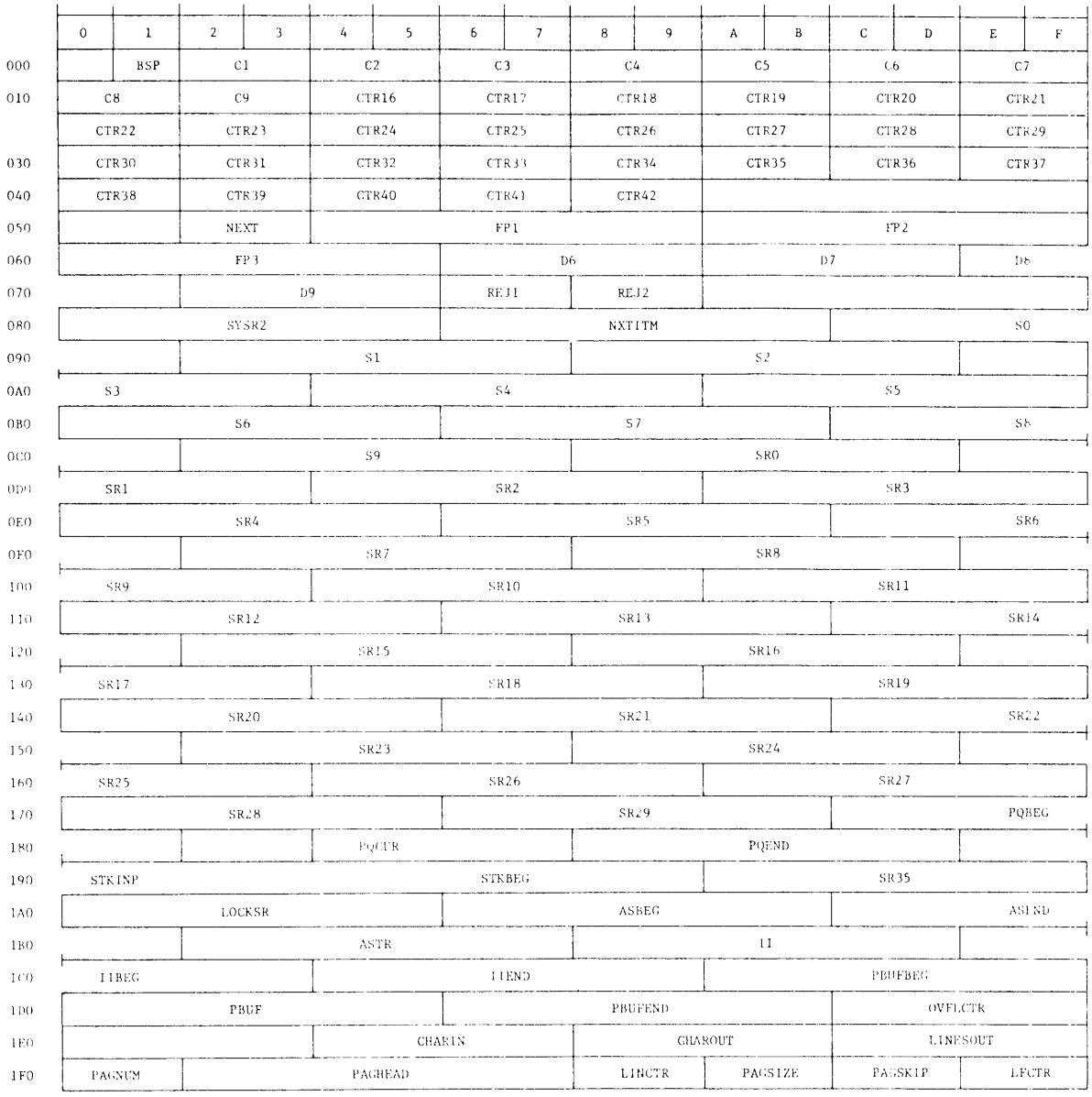
5.1.11 PRIMARY CONTROL BLOCK

Addressing register RO set to PCB. Areas bordered by heavy lines are accessed by hardware. Shaded areas are reserved for future system software use.



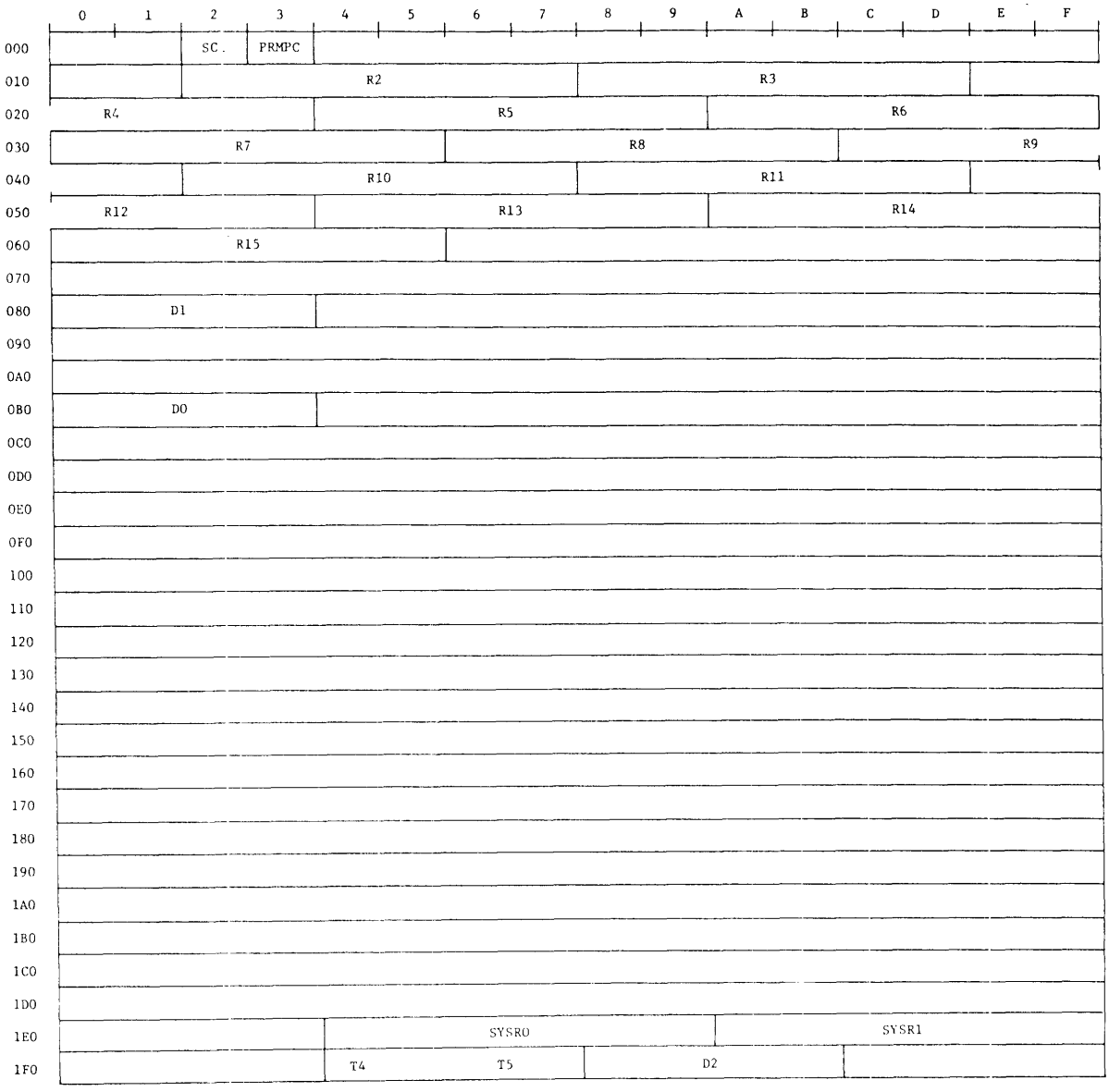
5.1.12 SECONDARY CONTROL BLOCK

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



5.1.13 DEBUG CONTROL BLOCK

PCB save areas. DCB = PCB + 2



5.1.14 PSYM

PSYM.....	D/CODE..	LINE 2	LINE 3
ABIT	B	080	0
ACF	H	001	0
AF	R	009	9
AFBEG	S	0D8	0
AFDSP	T	0A5	0
AFEND	S	0DB	0
AFFID	D	0A6	0
AM	N	0FE	0
ASBEG	S	0D3	2
ASEND	S	0D6	2
ASTR	S	0D9	2
ATTACH	B	0F2	0
B0	B	07F	0
B13	B	072	0
B15	B	070	0
B30	B	061	0
B31	B	060	0
B4	B	07B	0
BASE	D	028	0
BBIT	B	081	0
BITS	C	010	0
BKBIT	B	09B	0
BMS	R	008	8
BMSBEG	S	07A	0
BMSDSP	T	0A3	0
BMSEND	S	07D	0
BMSFID	D	0A2	0
BMSWA	T	0A0	0
BSPCH	C	001	2
C1	T	001	2
C2	T	002	2
C3	T	003	2
C4	T	004	2
C5	T	005	2
C6	T	006	2
C7	T	007	2
C8	T	008	2
C9	T	009	2
CBBIT	B	0D0	0
CBIT	B	082	0
CCDEL	B	0EB	0
CH0	C	020	0
CH1	C	021	0
CH2	C	022	0
CH3	C	023	0
CH4	C	024	0
CH8	C	025	0
CH9	C	026	0
COLHDRSUPP	B	0CB	0
CR	N	00D	0
CS	R	00C	C
CSBEG	S	0DE	0
CSDSP	T	0B1	0
CSEND	S	0E1	0
CSFID	D	0B2	0
CSWA	T	0B0	0
CTR	T	048	0
CTR0	T	048	0
CTR1	T	049	0
CTR10	T	052	0
CTR11	T	053	0
CTR12	T	054	0
CTR13	T	055	0
CTR14	T	056	0
CTR15	T	057	0
CTR16	T	00A	2
CTR17	T	00B	2
CTR18	T	00C	2
CTR19	T	00D	2
CTR2	T	04A	0
CTR20	T	00E	2
CTR21	T	00F	2
CTR22	T	010	2
CTR23	T	011	2
CTR24	T	012	2
CTR25	T	013	2
CTR26	T	014	2
CTR27	T	015	2
CTR28	T	016	2
CTR29	T	017	2
CTR3	T	04B	0
CTR30	T	018	2
CTR31	T	019	2
CTR32	T	01A	2
CTR33	T	01B	2
CTR34	T	01C	2
CTR35	T	01D	2
CTR36	T	01E	2
CTR37	T	01F	2
CTR38	T	020	2
CTR39	T	021	2
CTR4	T	04C	0
CTR40	T	022	2
CTR41	T	023	2
CTR42	T	024	2
CTR5	T	04D	0
CTR6	T	04E	0
CTR7	T	04F	0
CTR8	T	050	0
CTR9	T	051	0

(CURRENT LISTING TO BE SUPPLIED)

PSYM.....	D/CODE..	LINE 2	LINE 3	PSYM.....	D/CODE..	LINE 2	LINE 3
D0	D	006	0	H2	H	00D	0
D1	D	004	0	H3	H	00C	0
D2	D	018	0	H4	H	00B	0
D3	D	01A	0	H5	H	00A	0
D4	D	01C	0	H6	H	009	0
D5	D	01E	0	H7	H	008	0
D6	D	033	2	HBIT	B	087	0
D7	D	035	2	HDRSUPP	B	0C9	0
D8	D	037	2	HS	R	003	3
D9	D	039	2	HSBEG	S	05C	0
DACF	H	01F	0	HSEND	S	05F	0
DAF1	B	0D1	0	H7	H	009	0
DAF10	B	0C7	0	IB	R	00A	B
DAF2	B	0D2	0	IBBEG	S	0E4	0
DAF3	B	0D3	0	IBEND	S	0E7	0
DAF4	B	0D4	0	IBFID	D	0AA	0
DAF5	B	0D5	0	IBIT	B	088	0
DAF6	B	0D6	0	IBSIZE	T	05A	0
DAF7	B	0D7	0	IBWA	T	0A8	0
DAF8	B	0D8	0	IDSUPP	B	0C6	0
DAF9	B	0D9	0	II	S	0DC	2
DATEQ	T	040	0	IIBEG	S	0DF	2
DBASE	D	02C	0	IIEND	S	0E2	2
DBIT	B	083	0	INDEBUG	B	0F3	0
DBLSPC	B	0CA	0	INHIBIT	B	0Fu	0
DEBUG5	B	035	0	INHIBITSV1	B	0DA	0
DEBUG6	B	036	0	INHIBITSV2	B	0DB	0
DEBUG7	B	037	0	IOBIT14	B	0F6	0
DEBUGBYTE	H	006	0	IOBIT2	B	0EA	0
DISKERR	B	0F6	0	IOBIT4	B	0EC	0
DMOD	T	02F	0	IR	R	006	6
DSEP	T	02F	0	IRBEG	S	0F0	0
EBASE	D	034	0	IRDSP	T	099	0
EBIT	B	084	0	IREND	S	0F3	0
ECONVBIT	B	0CF	0	IRFID	D	09A	0
EMOD	T	036	0	IRWA	T	098	0
ENDBIT	B	09A	0	IS	R	004	4
EOFBIT	B	03A	0	ISBEG	S	062	0
EOTBIT	B	0E1	0	ISDSP	T	091	0
ESEP	T	037	0	ISEND	S	065	0
FBIT	B	085	0	ISFID	D	092	0
FP1	D	02A	2	ITAPEBIT	B	OCE	0
FP2	D	02D	2	JBIT	B	089	0
FP3	D	030	2	KBIT	B	08A	0
FRMN	D	022	0	LBIT	B	088	0
FRMP	D	024	0	LF	N	00A	0
GBIT	B	086	0	LFCTR	T	0FF	2
GMBIT	B	09C	0	LFDLY	T	0FF	2
H0	H	00F	0	LINCTR	T	0FC	2
H1	H	00E	0	LINESOUT	D	0F6	2

(CURRENT LISTING TO BE SUPPLIED)

PSYM.....	D/CODE..	LINE 2	LINE 3	PSYM.....	D/CODE..	LINE 2	LINE 3
LINQUE	D	022	0	PBUFEND	S	0EB	2
LISTFLAG	B	0F1	0	PQBEG	S	0BE	2
LOCK	H	000	0	PQCUR	S	0C1	2
LOCKSR	S	0D0	2	PQEND	S	0C4	2
LPBIT	B	0CD	0	PQFLG	B	0DE	0
MBASE	D	030	0	PRMPC	C	002	0
MBIT	B	08C	0	PROTECT	B	0E0	0
MMOD	T	032	0	PTIME	T	025	2
MODEID2	T	042	0	QBIT	B	090	0
MODEID3	T	045	0	QSTR	S	0E8	2
MODULO	T	02A	0	RO	R	000	0
MSEP	T	033	0	RODSP	T	081	0
NBIT	B	08D	B	ROFID	D	082	0
NEGBIT	B	00E	0	ROWA	T	080	0
NEXT	T	029	2	R1	R	001	1
NNCF	H	04C	0	R10	R	00A	A
NOBIT	B	09A	0	R10FID	D	0AA	0
NOBLNK	B	0EE	0	R10WA	T	0A8	0
NPCF	H	04D	0	R11	R	00B	B
NREC	D	020	2	R11DSP	T	0AD	0
NXTITM	2	043	2	R11FID	D	0A3	0
OB	R	00B	B	R11WA	T	0AC	0
OBBEG	S	0EA	0	R12	R	00C	C
OBDSP	T	0AD	0	R12DSP	T	0B1	0
OBEND	S	0ED	0	R12FID	D	0B2	0
OBFID	D	0AE	0	R12WA	T	0B0	0
OBIT	B	08E	0	R13	R	00D	D
OBSIZE	T	05B	0	R13DSP	T	0B5	0
OBWA	T	0AC	0	R13FID	D	0B6	0
OS	R	005	5	R13WA	T	0B4	0
OSBEG	S	068	0	R14	R	00E	3
OSBEGF	D	069	0	R14DSP	T	0B9	0
OSDSP	T	095	0	R14FID	D	0BA	0
OSEND	S	06B	0	R14WA	T	0BC	0
OSFID	D	096	0	R15	R	00F	F
OSWA	T	094	0	R15DSP	T	0BD	0
OVFBIT	B	00F	0	R15FID	D	0BE	0
OVRFLCTR	D	0EE	2	R15WA	T	0BC	0
OVRFLW	D	038	0	R2	R	002	2
OVRFLWO	Z	006	E	R2DSP	T	089	0
PAGFRMT	B	0CC	0	R2FID	D	08A	0
PAGHEAD	S	0F9	2	R2WA	T	088	0
PAGINATE	B	0F7	0	R3	R	003	3
PAGNUM	T	0F8	2	R3DSP	T	08D	0
PAGSIZE	T	0FD	2	R3FID	D	08E	0
PAGSKIP	T	0FE	2	R3SAVE	S	0FC	0
PARITY	B	0E4	0	R3WA	T	08C	0
PBIT	B	08F	0	R4	R	004	4
PBUF	S	0E8	2	R4DSP	T	091	0
PBUFBEG	S	0E5	2	R4FID	D	092	0

(CURRENT LISTING TO BE SUPPLIED)

PSYM.....	D/CODE..	LINE 2	LINE 3	PSYM.....	D/CODE..	LINE 2	LINE 3
R4WA	T	090	0	SB11	B	0AB	0
R5	R	005	4	SB12	B	0AC	0
R5DSP	T	095	0	SB13	B	0AD	0
R5FID	D	096	0	SB14	B	0AE	0
R5WA	T	094	0	SB15	B	0AF	0
R6	R	006	6	SB16	B	0B0	0
R6DSP	T	099	0	SB17	B	0B1	0
R6FID	D	09A	0	SB18	B	0B2	0
R6WA	T	098	0	SB19	B	0B3	0
R7	R	007	0	SB2	B	0A2	0
R7DSP	T	09D	0	SB20	B	0B4	0
R7FID	D	09E	0	SB21	B	0B5	0
R7WA	T	09C	0	SB22	B	0B6	0
R8	R	008	8	SB23	B	0B7	0
R8DSP	T	0A1	0	SB24	B	0B8	0
R8FID	D	0A2	0	SB25	B	0B9	0
R8WA	T	0A0	0	SB26	B	0BA	0
R9	R	009	9	SB27	B	0BB	0
R9DSP	T	0A5	0	SB28	B	0BC	0
R9FID	D	0A6	0	SB29	B	0BD	0
R9WA	T	0A4	0	SB3	B	0AE	0
RBIT	B	091	0	SB30	B	0BE	0
RECORD	D	020	0	SB31	B	0BF	0
REJO	T	059	0	SB32	B	0CO	0
REJ1	T	03B	2	SB4	B	0A4	0
REJ3	T	03D	2	SB5	B	0A5	0
REJ4	T	03E	2	SB6	B	0A6	0
REJ5	T	03F	2	SB60	B	0DC	0
REJCTR	T	058	0	SB61	B	0DD	0
RMBIT	B	09E	0	SB7	B	0A7	0
RMODE	T	044	0	SB8	B	0A8	0
RNIBIT	B	033	0	SB9	B	0A9	0
RNICTR	H	007	0	SBASE	D	03C	0
RSCWA	T	0C1	0	SBIT	B	092	0
RSEND	T	0C0	0	SC0	C	003	0
RTNSTK	T	0C2	0	SC1	C	004	0
S0	S	046	2	SC2	C	005	0
S1	S	049	2	SCP	C	027	0
S2	S	04C	2	SEPAR	T	02B	0
S3	S	04F	2	SIZE	T	027	0
S4	S	052	2	SM	N	0FF	0
S5	S	055	2	SMBIT	B	09F	0
S6	S	058	2	SMCONV	B	0ED	0
S7	S	05B	2	SMOD	T	03E	0
S8	S	05E	2	SR0	S	064	2
S9	S	061	2	SR1	S	067	2
SB	N	0FB	0	SR10	S	082	2
SB0	B	0A0	0	SR11	S	085	2
SB1	B	0A1	0	SR12	S	088	2
SB10	B	0AA	0	SR13	S	08B	2

(CURRENT LISTING TO BE SUPPLIED)

PSYM.....	D/CODE..	LINE 2	LINE 3
SR14	S	08E	2
SR15	S	091	2
SR16	S	094	2
SR17	S	097	2
SR18	S	09A	2
SR19	S	09D	2
SR2	S	06A	2
SR20	S	0A0	2
SR21	S	0A0	2
SR22	S	0A6	2
SR23	S	0A9	2
SR24	S	0AC	2
SR25	S	0AF	2
SR26	S	0B2	2
SR27	S	0B5	2
SR28	S	0B8	2
SR29	S	0BB	2
SR3	S	06D	2
SR35	S	0CD	2
SR4	S	070	2
SR5	S	073	2
SR6	S	076	2
SR7	S	079	2
SR8	S	07C	2
SR9	S	07F	2
SSEP	T	03F	0
STKBEG	S	0CA	2
STKFLG	B	0E8	0
STKFLGX	B	0E9	0
STKINP	S	0C7	2
SVM	N	0FC	0
SYSPRIV1	B	0F4	0
SYSPRIV2	B	0EF	0
SYSRO	S	0F6	0
SYSR1	S	0F9	0
SYSR2	S	040	2
T0	T	007	0
T1	T	006	0
T2	T	005	0
T3	T	004	0
T4	T	014	0
T5	T	015	0
T6	T	016	0
T7	T	017	0
TAPSTW	C	01C	0
TBIT	B	093	0
TPRDY	B	0E7	0
TS	R	00D	D
TSBEG	S	06E	0
TSDSP	T	0B5	0
TSEND	S	071	0
TSWA	T	0B4	0
TTLY	T	003	0
TYMO	T	022	2
UBIT	B	094	0
UPD	R	007	7
UPDBEG	S	074	0
UPDDSP	T	09D	0
UPDEND	S	077	0
UPDFID	D	09E	0
UPDWA	T	09C	0
USER	T	047	0
VBIT	B	095	0
VM	N	0FD	0
VOBIT	B	0DF	0
VT	N	00B	0
WBIT	B	096	0
WMBIT	B	09D	0
WMODE	T	043	0
XBIT	B	097	0
XMODE	T	046	0
XNFID	D	001	F
XNLOCK	H	000	F
XNNCF	H	001	F
XNPCF	H	00A	F
XNRES	H	00B	F
XPFID	D	003	F
YBIT	B	098	0
ZBIT	B	099	0
ZEROBIT	B	00D	0

(CURRENT LISTING TO BE SUPPLIED)

5.2 TCL PROCESSORS AND PROC INTERFACE

5.2.1 VERB FORMAT

A verb is an entry in the master dictionary whose D/CODE begins with the character "P". The special sequence "PQ" is reserved, and defines a PROC; otherwise the format of the verb is as defined below. The verb contains three mode-id's - hexadecimal character fields that specify the transfer of control from one processor to another. The first mode-id is mandatory and specifies the location to which TCL-I transfers control after editing the input statement; the second mode-id is also mandatory for TCL-II and ENGLISH verbs, and specifies a secondary processor exit. The third mode-id is usually optional. In addition, an option string may be present for TCL-II verbs.

Verb Format:

Attribute Number	Description	Examples
1	"Px"; x is a single character (not "Q") stored in the character register SCP. X may be null.	LIST verb: PA COUNT verb: PB
2	Primary mode-id, to which TCL-I transfers control.	ALL ENGLISH verbs: 35 ALL TCL-II verbs: 2
3	Secondary mode-id; stored in the tally MODEID2	EDIT verb: D ASSEMBLE verb: 17
4	Tertiary mode-id; stored in the tally MODEID3	
5	Option string, for TCL-II verbs (see TCL-II documentation)	

5.2.2 TCL-I

Functional Description

TCL-I is the basic entry point (not a subroutine) for the terminal control language process. It is entered solely from the WRAPUP processor after WRAPUP has completed processing of the previous TCL statement.

The primary functions of the TCL-I processor are as follows:

1) Determine if a PROC is in control, and if it is, exit to the PROC processor for continuation of the PROC.

2) If not, obtain a line of input from the terminal.

3) Attempt to retrieve the verb (first set of contiguous non-blank data in the input buffer) from the master dictionary and validate it as such.

4) Set up the parameters from the verb; edit and copy the remainder of the data in the input buffer to the work-space IS.

5) Exit to the processor specified in the primary mode-id parameter of the verb.

Editing Features

1) All control characters, and system delimiters (SB, SM, AM, VM, SVM) in the input buffer are ignored.

2) Redundant blanks (sequence of two or more blanks) are not copied, except in strings enclosed by single or double quote signs.

3) Strings enclosed in single quote signs are copied as: (SM) I string (SB)

4) Strings enclosed in double quote signs are copied as: (SM) V string (SB)

5) End of data is marked with a: (SM) Z

Output Interface

ISBEG	S	Defines start of work space where edited input data has been copied.

IS	R	=ISBEG
IR	R	Points to AM following attribute 4 of the verb, or to end-of-data AM of verb.
SR4	S	Points to AM at end-of-data of verb.
IBBEG	S	Points to start of last input line from terminal.
IB	R	Points to SM terminating input line.
IBEND	S	As above.
SCP	C	Contains character following "P" in verb attribute one; blank if none specified.
SC0	C	Contains a blank
SC1	C	Contains a blank
SC2	C	Contains a SB
MODEID2	T	Contains secondary mode-id from verb; zero if none specified.
MODEID3	T	Contains tertiary mode-id from verb; zero if none specified.
IBIT	B	Set if any string enclosed by single-quote signs has been found.
VBIT	B	Set if any string enclosed by double-quote signs has been found.
PQFLG	B	Set if a PROC is in control.
BASE MODULO SEPAR		Contains base-FID, modulo and separation of of master dictionary.

All other bits are zero (A through Z bit and SB0 through SB32. All other process work-space pointers are set to their initial condition. (See Section 5.8.1.)

Subroutines Used

RETIX: CVTHIR

Error Conditions

The following cause an exit to the WRAPUP processor with the message indicated:

271: One PROC cannot call another

3: Verb cannot be identified in the M/DICT.

30: Verb format error. (Premature end-of-data, non-hex character in mode-id's)

2: Uneven number of single or double quote signs in input data.

5.2.3 TCL-II

Functional Description

TCL-II is entered from TCL-I by those verbs requiring access to a file, and to all, or explicitly specified items, from the file. TCL-II exits to the processor whose mode-id is specified in MODEID2; typically processors such as the EDITOR, ASSEMBLER, LOADER, etc. use TCL-II to feed them the set of items which was specified in the input data

On entry, TCL-II checks the verb definition for a set of option characters (in attribute 5 of the verb); verb options are single characters as below; any combination may be specified.

Option Character	Meaning
C	Copy - the item retrieved is copied to the workspace IS.
F	File access only - the item-list is ignored; only the file parameters are set up by TCL-II. If this option is present, any others are ignored.
N	New item acceptable - if the item specified is not on file, the secondary processor still gets control (example: the EDITOR can process a new item).
P	Prints item-id on a full-file retrieval, as each item is retrieved.
U	Updating sequence flagged - if items are to be updated as retrieved, this option is mandatory.
Z	Final entry - the secondary processor will be entered once more after all items have been retrieved (example: COPY processor, to print a message).

The input data string to TCL-II consists of the file-name (optionally preceded by the modifier DICT, which specifies access to the dictionary of the file), followed by a list of items, or an asterisk (*) specifying retrieval of all items on the file. The item-list may be followed by an option list (options to the secondary processor) which must be enclosed in parentheses. These options must consist of a sequence of single characters, or a decimal number, or two decimal numbers separated by a minus sign (specifying a range of numbers). Multiple options are separated by commas. The option characters A through Z set the corresponding bits ABIT through ZBIT (A sets ABIT, etc.); the numbers are stored in tallies D4 and D5, and the bit NOBIT set if the numerical option is found.

Output Interface

DAF1	B	Set if update option was found.
DAF2	B	Set if "C" otpion was found.
DAF3	B	Set if "P" option was found.
DAF4	B	Set if "N" option was found.
DAF5	B	Set if "Z" option was found.
DAF6	B	Set if "F" option was found.
DAF8	B	Set if accessing a dictionary-file (DICT in input).
RMBIT	B	Set if item found and retrieved.
BASE	D	First exit only: base FID, modulo and separation of file being accessed.
MODULO	T	
SEPAR	T	
SBASE	D	Base FID, modulo and separation of file being accessed.
SMOD	T	
SSEP	T	
IS	R	Points to non-blank following file-name if "F" option specified.
DBASE	D	Contains base-FID, modulo and separation of dictionary of file being accessed if "F" option is specified.
DMOD	T	
DSEP	T	

Following specifications meaningful only if "F" option is not present:

BMSBEG	S	Points one prior to area containing the item-id, which is terminated by a AM.
ABIT through ZBIT	B	Set according to option list (see description above).
NOBIT	B	Set if numerical option found.
D4	D	Contains numerical option value(s).
D5	D	
SR0	S	Points one prior to count field of item on file.
SIZE	T	Contains count field of item.
SR4	S	Points to last AM of item on file.

MODEID3	T	Contains tertiary mode-id from verb.	
		"C" Option in Verb	No "C" Option in Verb
ISBEG	S	Points one prior to beginning of copied item (including item-id, not including count field)	
IS	R	Points to last AM of copied item.	Points to end of string.
ISEND	S	=IS	
IR	R	Points to last AM of item on file.	Points to AM following item-id on file.

Internal Usage

RMODE	T	Contains mode-id of entry-point within TCL-II that WRAPUP processor exits to; must be maintained by lower-level processors.

All elements as used by GETITM (q.v.)

Subroutines Used

RETIX, GBMS, GDLID, GETITM, GETOPT, GETFILE.

Error Conditions

The following conditions cause an exit to the WRAPUP processor with the error number indicated:

200	File-name not specified
201	File name illegal or incorrectly defined in the M/DICT.
202	Item not on file (will not abort processing).
203	Item list missing.
204	Error in format of option list.
13	DL/ID item not found in dictionary-file.

5.2.4 USER EXITS FROM PROC

A user- program can gain control during execution of a PROC, by using the Uxxxx command in the Proc, (where xxxx is the hex. mode-id of the user-program). The user-program can perform special processing, and then return control to the PROC processor. Necessarily, certain elements used by PROC must be maintained by the user-program. These elements are marked with an asterisk in the table below:

Input Interface

*BASE *MODULO *SEPAR	D T T	Contains FID, modulo and separation of M/DICT.
*PQBEG	S	Points one prior to the first PROC statement.
*PQEND	S	Points to terminal AM of the PROC.
PQCUR	S	Points to AM following the Uxxxx element.
IR	R	=PQCUR
*PBUFBEG	S	Points to buffer containing primary and secondary (if any) input buffers. Format: (SB) ... primary input ... (SM) (SB) ... secondary input ... (SM)
*ISBEG	S	Points to buffer containing primary output line.
*STKBEG	S	Points to buffer containing "stacked input" (secondary output)
*SBIT	B	Set if ST ON command is in effect.
IB	R	Current input buffer pointer (may be within primary or secondary input buffers).
*SC2	C	Contains a blank.

		SBIT on	SBIT off
IS	R	Last byte moved into secondary output buffer.	Last byte moved into primary output buffer.
UPD	R	Last byte moved into primary output buffer.	Last byte moved into secondary output buffer.

Output Interface

IR	R	Points to AM preceding next PROC statement to be executed; may be altered to change location of continued PROC execution.
IS	R	May be altered as needed to alter data within input and output buffers; but formats described above must be maintained.
UPD	R	
IB	R	

Exit Convention

The normal method of returning control to the PROC processor is to execute an external branch to 2, PROC-I. If it is necessary to abort PROC control and exit to WRAPUP, set PQFLG off, and execute an external branch to one of the WRAPUP entry points.

Note that, when the PROC eventually transfers control to TCL, (via the P operator), certain elements are expected to be in an initial condition. Therefore, if the user-program uses these elements, they should be reset before returning to PROC, unless the elements are deliberately set up as a means of passing parameters to other processors. Specifically, the bits ABIT through ZBIT are expected to be zero by the TCL-II and ENGLISH Processors. It is best to avoid usage of these bits in PROC user-exits. The scan character registers SC0, SC1 and SC2 must also contain an SB, a blank and a blank respectively.

5.3 WRAPUP PROCESSOR

The WRAPUP Processor is entered under the following conditions:

1) Termination of a TCL statement, when it is required to "wrap-up" processing and to return to the TCL level.

2) Intermediate stage in processing a statement, when it is required to print messages from the ERRMSG file, or to perform disc updates, and then return to the calling processor.

3) User-initiated termination of processing, via the 'END' command in DEBUG.

The WRAPUP Processor has several entry points, depending on the type of action required; several of these entry points are provided to simplify the interface requirements when an error message is required (Note, for instance, that MD995 may be entered immediately after return from a call to, say, RETIX, if the item is not found on file, by setting up C1 to the error message number).

WRAPUP also performs the following functions before returning to TCL:

1) Closes all open spool files, if LPBIT is set.

2) Releases linked overflow space if OVRFLCTR ≠ 0.

5.3.1 WRAPUP-I

Functional Description

1. Prints, or sets up for printing, messages that are stored in the file "ERRMSG" (must be catalogued in the process M/DICT.

2. Performs disk updates as specified in the history string.

3. Terminates processing of a TCL statement; re-initializes elements.

Interface Requirements

History String. The history string is from HSBEG through HSEND. If HSBEG=HSEND, the string is null; this is the initial condition on

entry to TCL. If HSBEG ≠ HSEND, elements in the string are processed by WRAPUP. There are three types of elements; all other element types are ignored.

1. Output message.

   (SM) 0 (AM) Message-Id (AM) (Parameter (AM) ...) (SM)

   where Message-Id is the item-id of an item in the ERRMSG file. (Normally a decimal numeric).

   Parameters are character string that is to be passed to the message formatter (PRTERR, WRAPUP-III).

2. Disk Update/Delete string.

   (SM) DU (AM) base (VM) modulo (VM) separ (AM) item-id (AM) .. (item-body) (AM) (SM)

   (SM) DD (AM) base (VM) modulo (VM) separ (AM) item-id (AM) (SM)

   DU - Disk update; replaces entire item in the file specified by the decimal parameters base, modulo and separ.

   DD - Disk delete; deletes item from the file.

3. End-of-string element

   (SM)Z

Conventionally, a process wishing to add data to the history string begins at HSEND+1; when the entire additional element(s) has been added, the string is terminated with a (ZM) Z, and HSEND reset to the (SM).

If WMODE is non-zero on any entry to WRAPUP, an indirect subroutine call (3SL*) via WMODE will be executed. This allows special processing to be done on every WRAPUP entry.

WRAPUP may be called as if it is a subroutine by setting RMODE to the Mode-ID of the program to which WRAPUP returns control to; note however, that the return-stack is always set to a null or empty condition by WRAPUP. On the error-message setup entry points (MD99/MD993/MD994/MD995), if VOBIT is set and RMODE non-zero, the appropriate messages are stored in the history string, for printing on a final entry with RMODE zero.

If OVRFLCTR is non-zero, it is assumed that it contains the starting FID of a linked set of overflow frames that is to be released to the system overflow pool. This tally is used, for instance, by the SORT processor to store the beginning FID of the sorted table; the overflow space used by the Sort is thus always released to the system even if the sort is aborted by the debug 'END' command.

Entry Points

MD993	A message number is stored in C1; a numeric parameter is stored in C2. Sets up the message in the history string and exits to MD99.
MD994	A message number is stored in C1; a character string parameter is stored for IS+1 through an AM or SM. Sets up the message and exits to MD99.
MD995	As above, except that the parameter is from BMSBEG+1 through an AM or SM.
MD99	Message numbers (without any parameters) may be stored in REJCTR, REJ0 and REJ1 (no action is taken if zero.) After setting up the messages in the history string, exits to MD999 (If VOBIT set, skips history string processing in MD999).
MD999	Processes all elements in the history string. Reinitializes process work spaces; exits to TCL if RMODE = 0; to calling program via RMODE if non-zero.
TCL	Kills history string; PROC control; exits to TCL (Entry point of "END" command for DEBUG).

Subroutines Called

PRTEER	(WRAPUP-III) to print error messages.
UPDITM	(WRAPUP-II) to perform disk updates.
ISINIT	To re-initialize ISBEG/ISEND/OSBEG/OSEND.
WSINIT	To re-initialize process work areas. (BMS/CS/AF/IB/OB/TS)
CVTNIS	To convert a decimal character string, from the IS, to binary.

External Branches

LOGOFF	If USER = 0

Error Messages

"DISK UPDATE STRING ERROR"; self explanatory.

5.3.2 UPDITM (WRAPUP-II)

This subroutine performs updates to the disc files in the system. It is described in Section 5.4.3.

5.3.3 PRTEER (WRAPUP-III)

Functional Description

This subroutine is used primarily by the WRAPUP processor to retrieve and print error messages from the system file ERRMSG. A parameter string may be passed to the subroutine, which will format and insert the parameters as specified by the codes in the message item. System message item-ids are numeric; however, any item-id can be specified. See description of error messages for format of the codes.

Input Interface

TS	R	Points one prior to the message item-id, which must be terminated by an AM. Parameters optionally follow, being delimited by AM's. The parameter string must terminate with a SM.

Output Interface

TS	R	Points to AM following message-id, or AM or SM following last parameter output.

Internal Usage

EBASE EMOD ESEP	D T T	Set up by PRTERR to ERRMSG file, if EBASE = 0 on entry; otherwise these elements are assumed to be already initialized.

All elements are used by WRTLIN and RETIX.

Error Message Formats

A (dec-number)	Parameter insertion code; the next parameter from the history string, if any, is placed in the output buffer. If "dec-number" is specified, the parameter is left-justified in a blank field of the specified length.

R (dec-number)	As above; the parameter is right-justified in the blank field of the specified length.
E (char-string)	The message ID, surrounded by brackets, is placed in the OB, followed by the optional character string.
H (char-string)	The character string is placed in the OB.
L (dec-number)	The output buffer is printed, and the specified number of line feeds are output (one if "dec-number" not specified).
X (dec-number)	The OB is incremented by the number of spaces specified. If the end of the line is reached, the output buffer is printed and a new line is started.
T	Adds system time in the format HH:MM:SS to the output buffer.
D	Adds system date in the format DD MMM YYYY to the output buffer.

On exit from this subroutine, the output buffer is printed and a new line started, unless the last character in the OB is an "+", which causes printing of the buffer only.

5.3.4 FUNCTIONAL ELEMENT USAGE BY ALI WRAPUP MODES

Bits	Description of Use	Mode
VOBIT	Store/Print Flag; Store Messages if set	ALL
SB60	Scratch	Wrapup-III
SB61	Scratch	Wrapup-III
RMBIT	As used by RETIX	II, III
Tallys		
REJCTR REJ0 REJ1	Input error messages	I(MD99)
C1	Input error messages	I (MD993/994/995)
C2	Input error messages	I (MD993)

Bits	Description of Use	Mode
C3	Scratch	II
D2	Scratch	II
D3	Scratch	II
SIZE	Scratch	II
NEXT	Scratch	II
RECORD, LINK(S)	As used by disc-I/O subroutines	II
OVRFLW	Scratch	I, II
D4	Scratch	II
T4	Scratch	II
BASE	Various	II
MODULO		I, II, III
SEPAR		
EBASE	Base-FID, modulo and separation	
EMOD	of the ERRMSG file.	III
ESEP		
RMODE	Return mode-ID	I
WMODE	Special processing exit mode-ID	I

Registers

HSBEG HSEND	Beginning and end of history string	I
BMSBEG BMSEND CSBEG CSEND ISBEG ISEND OSBEG OSEND IBBEG IBEND OBBEG OBEND AFBEG AFEND	Reinitialized on exit from MD999	
UPD	Scratch	I, II
IR	Scratch	
TS	Scratch	I, II, III
AF	Scratch	III
Return Stack	Reset to null condition on exit from MD999	

5.4 DISC FILE I/O

5.4.1 RETIX AND RETI

Functional Description

Retrieves an item stored in a file. The item-id is explicitly specified to this routine, as are the file parameters base, modulo and separation. The subroutine performs a "hashing" algorithm (see HASH documentation) to determine the group within which the item may be present, and then searches sequentially down the data in the group for a matching item-id. If found, the subroutine returns pointers to the beginning and end of the item, and the item size (from the item count-field).

The item-id is specified in a buffer defined by the register BMSBEG; if the entry RETIX is used, the item-id must be terminated by an AM; if RETI is used, the register BMS must point to the last byte of the item-id. An AM will be appended to the item-id by RETI.

Input Interface

BMSBEG	S	Points one prior to the item-id required.
BMS	R	RETIX: not an input interface requirement. RETI: points to the last byte of the item-id.
BASE	D	Contains the base FID of the file.
MODULO	T	Contains the modulo (number of groups) of the file.
SEPAR	T	Contains the separation (number of frames per group) of the file.

Internal Usage

XMODE	T	Used to exit to subroutine IROVF if a group format error occurs.

Output Interface

BMS BMSEND	R S	Points to last character of item-id.
RECORD		Contains the beginning FID of the group to which the item-id hashes.

NNCF FRMN FRMP NPCF	T D D T	Contain the link fields of the frame above.
XMODE	T	Zero

		Item Found	Item Not Found
RMBIT	B	Set.	Zero.
SIZE	T	Item count-field.	Zero.
R14	R	One prior to item count field.	Points to last AM of last item in group.
IR	R	Points to first AM of item	Points to AM indicating end of group data (=R14 +1).
SR4	S	Points to last AM of item	=R14

Subroutine Usage

RDREC, HASH, IROVF	Requires one additional level of subroutine linkage; three if a group format error occurs.

Error Conditions

If the data in the group is bad - premature end of linked frames, or non-hexadecimal character encountered in the count field-the message:

****GROUP FORMAT ERROR AT: .xxxxxx

is returned (xxxxxx is six character hexadecimal FID indicating where the error was found), and the routine returns with an "item not found" condition. Data is not destroyed, and the group format error will remain.

5.4.2 GETITM

Functional Description

Sequentially retrieves all items in a file. This routine is called repetitively to obtain items from a file one at a time until all items have been retrieved. The order in which the items are returned is the same as the pseudo-random storage sequence.

If the items retrieved are to be updated by the calling routine (using The routine UPDITM), this should be flagged to GETITM, which will then perform a two-stage retrieval process by first storing all item-id's (per group) in a table, then will use this table to actually retrieve the items on each call. This is necessary because, if the calling routine updates an item, the data within this group shifts around; GETITM cannot simply maintain a pointer to next item in the group, as it does if the "update" option is not flagged.

An initial entry condition has also to be flagged to GETITM; it then sets up and maintains certain pointers which should not be altered by the calling routines until all the items in the file have been retrieved.

Note the functional equivalence of the output interface elements with those of RETIX.

Input Interface

DAF7	B	Initial entry flag; must be zeroed on first call to GETITM.
DAF1	B	If set, "update" option is in effect.
DBASE DMOD DSEP	D T T	Contains base FID, modulo and separation of the file
BMSBEG	S	Points one prior to an area where the item-id of items retrieved may be copied. Must be at least 50 bytes in length.

Internal Usage

RECORD	D	
NNCF, FRMN, FRMD, NPCF, XMODE, EOFBIT		Used internally.

These elements should not be altered by any other routine while GETITM is used.

DAF7	B	See above
DAF1	B	
DBASE	D	Current group beginning FID.
DMOD	T	Number of groups left to be processed.
DSEP	T	Separation parameter of file (unchanged).
SBASE	D	Saved values of original base FID, modulo and separation.
SMOD	T	
SSEP	T	
NXTITM	S	If DAF1 set: Points one before next item-id in prestored table. If DAF1 zero: Points to last AM of item previously returned.
OVRFLCTR	D	Overflow space table starting FID; used if DAF1 set.

Output Interface

RMBIT	B	Set if item found; zeroed if all items exhausted.
SIZE	T	Contains item count-field.
R14	R	Points one prior to count-field.
SR0	S	=R14
IR	R	Points to first AM of item.
SR4	S	Points to last AM of item.
XMODE	T	Zero

Subroutines Used

RDREC, GNSEQI, GNTBLI (last two local); requires one additional level of subroutine linkage.

Error Conditions

As for RETIX (q.v.); except that the routine will continue retrieving items, if any more exist, after the error condition is reported.

5.4.3 UPDITM

Functional Description

This routine performs updates to a disc file defined by its base FID, modulo and separation. If the item is to be deleted, the routine will compress the remainder of the data in the group in which the item resided; if the item is to be added, it will be added at the end of the current data in the group; if the item is to be replaced, functionally a deletion and then an addition takes place. UPDITM does not perform a merge into an already existent item.

If the change of data in the group reaches an end of the linked frames, UPDITM will obtain another frame from the overflow space pool and link it to the previous linked set; as many frames as required will be added. If the deletion or replacement of an item causes an empty frame at the end of the linked frame set, and that frame is not in the "primary" area of the group, it will be released to the overflow space pool.

Once this routine is entered, the processing cannot be interrupted until completed.

Input Interface

BMSBEG	S	Points one prior to the item-id to be updated; the item-id must be terminated by an AM.
TS	R	Points one prior to the item body; the data must be terminated by a SM. This is not an input interface element for item-deletes.
CH8	C	Contains the character 'U' for an item-addition or replacement; 'D' for an item-delete.
BASE	D	Contains the base FID, modulo and separation of the file being updated.
MODULO	T	
SEPAR	T	

Internal Usage

RMBIT	B	Various
INHIBITSV1	B	Saves condition of INHIBIT
CTR0	T	
CTR1	T	

D2	D	
D3	D	
XMODE	T	

Output Interface

TS	R	Points to SM terminating item-body for item-add or replace.
IR	R	Points to AM terminating group data
UPD	R	Points to last byte of last frame in group
CS	R	Points one prior to item-id on item-add or replace
BMS	R	Points to AM following item-id
SIZE	T	Contains new item size on add or replace

Subroutine Usage

RETIX, RDREC, RDLINK, WTLINK, IROVF, BMSOVF, ATTOVF, RELOVF. Uses two additional levels of subroutine linkage.

A subroutine called LOCK is used by UPDITM. UPDITM is the subroutine called to most file updates, and locking this mode prevents simultaneous updates to files. This will eliminate the generation by UPDITM of group format errors, which can result if two processes attempt to update the same file simultaneously.

Note that no retrieval lock feature is used. This means that a 'TEMPORARY' group format error can appear on accesses to a file which is being updated. On a subsequent access, this group format error will not appear.

DATA/BASIC, which uses its own code for some updates, also uses LOCK.

Error Conditions

1. If the group data is bad, premature end of linked data set, or non-hexadecimal character found in a count field, the group data is terminated at the last good item, and the message:

   *** GROUP FORMAT ERROR AT: . xxxxxx

   is returned (xxxxxx is a six digit hexadecimal FFD indicating the location of the error), before the process continues.

2. If the file being updated is the M/DICT (as indicated by BASE being equal to MBASE), and the system privilege level one flag is not set, the routine aborts with the message:

   YOUR SYSTEM PRIVILEGE LEVEL IS NOT SUFFICIENT FOR THIS STATEMENT

   The update is not performed.

3. If the item-id is greater than 50 characters, the update is not performed; no indication is returned.

4. If the item exceeds the maximum size (32767 bytes, X'7FFF'), the item is truncated to 32767 bytes; no indication is returned.

5.4.4 GBMS

Functional Description

Sets up the base FID, modulo and separation parameters of a file from the file definition item that has been retrieved. Typically this routine will be called after a call to RETIX which retrieves the file-name from the master dictionary.

The routine handles both 'D' and 'Q' code items; a 'D' code item is a direct file-pointer, and has the base FID, modulo and separation of the file in attributes 2, 3 and 4. A 'Q' code item is a synonym pointer to a file defined in any account in the SYSTEM dictionary. This subroutine also performs the file access-protection checks. It is assumed that register LOCKSR points to the user's lock codes (in his logon entry in the SYSTEM dictionary; if the file has a lock code, a matching lock code is required for GBMS to return successfully. A non-match causes an exit to WRAPUP with message 210.

Input Interface

DAF1	B	If zero, uses retrieval lock-codes in LOGON entry for lock-code comparison; if set uses update lock-codes.
IR	R	Points to, or one prior, 'D' or 'Q' code in attribute one of file-definition item.
SR4	S	Points to AM at end of file-definition item.
LOCKSR	S	Points one prior to the user's lock-code field in his SYSTEM dictionary entry.

Output Interface

RMBIT	B	Set if base, modulo, separation successfully converted; zero if error in format, 'Q' item not found, etc.
BASE MODULO SEPAR	D T T	Contain base FID, modulo and separation of the file (if RMBIT set)
IR	R	Points to AM following attribute four of the file-definition item.

Subroutine Usage

CVDR15; recursive call to GBMS and RETIX if 'Q' code item; two further levels of subroutine linkage if 'D' Code; three if 'Q' code item.

Errors

On failing the lock code comparison test, an exit is taken to WRAPUP with message 210 (FILE IS ACCESS PROTECTED); to ensure termination of the curent process, RMODE is zeroed, PQFLG is set off and the history string set null before the exit.

5.4.5 GDLID

Functional Description

This subroutine gets the base, modulo and separation parameters from the DL/ID item in a dictionary. Typically this routine is called immediately after the dictionary base, modulo and separation have been obtained by GBMS.

GDLID retrieves the DL/ID item from the dictionary, and then enters GBMS to pick up its base modulo and separation.

Input Interface

BASE	D	Contains b, m, s of a file, containing the DL/ID item.
MODULO	T	
SEPAR	T	

Output Interface

RMBIT	B	Set if DL/ID found; zero otherwise.
BASE	D	Contains base FID, modulo and separation of the data-file (if RMBIT is set).
MODULO	T	
SEPAR	T	

Other elements as from GBMS and RETIX except that BMS/BMSBEG/BMSEND do not contain the DL/ID item-id.

5.5 TERMINAL I/O

5.5.1 GETIB AND GETIBX

Functional Description

GETIB and GETIBX are the standard terminal input routines. Register IBBEG points to a buffer area where the routine will input the data. Input continues to this area until either a carriage return or line feed is encountered, or until a number of characters equal to the count stored in IBSIZE have been input. The carriage return or line feed terminating the input line is overwritten with a segment mark (SM) and register IBEND points to this character on return. If the input is terminated because the maximum number of characters have been input, a SM will be added at the end of the line. This subroutine calls the subroutine GETBUF to read input data from the terminal. On return, GETIB then determines if the last character was a carriage return or a line feed which terminates the line, and causes a CR/LF echo to the terminal; if not, it either accepts or deletes the control character, depending upon the setting of the bit CCDEL, and calls GETBUF again.

The entry GETIB also provides the facility for taking the input from a stack instead of directly from the terminal. The bit STKFLG, if set, indicates that stacked input is present and register STKINP points to the area where the stacked input is stored. Input is copied from the stack area to the buffer, through the delimiter AM. The delimiter SM is used to signal the end of stack input. When this is encountered, STKFLG is turned off to indicate no more stacked input, and the routine then goes to the terminal for further input. The entry GETIBX does not test for stacked input. The stacked input feature is used primarily by the processor PROC to store input lines, which are returned to requesting processors as if they originated the terminal.

The stacked input routine in GETIB tests the last character of a stacked line for a "<"; if found, that character is overwritten with an SM. This allows the user to pass a line-continuation character to processors which recognize it.

Input Interface

IBBEG	S	Points one prior to buffer area where input is to be stored. Size of the buffer must be two characters greater than the value in IBSIZE.
IBSIZE	T	Contains maximum number of input characters to be accepted.
STKFLG	B	If set, indicates that "stacked" input may exist; if it does, terminal input will not be requested until the stack is exhausted.
STKINP	S	Points to next "stacked" input line; lines are delimited by AM's; a SM indicates end of stack.
PRMPC	C	Terminal "prompt" character; output before data is requested from the terminal.

LFDLY	T	Low-order byte contains the number of "fill" characters (nulls) to be issued after a carriage-return/line feed is output to the terminal.
CCDEL	B	If set, control characters are deleted from terminal input.

Output Interface

IB	R	Set to IBBEG.
IBEND	S	Points to SM one past last character input. (Overwrites CR or LF character).
STKFLG	B	Zeroed if end of stacked input reached.
STKINP	S	Points to next line of stacked input.

Subroutines Usage

GETBUF, PCRLF (both local); uses one additional level of subroutine linkage.

Error Conditions and Abnormal Exits

If the "stacked" input line exceeds IBSIZE, the line is truncated at IBSIZE; the remainder of the line is lost.

5.5.2 GETBUF

Functional Description

Inputs data from the terminal, and performs line editing functions. Returns control when a non-editing control character is input, or when the number of characters specified in T0 has been input.

Line editing features:

Character Input	Action
Control-H	Logically backspaces buffer pointer; echoes character defined at BSPCH.
Control-X	Logically deletes (or cancels) entire input buffer; echoes a carriage return, line feed, re-issues prompt character.
Control-R	Re-types input line.
Rubout	Ignored; the character is echoed, but is not stored in the buffer.
Control-shift-K Control-shift-L Control-shift-M Control-shift-N Control-shift-O	These characters are converted to the internal delimiters SB, SVM, VM, AM, and SM respectively; they echo as the characters [,/,], ,_

Note: The high-order bit of all characters input is zeroed.

Input Interface

R14	R	Points one prior to input buffer area.
T0	T	Maximum number of characters to be accepted.
PRMPC	C	Character output as a "prompt" when input is first requested.
BSPCH	C	Character echoed when a control-H is input.

Output Interface

R15	R	Points to control character causing return to caller.

5.5.3 WRTLIN AND WRITOB

Functional Description

Standard terminal output routines. Outputs data to the terminal or line-printer; controls pagination and page-heading routines. Entry WRTLIN adds a carriage-return/line-feed to the data; WRITOB does not. The data to be output starts at OBBEG, and continues through to the location addressed by OB.

The data is output to the terminal if LPBIT is off; it is stored in the printer spooling area if it is set. Pagination and page-heading are controlled by PAGINATE, if set. In this case, when the number of lines output in the current page (in LINCTR) exceeds the page size (in PAGSIZE), the following actions take place: 1) The number of lines specified in PAGSKIP are skipped; 2) the page number in PAGNUM is incremented, and 3) A new page heading is printed (see Section 5.5.5). A zero value in PAGSIZE suppresses pagination regardless of the setting of PAGINATE.

The "delay" character that is output at the end of a line is X'00'.

Editing Features

The internal delimiters SM, AM, VM, SVM, SB, are converted to the characters _, ,], /, [, respectively, if SMCONV is off; the output buffer area is blanked if NOBLNK is off. Trailing blanks are always deleted by the entry WRTLIN.

Carriage returns and line-feeds should not occur in the buffer if pagination is to be used.

Input Interface

OBBEG	S	Points one prior to the output data buffer.
OB	R	Points to the last character in the buffer; the buffer must extend two characters beyond this location.
LPBIT	B	If set, routes output to the spooler.
LISTFLAG	B	If set, suppresses all output to the terminal.
SMCONV	B	If set, suppresses conversion of internal delimiters.
NOBLNK	B	If set, suppresses blanking of output buffer.
PAGINATE	B	If set, pagination and page-headings are invoked.
PAGHEAD	S	Location of page-heading message.

LINCTR	T	Number of lines printed in current page.
PAGSIZE	T	Number of printable lines per page.
PAGSKIP	T	Number of lines to be skipped at bottom of page.
		(Above four elements used only if PAGINATE IS set.)
LINESOUT	T	Incremented on every entry.
LFDLY	T	Lower byte contains number of "fill" characters to be output after CR/LF.

Internal Usage

If LPBIT is set, the spooler routine PPUT will be called, and elements R8, T4, T5, D0, D1, RECORD D2, R14, R15, OVRFLW, SYSR0 may be destroyed.

Output Interface

OB	R	Reset to OBBEG

LINCTR, PAGNUM, LINESOUT: reset appropriately.

Subroutines Used

PPUT (printer spooler); PCRLF: NPAGE. Requires two additional levels of subroutine linkage.

Errors and Abnormal Exits

None

5.5.4 PCRLF

This subroutine may be used to output a CR/LF, with appropriate delays if necessary. Its use is not compatible with pagination.

Input Interface

LFDLY	See above.

5.5.5 PRNTHDR AND NEWPAGE

Page-heading control routine; PRNTHDR is used to initialize the pagination control elements; NEWPAGE is used to cause a skip to a new page, and to output a new page heading.

PRNTHDR sets the page number to one, the line counter to zero; sets the pagination flag, and outputs the first page heading. The page heading must be stored in a buffer defined by PAGHEAD; the header message is a string of data terminated by an SM; other system delimiters are used as below:

Delimiter	Action
SM,X'FF'	Terminates header line with a CR/LF.
AM,X'FE'	The current page-number is inserted in the heading.
VM,X'FD'	Prints header line, starts a new header line.
SVM,X'FC'	The current time and date are inserted in the heading.

Carriage-returns, line feeds and form-feeds should not be included in the header message.

Input Interface

LPBIT	B	If set, output is routed to the print spooler.
PAGHEAD	S	Points one prior to the beginning of the header message, which must be terminated by an SM.
LINCTR	T	Line number in current page.
PAGNUM	T	Current page number.
LFDLY	T	Lower byte contains number of "fill" characters to be output to terminal after a CR/LF. Upper byte: if non-zero, a form-feed (X'0C') character will be output before the start of a new page, and that number of "fill" characters will be output.
OBBEG	S	Points one prior to a buffer when the translated header message is built; this buffer area must be 16 bytes greater than the longest header line. (Not total header message size.)

Internal Usage

OB	R	Used to build the header message and to output it.

Output Interface

LINCTR, PAGNUM	Reset appropriately.

Subroutine Usage

WRITOB, TIMDATE	Uses two additional levels of subroutine linkage (three if time and date are inserted in header)

Errors

None

5.5.6 PRINT AND CRLFPRINT

Functional Description

Sends a message to the terminal from textual data in the calling program; used primarily for printing error messages. These subroutines are not compatible with output conventions to the lineprinter, and with the pagination routines. The message is a string of characters assembled immediately following the subroutine call in the calling program. The message must be terminated by one of the four delimiters SM, AM, VM, or SVM. Control is returned to the instruction at the location immediately following the terminal delimiter.

Delimiter	Action
SM,X'FF' AM,X'FE'	End of message; print carriage-return/line-feed before return.
VM,X'FD'	Print carriage-return/line-feed, continue.
SVM,X'FC'	End of message, no carriage return/line-feed.

Input Interface

Message follows the call to this routine.

LFDLY	T	Low-order byte contains number of "fill" characters after CR/LF is output.

Subroutine Usage

PCRLF; one additional level of subroutine linkage.

Errors

None

5.6 VIRTUAL MEMORY I/O

5.6.1 RDREC

Functional Description

RDREC is used to set up the registers IR, IRBEG, and IREND to the beginning and ending of the frame as defined by the tally RECORD. The subroutine assumes the frame has the linked format and therefore, IR and IRBEG are set pointing to the eleventh byte of the frame, that is, one prior to the first data byte of the frame. IREND is set up pointing to the last or 511th byte of the frame. Additionally the subroutine RDLINK is entered to set up R15 pointing to the link portion of the frame, and to set up the link elements NNCF, NPCF, FRMN, FRMP.

Input Interface

RECORD	D	Contains FID required.

Output Interface

IR	R	Points one prior to first data byte of frame.
IRBEG	S	As above.
IREND	S	Points to last data byte of frame.
R15	R	Points to zero-th byte of frame.
NNCF	H	Contains "nncf" field of frame.
FRMN	D	Contains forward link FID of frame.
FRMP	D	Contains backward link FID of frame.
NPCF	H	Contains "npcf" field of frame.

Subroutines Usage and Error Condition

None

5.6.2 RDLINK AND WTLINK

Functional Description

These routines read or write the link fields from or to a frame, to or from the tallies NNCF, FRMN, FRMP and NPCF. The FID of the frame is specified in RECORD.

Input/Output Interface

RECORD	D	FID of frame whose links are to be written or read.
NNCF	H	Number of next contiguous frames.
FRMN	D	Next or forward link FID.
FRMP	D	Previous or backward link FID.
NPCF	H	Number of previous contiguous frames.
R15	R	Points to zero-th byte of frame.

Subroutines Used and Error Conditions

None

5.6.3 LINK

Functional Description

Sets up the link fields of a group of unlinked contiguous frames. Up to 127 frames can be so linked. In each frame of the linked set, this subroutine sets up the number of next contiguous frames field, the next or forward link field, the previous or backward link field, and the number of previous contiguous frames field.

Input Interface

RECORD	D	Contains first FID of the group to be linked.
NNCF	H	Contains one less than the number of frames in the group (NNCF <127).

Output Interface

R14	R	Points one prior to the first data byte of first frame of linked set.
R15	R	Points to the last data byte of the last frame of linked set.

Subroutines Called and Error Conditions

None

5.7 OVERFLOW SPACE MANAGEMENT

5.7.1 GETOVF AND GETBLK

Functional Description

These routines obtain overflow frames from the overflow space pool maintained by the system. GETOVF is used to obtain a single frame; GETBLK is used to obtain a block of contiguous space (used mainly by the CREATEFILE processor). Note that the link fields of the frame(s) obtained by a call to GETBLK are not reset or initialized in any way; this is a function of the calling routine (also see Sections 5.7.3 and 5.7.4); GETOVF zeroes all the link fields of the frame it returns.

Input Interface

DO	D	(Accumulator) contains number of frames needed (block size), for GETBLK only.

Output Interface

OVRFLW	D	Contains FID of the frame obtained (GETOVF) or first FID of the block obtained (GETBLK).

Subroutines Used

SYSGET	One additional level of subroutine linkage required.

Error Conditions

Zero returned in OVRFLW if system overflow space is exhausted.

5.7.2 RELOVF, RELCHN AND RELBLK

Functional Description

These routines are used to release frame(s) to the overflow space pool. RELOVF is used to release a single frame; RELCHN is used to release a chain of linked frames (which may or may not be contiguous); RELBLK is used to release a block of continguous frames. A call to RELCHN specifies the first FID of a linked set of frames; the routine will release all frames in the chain until a zero forward link is encountered.

Input Interface

OVRFLW	D	Contains the FID of the frame to be released (RELOVF), or the first FID of the chain or block to be released.
D0	D	(Accumulator) contains the number of frames in the block to be released (block-size), for RELBLK only.

Output Interface

None

Subroutines Called

SYSREL (RELBLK only); one additional level of subroutine linkage required.

Errors

None

5.7.3 ATTOVF

Functional Description

ATTOVF is used to obtain a frame from the overflow space pool and to link it to the frame specified in RECORD. The forward link field of the frame specified in RECORD is set to point to the overflow frame obtained; the backward link field of the overflow frame is set to point to that in RECORD, and the other link fields of this frame are zeroed.

Input Interface

RECORD	D	Contains FID of the frame to which an overflow frame is to be linked.

Output Interface

OVRFLW	D	Contains FID of the frame obtained from overflow space.

Subroutines Used

GETOVF	Requires two additional levels of subroutine linkage.

Error Conditions

Zero returned in OVRFLW if system overflow space is exhausted.

5.7.4 NESTIR AND NEXTOVF

Functional Description

These routines obtain the foward linked frame of the frame to which the register IR currently points; if the forward link is zero, an available frame from the system overflow space pool is obtained and linked appropriately (see Section 5.7.3). In addition, the IR register triad is then set up before return, using the subroutine RDREC.

NEXTOVF may be used in a special way to automatically handle end-of-linked-frame conditions, on register six (IR), on single or multiple byte move on scan instructions. Set tally XMODE to the mode-id of the subroutine NEXTOVF before the move or scan instruction is executed; if the instruction causes register IR to reach an end-of-linked-frame condition (forward link zero), the system will generate a subroutine call to NEXTOVF, which in turn obtains and links up an available frame, and then resumes execution of the interrupted instruction. Note that the instruction "increment register by tally" cannot be so handled. Instructions compatible with NEXTOVF are: MITD, MTI and MCI.

Input Interface

IR	R	On last data byte of frame.

Output Interface

IR IRBEG	R S	Points to first data byte of forward linked frame.
IREND	S	Points to last data byte of frame.
RECORD	D	Contains FID of frame to which IR points.
FRMN, NNCF, FRMP, NPCF, R15		As set up by RDREC.

Subroutine Used

RDREC; ATTOVF; uses two additional levels of subroutine linkage.

Error Conditions

None

5.8 WORK SPACE INITIALIZATION

5.8.1 WSINIT

Functional Description

Initializes the process work-space pointer dyads: BMSBEG, BMSEND; CSBEG, CSEND; AFBEG, AFEND; IBBEG, IBEND; OBBEG, OBEND; PBUFEND, PBUFBEG. All work-spaces except the last are contained on one frame; PBUFBEG and PBUFEND define a 4-frame linked work-space.

Work-Space	Size (Bytes)
BMSBEG-BMSEND	50
AFBEG-AFEND	50
CSBEG-CSEND	100
IBBEG-IBEND	Contents of IBSIZE; max. 140
OBBEG-OBEND	Contents of OBSIZE; max. 140
PBUFBEG-PBUFEND	2000 (4 linked frames)

Input Interface

VOBIT	B	Set if linking of PBUF-space required.
IBSIZE	T	Size of IB buffer.
OBSIZE	T	Size of OB buffer.

Output Interface

Storage registers set up as in above table; associated address registers BMS, AF, CS, IB, OB, set at beginnings of respective buffers.

Subroutine Usage

LINK; one additional level of subroutine linkage is used.

5.8.2 TSINIT

This routine sets up the pointers to a one-frame scratch space; the pointers set up are TSBEG, TSEND, by TSINT. The associated address register, TS, is set at the beginning of the buffer.

5.8.3 ISINIT

Functional Description

This routine initializes all the system work-space pointers. The link-fields of linked work-spaces (IS, OS, HS, PBUF) are not initialized unless VOBIT is set. As the IS, OS and HS may have additional work-space assigned to them, calling ISINIT with VOBIT set will cause a loss of the additional work-space and a loss of system overflow space.

Output Interface

As for TSINIT and WSINIT above;

ISBEG, IS	Point to PCB + 16
ISEND	= ISBEG + 3000
OSBEG, OS	Point to PCB + 22
OSEND	=OSBEG + 3000
HSBEG	Points to PCB + 10
HSEND	=HSBEG

5.9 TAPE CONTROL ROUTINES

These routines provide for passing control commands to the magnetic tape unit. As in all tape commands, it is assumed that the tape unit is "attached" to the process executing these routines; this is flagged by the bit ATTACH being set. Conventionally, ATTACH should only be set by executing the verb T-ATT from the TCL level.

5.9.1 INIT AND TPSTAT

All tape I/O routines use the INIT and TPSTAT subroutines. INIT outputs a function-code of "1" to the tape controller, thereby setting it to an initial condition, and then falls through into the tape status from the controller. It will return only if the tape is in a "ready" state. If the tape is rewinding, the subroutine will wait till it finishes. Otherwise, the status is tested up to one hundred times; if the tape unit is still not ready, an exit is taken to MD99 with error message 95 (NOT ON-LINE).

Input Interface

None

Internal Usage

T6	T	Used as a delay counter

Output Interface

REJCTR	T	Zero

Tape status bits are as below:

EOFBIT	B	Set if an end-of-file mark is reached.
EOTBIT	B	Set if the tape is at load point, or at the end-of-tape marker.
PARITY	B	Set if a parity error is detected.
NORING	B	Set, on a write operation, if the write ring in the tape is not present.

5.9.2 WEOF

Writes and end-of-file mark on the tape.

5.9.3 BCKSP

Back spaces the tape by one record.

5.9.4 REWIND

Rewinds the tape unit.

5.9.5 FRWSP

Forward spaces the tape by one record; this subroutine destroys location X'1FF' in the PCB.

5.10 TAPE I/O ROUTINES

5.10.1 TPREAD AND TPWRITE

TPREAD reads one record from the tape to a buffer defined by R15; the read stops either when the inter-record gap in the tape is detected, or at the end of the frame to which R15 points.

TPWRITE writes one record from the buffer defined by R15 to the magnetic tape; the write will always continue until the end of the frame to which R15 points.

A maximum of 512 bytes may be transferred by these routines.

Input Interface

ATTACH	B	Must be set, indicating tape unit is attached.
R15	R	Points to first byte of buffer area.

Output Interface

R15	R	Points to last byte read (TPREAD). In the case of a read where the tape record is shorter than the buffer, R15 points one byte past the last data byte.

Tape status bits set appropriately.

Subroutines Used

INIT, TPSTAT (local); uses two additional levels of subroutines linkage.

Error Conditions

Read parity error: the read is repeated ten times; if the parity error persists, an exit is taken to MD99 with error number 98.

Write parity error: the write is re-tried once; then the sequence - backspace / write end-of-file mark / backspace and repeat write - is tried nine times; if the parity error persists, an exit is taken to MD99 with error number 98.

Also see INIT and TPSTAT.

5.10.2 ITPIB, TPIB, OBTP, AND FOBTP

The routines ITPIB, TPIB, OBTP and FOBTP allow reading and writing variable length records, blocked in fixed-length 500-byte records. These routines use the first two frames of the OS workspace to block and de-block; the unblocked data is passed to the write routine (OBTP) in the OB; it is passed from the read routine (IBTP) in the IB.

Reading a blocked tape: An initial call must be made to ITPIB to initialize the de-blocking pointers; subsequently, each call to TPIB will return one tape record.

Writing a blocked tape: The data to be written to the tape is placed in the OB, and OBTP is called to store it in the blocking area. When the output is to be terminated, one call to FOBTP must be made to clear the blocking area and force the data to be written to the tape.

These routines use the delimiter SB (X'FB') as the block delimiter; therefore, SB's in the data to be written to tape are converted to blanks before being output.

Note interface equivalence of these routines with the corresponding terminal I/O routine.

Input Interface

ITPIB	:	OSBEG	S	Points are prior to deblocking buffer.
TPIB	:	IBBEG	S	Points one prior to buffer area where deblocked data is to be copied.
OBTP	:	OBBEG	S	Points one prior to buffer area containing data to be blocked.
		OB	R	Points to last byte of data.
FOBTP	:	NONE		

Internal Usage (All Routines)

OSBEG	S	Points to a linked work-space used by the blocking and de-blocking routines. Must be at least 3 frames.
OS	R	Current pointer in blocked data area; must be maintained.
SC2	C	Scratch (OBTP, FOBTP only).

Output Interface

ITPIB : OS	R	Points to OSBEG; first two tape records have been read into the OS frames.
TPIB : IB	R	Points to IBBEG, one prior to de-blocked data.
IBEND	S	Points to a SM following last byte of data.
OBTP : OB	R	Reset to OBBEG
FOBTP : NONE		

Subroutines Called

TPREAD or TPWRITE; three additional levels of subroutine linkage.

Errors

See preceding documentation.

5.10.3 SEGMNT (3,TAPEIO-II)

This subroutine is used by the File-Restore, Sel-Restore, and Acc-Restore processors to de-block data from a File-Save tape. The built-up "segment" (data between segment marks on the tape) is stored in the IS.

Input Interface

S1	S	Points to location, within IB, of the last SM found. On initial entry, this is setup by the calling mode; it must be maintained between calls to SEGMNT.

Output Interface

ISBEG	S	Buffer where the segment has been copied.
IS	R	=ISBEG
XMODE	T	Zero

Subroutines Used

TPIB

5.11 LABELED TAPE I/O ROUTINES

Label format:

(SM)L...label data...(VM) time date (AM) reel # (AM) (SM)

The label is stored in the quadrenary control block (PCB + 3); displacements to various elements are as below:

Byte Displacement	Type	Description
1A5 (Bit Ø)	B	"unlabeled tapes in use" flag
1A6	T	Reel number
1A8	L	Label save buffer (46 bytes)
1D6	L	Label write/read buffer (30 bytes)

Since the tape-I/O routines are non-reentrant, internal storage is utilized when an EOT condition is handled by the tape write or read subroutines.

These routines save R13, R14, R15 in internal save areas (defined in TAPEIO-II), and set up R13 to displacement X'1A6' in the quadrenary control block in order to address elements in that block.

R13, R14 and R15 are restored on exit.

5.11.1 RDLABEL (2,TAPEIO-II)

May be called once by any program to read the label from reel #1; if the tape is labeled, the label is stored in the save area; if not, the "unlabeled tapes in use" flag is set. If the tape is not at the load point, no action is taken. No input interface. On output, the label save area is set up.

5.11.2 RDLABELX (5,TAPEIO-II)

As for RDLABEL, except that no check is made to see if the tape is at the load point. This routine is used by the FILE-RESTORE processor (ABSL), to read a label even though the tape may be positioned past the load point. May be used by user-programs, though the implications of doing so should be kept in mind.

5.11.3 WTLABEL (3,TAPEIO-III)

May be called once by any processor to write a label on reel #1; no action is taken if the tape is not at load point. The label (if any) passed as an input parameter is written to the tape, with the current time and date, and reel number one, added. The label is also stored in the label save buffer.

Input Interface:

IS	R	Points one before the label data, which must be terminated by any standard system delimiter. The label cannot be greater than 16 characters; it will be truncated to 16 if it is. If a null label is submitted, no label is written to the tape, and the "unlabeled tapes in use" flag is set.

Output Interface

IS	R	Points to delimiter terminating label, or to 16 bytes beyond the input position if none is found.

Label save area initialized.

5.11.4 WTLABELX (4,TAPEIO-III)

As for WTLABEL, except that no check is made to see if the tape is not set to load point. This routine is used by the FILE-SAVE processor (ABSD), to write a label at the current position of the tape. May be used by user-programs, though the implications of doing so should be kept in mind.

5.12 FILE-INITIALIZATION

5.12.1 DLINIT (6,DLOAD)

Functional Description

Obtains a block of contiguous overflow space for a file; links the frames and sets up initial conditions using the routine DLINIT1 (described below)

Input Interface

MODULO	T	Contains the modulo required for the file.
SEPAR	T	Contains the separation required for the file; if SEPAR is greater than 127, it will be reset to one.

Output Interface

BASE	D	Contains the beginning FID of a contiguous block of size MODULO*SEPAR. If BASE=0, the system does not have sufficient overflow space.

Note: This subroutine automatically enters DLINIT1 if the overflow space is obtained.

Subroutines Used

GETBLK; two additional levels of subroutine linkage.

5.12.2 DLINIT1 (7,DLOAD)

Functional Description

Initializes link fields of a file as specified by its base, modulo and separation parameters; sets each group empty by adding an AM at the beginning.

Input Interface

BASE	D	Contains base-FID, modulo and separation of file.
MODULO	T	
SEPAR	T	

Internal Usage

CTR1	T	
RECORD	D	

Output Interface

R14	R	As returned by subroutine LINK
R15	R	

Subroutines Used

LINK; RDREC; two additional levels of linkage required.

System Accessing Routines

These subroutines may be used to setup pointers to system-files (SYSTEM,ACC), to the PIB's, etc.

5.12.3 GPCB0 (4,ABSL)

Functional Description

Returns the PCB-FID for channel zero in the accumulator.

Input Interfaces

None

Output Interfaces

D0	D	Contains FID of PCB for channel zero. High order 16 bits are zero.

5.12.4 SETPIB (4,LOGON)

Functional Description

Sets up R14 to point to the first byte of the PIB associated with the process. No input interface.

Internal Usage

SR5	S	Scratch

Output Interface

R14	R	As described.

5.12.5 SETPIBF (3,ABSL)

Functional Description

Sets up R14 to point to the first byte of the PIB associated with channel zero. No input interface.

Output Interface

R14	R	As described.

5.12.6 GMMBMS

Functional Description

Sets up pointers to the SYSTEM dictionary (formerly called MM/DICT). No Input Interface.

Output Interface

BASE MODULO SEPAR	D T T	Contains base-FID, modulo and separation of SYSTEM dictionary.

5.12.7 GACBMS (1,LOGOFF)

Functional Description

Sets up pointers to the ACC dictionary. No Input Interface.

Internal Usage

SR1	S	Scratch
T6	T	Scratch

All elements as used by GBMS

Output Interface

BASE MODULO SEPAR	D T T	Contains base-FID, modulo and separation of the ACC dictionary (actually a file, since ACC as defined in SYSTEM is a Q-entry to the DL/ID of ACCOUNT)
REJ1	T	Contains the value 331 if the ACC file is missing.

Subroutines Called

GMMBMS; GBMS; two additional levels of linkage required.

5.12.8 GETOPT (10,SYSTEM-SUBS-II)

Functional Description

This program converts an option string consisting of single alphabetic characters or a numeric specification. Alphabets set the corresponding bit (A sets ABIT, etc.). Multiple options are separated by commas, and the string must be terminated by a ")". ABIT through ZBIT are not zeroed on entry.

Input Interface

IS	R	Points one before the option string.

Output Interface

ABIT through ZBIT		Set as described above.
NOBIT	B	Set if numeric option is found; zero otherwise.
RMBIT	B	Set
D4	D	Contains numeric values if numeric options is found; unchanged otherwise. D4 contains first numeric, D5 second if found, otherwise the same value as D4.

Error Conditions

Exits to MD99 with error 209 after setting RMODE zero if a format error is encountered.

5.12.9 GETUPD

Functional Description

Sets up the register triad UPDBEG,UPD and UPDEND, in unlinked format, to PCB+28. A convenient way to setup a register to a buffer (also used by RPG). Note that the UPD registers are treated as a scratch register by some system subroutines. No Input Interface.

Output Interface

UPBEG	S	Points to byte zero of PCB+28
UPD	R	=UPDBEG
UPDEND	S	Points to last byte of PCB+28.

5.12.10 XISOS

Exchanges the register triads ISBEG/IS/ISEND and OSBEG/OS/OSEND.

5.12.11 PRIVTST1 (5,SYSTEM-SUBS-III)

Tests if the process has system privileges, level one; exits to MD99 with error 82 after setting RMODE zero, clearing PQFLG and LISTFLAG, and setting the history string null if not.

5.12.12 PRIVTST2 (7,SYSTEM-SUBS-III)

As above, for system privileges, level two.

5.13 MISCELLANEOUS ROUTINES

Some of these subroutines are primarily used by system processors, and, therefore may use elements other than the minimum set used by the general-purpose system subroutines.

5.13.1 TIMDATE, TIME, AND DATE

Functional Description

These routines obtain the system time and/or the system date, and store it in the buffer area specified by R15. The time is returned as on a 24-hour clock.

Entry	Buffer Size Required	Format
TIME	8	HH:MM:SS
DATE	11	DD MMM YYYY
TIMDATE	21	HH:MM:SS DD MMM YYY

Input Interface

R15	R	Points one prior to the buffer area.

Output Interface

R15	R	Points to last byte of data stored.

Subroutines Used

Entry TIMDATE uses TIME; requires two additional levels of subroutine linkage; other entries require one level.

Errors

None

5.13.2 ASCII TO BINARY CONVERSION

Functional Description

The routines described below will convert a string of ASCII decimal or hexadecimal characters to their binary equivalent; the conversion continues until an illegal (non-decimal or non-hexadecimal) character is encountered.

On entry, the appropriate register (see table) points either to a non-numeric character, one prior, or to the first character of the string, which must be a plus sign, a minus sign or an appropriate numeric (0-9 for the decimal routines, 0-9 and A-F for the hexadecimal routines). On return, the converted binary number is in the accumulator (and in some cases, in CTR1); the register points to the illegal character causing the conversion to terminate. Note that the register will always be incremented by one even in the case of a null string (no legal characters). Arithmetic overflow due to too many digits in the character string cannot be detected.

The routines CVDR15 and CVXR15 will test for a "+", a "-", or an appropriate numeric character at R15 on entry; if none of these, R15 is incremented and the tests repeated. Thus, it is not necessary to call either of these two routines with R15 pointing to a minus sign.

Entry Name	Register Used	Conversion From: Dec.	Hex.	Value Returned IN: Accumulator	CTR1
CVDR15	R15	X		X	
CVXR15	R15		X	X	
CVTNIS	IS (R4)	X		X	X
CVTHIS	IS (R4)		X	X	X
CVTNOS	OS (R5)	X		X	X
CVTHOS	OS (R5)		X	X	X
CVTNIR	IR (R6)	X		X	X
CVTHIR	IR (R6)		X	X	X
CVTNIB	IB (R10)	X		X	X
CVTHIB	IB (R10)		X	X	X

Subroutine Used

CVDR15 or CVXR15 are called by the other routines; one additional level of linkage is required.

5.13.3 BINARY TO ASCII CONVERSION (MBDSUB AND MBDNSUB)

Functional Description

These routines will convert a binary number to the equivalent string of decimal ASCII characters. The conversion will store a minimum number of characters (that is, leading zeroes will be padded if needed) if the entry MBDNSUB is used; if MBDSUB is used, only as many characters as are needed to represent the number will be stored. A minus sign will precede the character string if the number to be converted is negative.

These subroutines are implicitly called by the assembler instructions MBD (move binary to decimal) and MBDN.

Input Interface

D0	D	(Accumulator) Contains number to be converted.
R15	R	Points one prior to buffer where converted characters are to be stored (maximum 9 characters).
T4	T	(MBDNSUB entry only) Contains minimum number of characters to be stored.

Output Interface

R15	R	Points to last converted character.

Subroutines Used and Error Conditions:

None.

5.13.4 EBCDIC TO ASCII CONVERSION (ECONV, R.ETA.M, AND R.ATE.M)

EBCDIC to ASCII conversion - ECONV

The register IB points to the EBCDIC character; a call to ECONV converts it to ASCII; characters that cannot be converted are returned as a question mark (?).

String EBCDIC to ASCII with move - R.ETA.M

The registers R15 and R8 point to the first character of the source and destination strings, respectively. CTR1 contains the character count. R13, R14, and T0 are destroyed. All characters are converted. CTR1 should be zero on return. R15 and R8 point to the last character of their respective strings on return.

String ASCII to EBCDIC with move - R.ATE.M

Same as R.ETA.M except ASCII to EBCDIC translation.

5.13.5 CREAD

Functional Description

The subroutine either reads a card and returns the card reader status after the read or it just returns the status if it cannot read a card. Cards are read in EBCDIC and are not converted by this routine.

Input Interface

R2	R	Must point to a scratch byte. (Typically R2 will always point to byte zero of the SCB.)
OBBEG	S	Points anywhere within the frame that the card is to be read into. (Typically OBBEG will always point within PCB+4).

Internal Usage

T3	T	Used as a counter for status timeout after a read.

Output Interface

R2	R	Unchanged. The byte that R2 points to contains the status of the card reader.
CBIT	B	Zero if no card was read. Set if an attempt to read a card was made.
R15	R	Points to first byte of card read, 80 bytes from the end of the frame that OBBEG points to.

Errors

None, except card reader errors returned as status. The status bits are as follows:

Bit	Explanation of the set condition.
0-2	Unused by the controller. Will be zero.
3	Card reader mechanical error (e.g., pick failure, card motion error, etc.)
4	EBCDIC error detected. (e.g., an invalid punch combination was detected.) Not an error if CBIT is zero.

5	Input hopper empty. Not an error if CBIT is set.
6	This bit is always zeroed by the routine. It is only used for byte I/O.
7	Card reader ready.
3,5,7	If bits 3, 5, and 7 are all set, this indicates that power is off on the card reader.

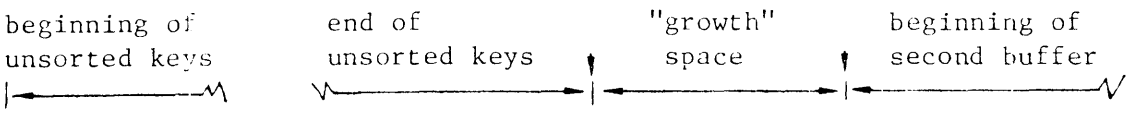
5.13.6 SORT

Functional Description

Sorts an arbitrarily long string of keys in ascending sequence only; the calling program must complement the keys if a descending sort is required. The keys are separated by SM's when presented to SORT; they are returned separated by SB's. Any character, including system delimiters other than the SM and SB may be present within the keys.

An n-way polyphase sort-merge sorting algorithm is used. The original unsorted key string may "grow" by a factor of 10%; and a separate buffer is required for the sorted key string, which is about the same length as the unsorted key string. The "growth" space is contiguous to the end of the original key string; the second buffer may be specified anywhere. The SORT subroutine will automatically obtain additional overflow space and link it if needed.

Due to this, one can follow standard system convention and build the entire unsorted string in an overflow table with OVRFLCTR containing the beginning FID; the setup is then:



The second buffer pointer then is merely set at the end of the "growth" space, and SORT allowed to obtain additional space as required.

Alternately, the entire set of buffers may be in the IS or OS work-space if they are large enough.

Input Interface

SR1	S	Points to SM preceeding first key.
SR2	S	Points to SM terminating last key.
SR3	S	Points to beginning of second buffer area.

Internal Usage

Entire BMS work area.

S1 through S9: Scratch

BMS	R
CS	R
IS	R
OS	R
TS	R
HBIT	B
LBIT	B
SB1	B

Output Interface

SR1	S	Points two bytes before SB preceding first sorted key. The sorted keys are delimited by SB's, and the entire string terminated by an SM.

Subroutine Usage

Internal call to COMP; ATTOVF if end of second buffer is reached. One additional level of linkage required.

5.13.7 BLOCK-LETTERS

Functional Description

This program provides the block letter capability. In addition to its use at the verb level, it may be called as a subroutine (DEFM 2,290). It will format a string of words on the terminal or the printer and return to the caller.

Input Interface

ZBIT	If set, direct output to the terminal; if not set, direct output to the printer
IS	Points one character prior to the first character to be output; end of data is indicated by the character pair SM,Z, if a segment mark is present in the string not followed by a "Z" the string must be terminated by a start buffer (SB) X'FB' (see TCL-I interface).
PAGSIZE	Maximum number of lines per page.
OBSIZE	Maximum number of characters on each output line.

Internal Usage

The following functional elements are used and not restored.

SC2, SCL REJCTR, C1, PAGINATE, BASE, MODULO, SEPAR, CTR16, CTR17, CTR18, SR4, SR5, SR6, SR7, SR8, SR9, SR10, SR11, SR12, SR13, SR14, SR15, SR16, SR17, SR18, SR19, SR20, SR21, SR22, AFEND, LPBIT, ZBIT

Subroutines Called:

RETIX, GBMS, NEWPAGE, CVTNIR, WRTLIN

Error Conditions:

If a BLOCK-TERM or BLOCK-PRINT error is detected, the program exits to WRAPUP without resetting RMODE or VOBIT. See verb write-up on BLOCK-TERM and BLOCK-PRINT in Reality Programmer's Reference Manual.

5.14 ENGLISH AND BATCH INTERFACES

5.14.1 ENGLISH INTERFACE

It is possible to interface with the ENGLISH processor at several levels. A typical LIST on SORT statement passes through the Pre-processor and Selection Processor before entering the LIST processor. All statements must pass through the first two stages; but control can be transferred to user-programs from that point onward.

5.14.2 GENERAL CONVENTIONS

The ENGLISH processors use a compiled string that is stored in the IS work-space. String elements are separated by segment marks; there is one element for each attribute specified in the original statement; one file-defining element, and special elements pertaining to selection criteria, sort-keys, etc.

Formats:

File-Defining Element; at ISBEG+1

(SM)D file-name (AM) base (VM) modulo (VM) separ (AM) conv. (AM) correl. (AM) type (AM) just. (AM) (SM).

Attribute-Defining Element

(SM) c attribute-name (AM) amc (AM) conv. (AM) correl. (AM) type (AM) just. (AM) (SM)

c = A - regular or D2 attribute.

Q - D1 attribute.

BX - SORT-BY, SORT-BY-DSND, etc. "X" is from attribute one of the connective.

End-of-String Element

(SM)Z

Explicit Item-id's

(SM)I item-id (SM)

5.14.3 THE SELECTION PROCESSOR

This performs the actual retrieval of items which pass the selection criteria, if specified. Every time an item is retrieved, the processor

at the next level is entered with RMBIT set; a final entry with RMBIT zero is also made after all items have been retrieved. If a sorted retrieval is required, the Selection processor passes items to the GOSORT mode, which builds up the sort-keys preparatory to sorting them. After sorting, GOSORT then retrieves the items again, in the requested sorted sequence.

A user-program may get control directly from the selection processor (or GOSORT if a sorted retrieval is required); the formats of the verbs are:

Line Number	Non-Sorted	Sorted
001	PA	PA
002	35	35
003	xxxx	76
004		xxxx

where "xxxx" represents the mode-id of the user-program. Note that, in this method of interface, only item retrieval has taken place; none of the conversion and correlative processing has been done. For functional element interface, the column headed "Selection Processor" in the table shown later must be used.

Exit convention: On all but the last entry, exit indirectly via RMODE (using ENT* RMODE); on the last entry, exit to one of the WRAPUP entry points. Processing may be aborted at any time by setting RMODE zero and entering WRAPUP. SBO must be set on first entry.

5.14.4 SPECIAL EXIT FROM THE LIST PROCESSOR

A user-program may also gain control in the place of the normal LIST formatter, to perform special formatting. The advantage here is that all conversions, correlatives, etc. have been processed, and the resultant output data has been stored in the history string (HS area). The formats of the verbs that are:

Line Number	Non-Sorted	Sorted
001	PA	PA
002	35	35
003	4D	4E
004	xxxx	xxxx

Where "xxxx" is the mode-id of the user-program.

History-String Format: The output data is stored in HS area; data from each attribute specified is stored in the string, delimited by AM's; multiple values and sub-multiple values are delimited within by VM's and SVM's respectively. Since the HS may contain data other than the retrieved item, the user-program should scan from HSBEG, looking for a segment preceded by an "X"; all segments except the first are preceded by an SM.

X item-id (AM) value one (AM)...(AM) value n(AM)(SM)Z

The program must reset the history string pointer HSEND, as items are taken out of the string. In special cases, data may not be used till, say, four items are retrieved, in which case HSEND is reset on every fourth entry only. HSEND must be reset to point one byte before the next available spot in the HS; normally one before the first "X" code found.

Exit convention: see preceding section.

Example: The following program is an example of one which prints item id's (only) four-at-a-time across the page.

```
001             FRAME   504                 INTERNAL FLAG
002             ZB      SB30                INTERNAL FLAG
003             BBS     SB0,NOTF            NOT FIRST TIME
004  * FIRST TIME SETUP
005             MOV     4,CTR32
006             SB      SB0
007  *
008  NOTF       BBZ     RMBIT,PRINTIT       LAST ENTRY
009             BDNZ    CTR32,RETURN        NOT YET 4 ITEMS OBTAINED
010             MOV     4,CTR32             RESET
011  PRINTIT    MOV     HSBEG,R14
012  LOOP       INC     R14
013             BCE     C'X',R14,STOREIT    FOUND AN ITEM
014             BCE     C'Z',R14,ENDHS      END OF HS STRING
015  SCANSM     SCD     R14,X'C0'           SCAN TO NEXT (SM)
016             B       LOOP
017  STOREIT    BBS     SB30,COPYIT         NO FIRST ID FOUND
018             SB      SB30                FLAG FIRST ID FOUND
019             MOV     R14,SR28            SAVE LOCATION OF FIRST "X"
020  COPYIT     MIID    R14,OB,X'A0'        COPY ITEM-ID TO OB
021             MCC     C' ',OB             OVERWRITE (AM)
022             INC     OB,5                INDEX
023             B       SCANSM
024  ENDHS      BSL     WRTLIN              PRINT A LINE
025             MOV     SR28,HSEND          RESTORE HSEND TO FIRST "X" CODE
026             DEC     HSEND               BACKUP ONE BYTE
027             BBZ     RMBIT,QUIT
028  RETURN     ENT*    RMODE               RETURN TO SELECTION PROCESSOR
029  QUIT       ENT     MD999               TERMINATE PROCESSING
030             END
```

5.14.5 FUNCTIONAL ELEMENT USAGE

The following table summarizes the functional element usage by the Selection and LIST processors. Only the most important usage is described; elements that have various usages are labeled "scratch". A " " indicates that the processor does not use the element. Since the LIST processor is called by the Selection processor, any element used for "memory" purposes (not to be used by others) in the former is indicated by a blank usage in the latter column.

In general, user-programs may freely use the following elements:

```
Bits   :   SB20 upwards
Tallys :   CRT30 upwards ; D3-D8
S/R's  :   SR20 upwards
```

SB0 and SB1 have a special connotation; they are zeroed by the Selection processor when it is first entered, and not altered thereafter. They are conventionally used as first-time switches for the next two levels of processing. SB0 is set by the LIST processor when it is first entered, and user-programs that gain control directly from Selection should do the same. SB0 may be used as a first-entry switch by user-programs that gain control from the LIST processor.

An ENGLISH verb is considered an "update" type of verb of the SCP character (from line one of the verb definition) is A, B, C, D, E, G, H, I or J. SCP characters of B, C, D and E are reserved for future ENGLISH update verbs.

BITS	Selection Processor	LIST Processor
ABIT	unused	non-columnar list
BBIT	first entry flag	
CBIT	scratch	scratch
DBIT	scratch	dummy control-break
EBIT	reserved	control-break flag
FBIT	reserved	scratch
GBIT	reserved	scratch
HBIT	reserved	scratch
IBIT	explicit item-ids specified	
JBIT	reserved	D2 attribute in process
KBIT	scratch	scratch
LBIT	scratch	left-justified field
MBIT	conversion interface; zero	zero
NBIT	scratch	scratch
OBIT	selection test on item-id	
PBIT	scratch	scratch
QBIT	scratch	scratch
RBIT	full-file retrieval flag	
SBIT	selection on values (WITH)	
TBIT	scratch	print limiter flag

BITS	Selection Processor	LIST Processor
UBIT	scratch	reserved
VBIT	reserved	scratch
WBIT	scratch	reserved
XBIT	scratch	reserved
YBIT	left-justified value being tested	left-justified print-limiter test
ZBIT	left-justified item-id	
SB0	Unavailable	first entry flag, level one
SB1	Unavailable	first entry flag, level two
SB2	reserved; zero	
SB4 through SB16	scratch or reserved	scratch or reserved
VOBIT	set for WRAPUP interface	
COLHDRSUPP DBLSPC HDRSUPP IDSUPP LPBIT CBBIT PAGINATE	set if corresponding connective was found in input statement	
RMBIT	set on exit if an item was retrieved; zero on final exit.	
SMBIT	FUNC interface	FUNC interface
GMBIT	FUNC interface	FUNC interface
BKBIT	scratch	scratch
DAF1	set if SCP = B,C,D,E,G, H,I or J	
DAF8	set if accessing a dictionary	

Tallys	Selection Processor	LIST Processor
C1;C3-C9	Scratch	Scratch
C2	contents of MODEID2	
CTR1-CTR4	Scratch	Scratch
CTR5	Scratch	AMC of current element in IS
CTR6	reserved	Scratch
CTR7	reserved	AMC corresponding to IR
CTR8	reserved	Scratch
CTR9	reserved	Scratch
CTR10	reserved	Scratch

Tallys	Selection Processor	LIST Processor
CTR11	reserved	Scratch
CTR12	FUNC interface	current-sub-value count
CTR13	FUNC interface	current value count
CTR14	reserved	Scratch
CTR15	reserved	Scratch
CTR16	reserved	Contents of item count file
CTR17	reserved	reserved
CTR18	reserved	Scratch
CTR19	reserved	Scratch
CTR20	CONV interface	CONV interface
CTR21	CONV interface	CONV interface
CTR22	CONV interface	CONV interface
CTR23	CONV interface	CONV interface
CTR24	reserved	Scratch
CTR25	reserved	Scratch
CTR26	reserved	Scratch
CTR27	reserved	current max-length
CTR28	reserved	Scratch
D9	count of retrieved items	
FP1-FP3,D7	FUNC interface	FUNC interface
RMODE	return mode-id (MD30)	
SIZE	item-size	Scratch
SBASE SMOD SSEP	b,m,s of file	
DBASE DMOD DSEP	b,m,s of dictionary	

S/R's	Selection Processor	LIST Processor
S1	Points to next explicit item-id	
S2-S9	Scratch	Scratch
SR0	Points one before count field of item	
SR1	Points to correlative field	Current correlative field
SR2	Scratch	Scratch
SR3	reserved	Scratch
SR4	Points to last AM of item	
SR5	reserved	Next segment in IS
SR6	Points to conversion field	Current conversion field
SR7	reserved	Scratch

S/R's	Selection Processor	LIST Processor
SR8	reserved	reserved
SR9	reserved	reserved
SR10	reserved	Scratch
SR11	reserved	reserved
SR12	reserved	reserved
SR13	GOSORT only: next sort-key	reserved
SR14	reserved	reserved
SR15	reserved	reserved
SR16	reserved	reserved
SR17	reserved	reserved
SR18	reserved	reserved
SR19	reserved	reserved
PAGHEAD	Heading in HS if HEADING was specified	generated heading in HS
Workspace pointers	See section on work-space usage	

A/R's	Selection Processor	LIST Processor
AF	Scratch	Scratch
BMS	within BMS area	Scratch
CS		Scratch
IB		Scratch
OB		Output data line
IS	compiled string	compiled string
OS		Scratch
TS	within TS area	within TS area
UPD		within HS area
IR	within item	within item

Work-Space Usage	Selection Processor	LIST Processor
AF	Scratch	
BMS	Contains item-id	
CS		
IB		
OB		Output line.
IS	Compiled string	
OS		Scratch

Work-Space Usage	Selection Processor	LIST Processor
HS	Heading data	Heading data, attribute data for special exits.
TS	Scratch	Current value in process.

Additional Notes

1. If a full-file retrieval is specified, the additional internal elements as used by GETITM will be used. If explicit item-id's are specified, RETIX is used for retrieval of each item.

2. Elements as used by the FUNC and CONV processors have been shown in the table; both may be called either by the Selection processor or the LIST processor.

3. Since the ISTAT and SUM/STAT processors are independently driven by the Selection processor, the element usage of these processors are not shown.

4. The section of the IS and OS used by the Selection and LIST processors is delimited by ISEND and OSEND respectively. The buffer space beyond these pointers is available for use by other programs.

5.14.6 BATCH PROCESSOR INTERFACE

The BATCH processor uses a BATCH- string which defines the method of updating one or more items in one or more files using a single line of input data. The updated item(s) is(are) built as disc-update string(s) in the history string area (see WRAPUP for format).

A user-exit can be defined in the BATCH-string; the functional elements used by BATCH are described in the following tables; the column headed "Level" has the following entries:

- 0 - Element is used in the described fashion throughout the BATCH processing.
- F - Element is redefined every time a file-defining element is found.
- A - The element is redefined for every attribute.
- blank - Scratch element, on reserved for future usage.

As far as user-exit programs are concerned, therefore, all elements defined at the "A" level can also be considered scratch.

Exit Convention: The user-exit must return to the BATCH processor by executing the external transfer to the mode BATCH5 (DEFM 0,84).

Bits	Level	Description
ABIT	0	First-time switch for BATCH process.
BBIT		reserved
CBIT		Scratch
DBIT	A	D2 attribute in process.
EBIT	F	Updates to be merged with item on file.
FBIT	0	Set when a BV or BC Arb-element is found.
GBIT		reserved
HBIT	A	D1 attribute in process.
IBIT	0	Set when a "secondary" file.
JBIT		reserved
KBIT	F	Item to be verified as existing on file.
LBIT	F	Item to be verified as not existing on file.
MBIT	A	Set; CONV interface.
NBIT		reserved
OBIT		reserved
PBIT	F	Flag indicating that a multi-valued field referenced by BC/BV element.
QBIT		reserved
RBIT		reserved
SBIT		scratch
TBIT		scratch
UBIT	0	Item is being deleted (X element in file-definition)
VBIT		scratch
WBIT		scratch
XBIT		reserved
YBIT	0	Primary item being deleted.
ZBIT		scratch
SB1 through SB9		scratch
DAF10	0	Set if SELECT/SSELECT is driving BATCH.

Tallys	Level	Description
C1		scratch
C2		scratch
C3-C9		reserved
CTR1		scratch
CTR2		scratch
CTR3		scratch
CTR4	A	D1-D2 set number (follows D1 or D2 element)
CTR5		reserved
CTR6		reserved
CTR7		scratch
CTR8	F	Current amc in process.
CTR9		reserved
CTR10		reserved
CTR11	F	Value no. of "D1;1" attribute; 0 if unspecified.
CTR12	F	Value no. of "D1;2" attribute; 0 if unspecified.
CTR13	F	Value no. of "D1;3" attribute; 0 of unspecified.
CTR14-CTR19		reserved
FP1-FP3		scratch
BASE		
MODULO		scratch
SEPAR		scratch
SBASE		scratch
SMOD		scratch
SSEP		scratch
D7		scratch
D9		scratch
RMODE	0	Return mode-id for WRAPUP

Work-Spaces & A/R's	Level	Description
BMS	A	Work-space contains current value
CS		Scratch; work-space reserved.
AF		Unused
IB	0	Input data line
OB		Unused
TS	0	Used for reading input lines.
IS	0	Contains BATCH string; IS points to AM before next element.
OS		Scratch work-space
UPD	0	Points to history-string

S/R's	Level	Description
S1-S9		Scratch
SR0	0	One before count field of primary item on file.
SR1	0	End of primary item on file
SR2		scratch
SR3		reserved
SR4	F	End of current item on file.
SR5		reserved
SR6		reserved
SR7	0	End of OS deletion table
SR8		reserved
SR9	A	Last byte of value in BMS area.
SR10	F	
SR11	0	End of primary update string if FBIT set.
SR12	0	Points one before "DU" of history string for primary item update.
SR13		reserved
SR14	F	Location of file-defining element in IS
SR15	F	Location of IB when current file-defining element was found.
SR16-SR19		reserved.

Characters	Level	Description
SCP	0	Contains a "D" for B/DEL; "A" for B/ADD
SC0	0	Contains a blank.
SC1	0	Scratch
SC2	0	Contains a comma.

Also note elements used by CONV processor.

5.14.7 CONVERSION PROCESSOR INTERFACE

The Conversion processor is called as a subroutine (CONV DEFM 0, 90) and may be used to perform the Translate, Date, or Mask Conversions. More than one conversion can be performed at once if the conversion string is set up to do so; multiple conversion codes are separated by VM's. Conversion is called by the ENGLISH pre-processor to perform conversions on "input" data (in selection criteria), and by the LIST/SORT processor to perform "output" conversion.

Input Interface

MBIT	B	Set if an "input" conversion is to be performed; zero for an "output" conversion.
TSBEG	S	Points one before the value to be converted; the value is converted "in place", and the buffer is used for scratch spare; therefore it must be large enough to contain the converted value. The value to be converted is terminated by any of the standard system delimiters; SM, AM, VM, SVM.
IS	R	Points to the first character of the conversion code specification string; the code(s) must be terminated by an AM.
BMSBEG	S	Used for item-id copy on Translate conversions.

Internal Usage

SB10	B
SB11	B
SB12	B
SC2	C
CTR20	T
CTR21	T
CTR22	T
CTR23	T

S4	S	Scratch; used to save and restore various elements.
S5	S	
S6	S	
S7	S	

Output Interface

IS	R	Points to AM terminating the conversion code(s).
TSBEG	S	Points one before converted value.
TS	R	Points to the last character of the converted
TSEND	S	value; a SM is also placed one past the value. If a null value is returned, TS=TSEND=TSBEG.

If a Translate conversion is used, subroutines GBMS, GDLID, and RETIX are used. Thus all elements used by those subroutines will be destroyed, with the exception of IR, SR4 and SIZE, which are restored before exit to their values or entry.

Subroutines Used: GBMS, GDLID, RETIX (T-conversion only); MBDSUB, CVDR15

Error Exits:

CONV will exit to WRAPUP after setting RMODE zero under the following conditions:

705	Illegal conversion code
706	Illegal T-conversion: format incorrect, filename cannot be found, etc.
707	DL/ID cannot be found for T-conversion file.

WRAPUP is also entered, without setting RMODE zero, under the following error conditions:

708	Value cannot be converted by the T-conversion.
339	Invalid format for input data conversion.

5.14.8 USER CONVERSION PROCESSING

The Conversion Processor will pass control to a user-written routine if a "Uxxxx" code is found in the conversion string, where "xxxx" is the hexadecimal mode-ID of the user-routine. This routine can then perform special conversion before returning. The input interface at the user-routine will be identical to that described in the preceding section; after performing the conversion the user-routine should setup the

output interface elements (see ENGLISH manual), and exit via an external branch to 1,CONV, which will continue the conversion process if multiple conversions are specified; a RTN may be executed if this is not needed, or to prevent further conversions being performed. Elements used by the regular conversion processors may be safely used by user-routines; however, if additional elements are needed, a complete knowledge of the processor that called CONVERSION (LIST, SELECTION, etc.) will be necessary.

5.14.9 FUNCTION PROCESSOR INTERFACE

The FUNCTION processor is used by the ENGLISH LIST/SORT processors to compute values which have an "F" correlative specified. It may also be called, as a subroutine, by user routines. Each call to FUNC (DEFM 0,101) returns one value.

Input Interface

SR1	S	Points to the "F" of the Function string.
SR0	S	Points one before the count-field of the item.
SR4	S	Points to the last AM of the item.
CTR13	T	Contains the "value number" currently being processed (one on initial entry).
CTR12	T	Contains the "sub-value number" (D2 sub-value) currently being processed.
TSBEG	S	Points to a buffer area-350 where the value is to be stored on exit.

Internal Usage

SMBIT	B
GMBIT	B
CTR1	T
CTR12	T
CTR20	T
CTR21	T
IR	R
IS	R
D7	D
FP1-FP3	D

Output Interface

IR	R	Points one before the value copied (TSBEG+350); the value is delimited by an AM if none of the referenced fields contained multiple or sub-multiple values; by a VM if at least one of the referenced fields contained a VM on this entry by A SVM if at least one of the referenced fields contained a SVM on this entry.
R15	R	Points to a blank following the terminal delimiter of the value.
IS	R	Points to the AM or one past a VM, terminating the Function string.
D0,FP1	D	Contains final computed result.

Programming Note

On the first call to FUNC, CTR12 and CTR13 are both set to one; when FUNC returns a value, the terminal delimiter determines what action to take or subsequent calls--a VM indicates increment of CTR13 before the next call; a SVM indicates increment of CTR12; an AM indicates end of processing:

```
            .
            ONE   CTR13         SET VALUE NUMBER TO ONE
FC1         ONE   CTR12         SET SUB-VALUE NUMBER TO ONE
FC2         BSL   FUNC
             .
             .
            store value from IR
             .
             .
            DEC   R15
            BCE   AM,R15,END    END OF PROCESSING
            INC   CTR12         INCREMENT SUB-VALUE COUNT
            BCE   SVN,R15,FC2   GET NEXT SUB-VALUE
            INC   CTR13         INCREMENT VALUE COUNT
            B     FC1           GET NEXT VALUE; RESET SUB-VALUE COUNT
END         EQU   *             CONTINUE
```

5.14.10 SPECIAL U-CORRELATIVE EXIT

Format: 'U508E'

This may be used in a dictionary attribute to generate an output listing via the ENGLISH LIST or SORT statements that is identical in format to the COPY ... (T) output. This is mainly useful to list BASIC programs and other textual data in the COPY format, but where the ENGLISH feature such as pagination, headings, selection, etc. are to be used.

Example:

The dictionary of the BP (Basic Programs) file contains an attribute 'UCOPY':

```
            UCOPY

001     A
002
003
004
005
006
007
008     U508E
009     R
010     1
```

A formatted listing of items in the BP file can be generated by the statement:

```
:LIST BP UCOPY 'PROG1''PROG2' ID-SUPP COL-HDR-SUPP(CR)

:HEADING"BASIC PROGRAMS                    /]'   LPTR (CR)
```

The "ID-SUPP" is needed to prevent the item-id being duplicated on the output; the "COL-HDR-SUPP" suppresses the normal ENGLISH time/date heading and the attribute heading.

Notes:

1. Line numbers will appear on the output if the attribute with the U-correlative is right-justified (as in the example above); by using an "L" in line 9 of the attribute, line numbers will be suppressed.

2. The attribute with the "U508E" correlative should be the only attribute specified.