Section 3

Central Processor Architecture

This section describes the architecture of the Models 23 and larger central processors. From a functional point of view the processors are identical. The differences among them are in the amount of memory they can address, the number of registers, the kinds of options, and the number of peripherals each is capable of handling.

REGISTERS

In the Models 23 and 33, there are 18 registers visible to the programmer:

- o 7 general registers (R1–R7); (R1–R3) can be used as index registers
- o 7 base registers (B1–B7)
- o 1 program counter (P register)
- o 1 system status register (S register)
- o 1 mode control register (M1 register)
- o 1 indicator register (I register)

In the Models 43 and larger, there are 26 registers visible to the programmer. They include all of those found in the Models 23 and 33 plus the following:

- o 1 stack address (T) register
- o 1 remote descriptor base register (RDBR)[1]
- o 1 mode control register (M3) for the CIP
- o 3 RFU[2] mode control registers (M2, M6, M7)
- o 2 mode control registers for the SIP (M4, M5)

All of the 26 registers can be accessed via the control panel. Other registers can also be accessed from the control panel, including:

- o Instruction register
- o Memory address register
- o Memory data register

Finally, there is a single 32-bit scientific register that is simulated by software and is used to store floating-point operands. This register is utilized by the scientific instruction set that is automatically trapped to software routines in Model 23 and 33 systems. In Models 43 and larger, an optional Scientific Instruction Processor is offered that can hold either single-precision (32-bit) or double-precision (64-bit) floating-point quantities.

---

[1] Used with CIP instructions.

[2] Reserved for Future Use.

Data Formats

The word length in all Level 6 central processors is 16 bits. All hardware registers are 16 bits in length except for the M and I registers, which are 8 bits; scientific accumulators, which are 32 or 64 bits; and for the Models 43 and larger, the address registers, which are 20 bits. Within this framework the central processor has the ability to process double-words (32 bits in length), words (16 bits), half-words or bytes (8 bits), and single bits.

Basically, there are two types of data: signed data used in arithmetic operations; and unsigned data, used for logical quantities, addresses, ASCII characters, or any other type of internal data coding.

Unsigned data is usually expressed in hexadecimal notation. Thus, a 16-bit address can range from $(0000)_{16}$ to $(FFFF)_{16}$. The contents of a byte can be expressed from $(00)_{16}$ to $(FF)_{16}$. A 16-bit word can contain two ASCII characters; if a word contained $(4139)_{16}$, this would represent "A9" in ASCII.

Arithmetic (signed) data is represented in twos complement notation. All arithmetic is performed in binary, with single word (16-bit) values extending from -32,768 to +32,767. A signed value in a byte can range from -128 to +127. Figure 3-1 shows the various data types for both signed and unsigned data.

A byte in memory can represent either an unsigned 8-bit quantity or an arithmetic value with a sign and seven bits. The byte can occupy the left-or right-hand half of a word. However, when the byte is loaded into a register, it will occupy the right-half of the register. The left-half will contain either zeros, if a logical load instruction was used, or the sign extended, if an arithmetic load was used. (See Figure 3-2 for examples).

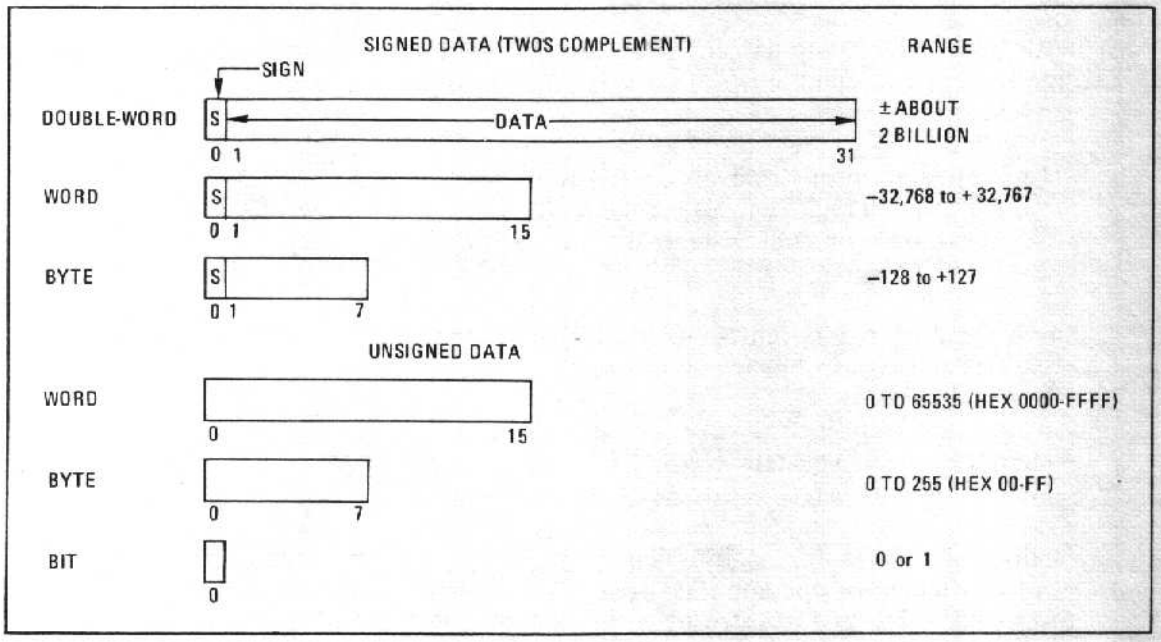


Figure 3-1. Data Formats

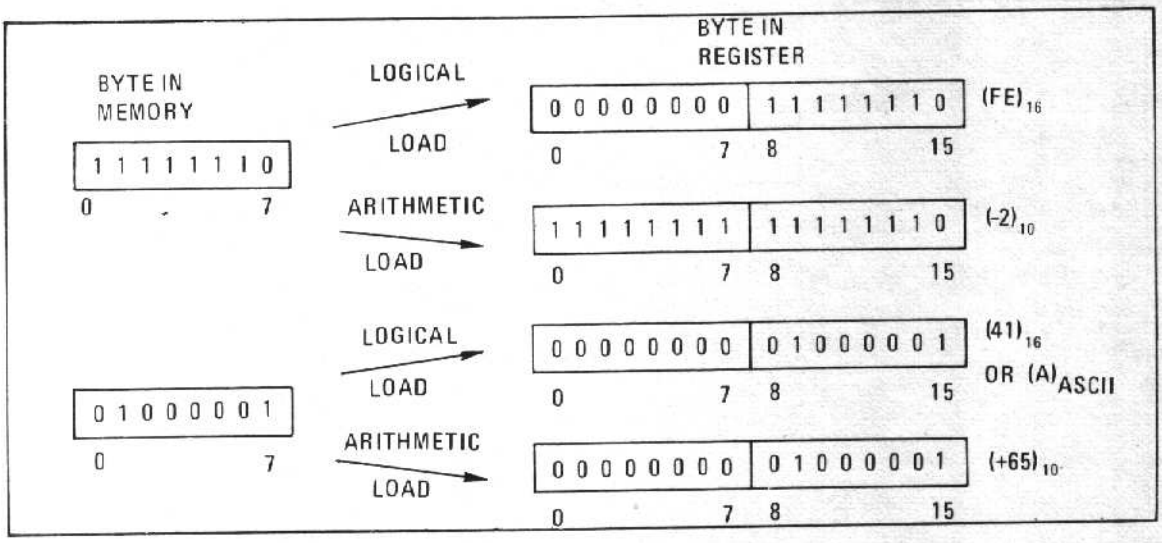
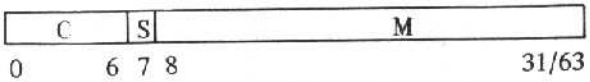


Figure 3-2. Byte Formats

Floating-point values occupy two or four words. The format for a floating-point value is as follows:



Where C is a 7-bit value representing the characteristic expressed as an excess 64 power-of-sixteen exponent
S is the sign of the mantissa
M is the magnitude of the mantissa

General Registers

There are seven general registers (R registers) numbered R1 through R7. These registers may be loaded from or stored in memory either a word or a half-word at a time. (If a half-word store is executed, it is the right-half of the register that is stored.) It is possible to load and store two of them (R6 and R7) as a single double-word. Each register can be shifted individually or as pairs (even/odd) and used as operands in arithmetic, logical, and compare operations. Additionally, the first three registers (R1—R3) also double as index registers and may be used to store double-word, word, half-word, or bit counts. These quantities are used to modify the addresses of items in memory. See Figure 3-3.

Base Registers

In addition to the seven general (R) registers, there are seven base registers (B registers). These are numbered B1 through B7. It is a very important concept of Level 6 architecture that the general registers are separate from the base registers. The Models 43 and larger also have a stack address (T) register and a remote descriptor-base register.

Address registers in the Models 23 and 33 are all 16 bits in length; however, in the Models 43 and larger they are 20 bits in length and are used either in the Short Address Form (SAF) or in the Long Address Form (LAF). SAF is oriented to systems of 128K bytes (64K words) or less of addressable memory where each word can be accessed through a 16-bit address pointer. LAF is oriented to systems of 1M words where each word can be accessed through

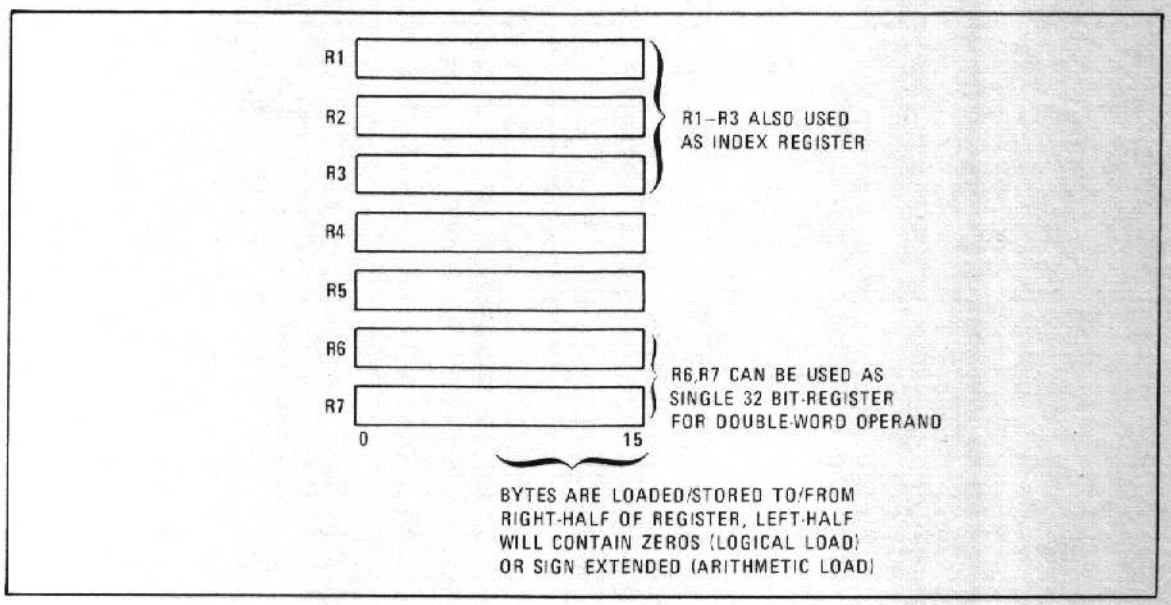
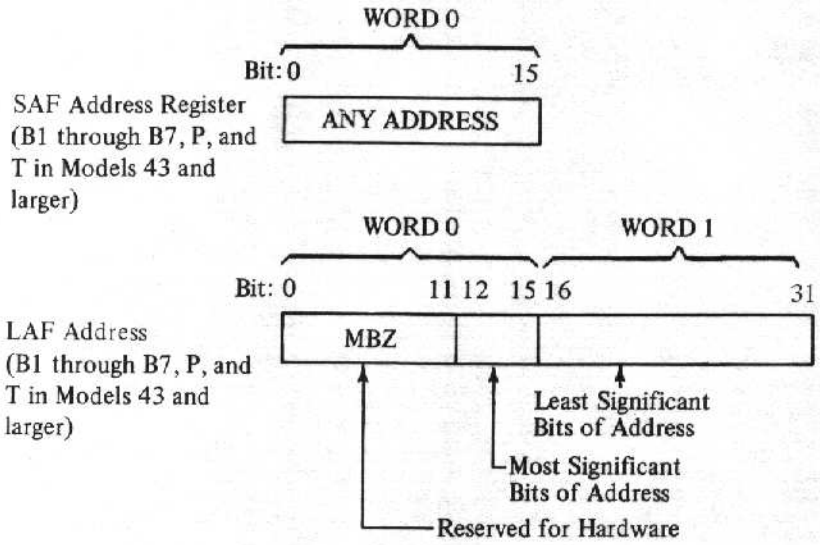


Figure 3-3. General Registers

a 20-bit address. An address stored in memory takes up one word (16 bits) in SAF mode and two words (32 bits) in LAF mode. The most significant 12 bits of an LAF address field in memory are reserved for hardware use.



The seven base registers can be used for formulating addresses by pointing to any procedure, data, or arbitrary location in the system. Address registers typically contain addresses and are not used for arithmetic calculations. However, they do have the capability of being automatically incremented or decremented during instruction execution. This allows them to be used to scan arrays either forwards or backwards and also to conveniently utilize stacks.

Addresses are normally expressed in memory as four or eight hexadecimal digits, with the memory addresses running from 0-0 through F-F. Details of how base registers are used in formulating addresses are given in Section 4.

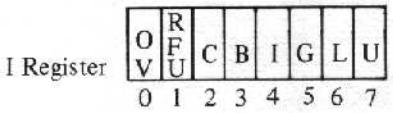
Program Counter

The program counter (P register) is a 16-bit (SAF) or 20-bit (LAF) register which points to the next instruction to be executed. Instruction length is variable. The majority of instructions are one or two words in length, but certain instructions can be as long as ten. The program counter is always incremented during instruction execution to point to the next sequential instruction, except for branches and jumps. Thus, for example, if a two-word instruction is executed, the program counter will be incremented by two during its execution.

The Models 43 and larger have a pre-fetch or look-ahead capability. Software must ensure that procedures within the pre-fetch range are not modified at run time. If procedures must be modified at run time, a jump or branch instruction must be executed after the modification, and before subsequent execution of the modified code.

Indicator Register

The indicator register (I register) is an 8-bit register that contains various single-bit indicators. The register format is as follows:



The indicators contained in this register can be grouped as follows:

- o Arithmetic indicators
  - – OV (overflow indicator)
  - – C (carry bit)
- o Comparison indicators
  - – G (greater than indicator)
  - – L (less than indicator)
  - – U (unequal signs indicator)
- o Bit indicator
  - – B (bit test indicator)
- o I/O Indicator
  - – I (input/output indicator)

Arithmetic Indicators

Two indicators are affected by arithmetic and shifting operations: the overflow (OV) indicator and the carry (C) indicator.

The overflow indicator is set when any of the seven general registers "overflows," that is, when an arithmetic result produced is larger than the capacity of the register. For example, adding the quantity 1 to a register that contains +32,767 $(7FFF)_{16}$ will set overflow because the *arithmetic* capacity is exceeded. The register would contain $(8000)_{16}$, or −32,768 after the addition. If the data were not to be interpreted as signed data, the overflow could be ignored.

The carry (C) indicator, conversely, is set when the *logical* capacity of a register is exceeded. Thus if 3 were added to a register that contained $(FFFE)_{16}$, a carry to the 17th bit would be produced, i.e., the answer would be $(1)(0001)_{16}$. However, if these were arithmetic quantities, $(FFFE)_{16}$ would represent −2, and adding +3 to it would produce $(0001)_{16}$; this is the correct answer and thus the carry would be ignored.

Comparison Indicators

The three indicators which are controlled by compare instructions are the greater than (G), less than (L), and unlike signs (U) indicators. These one-bit indicators contain the results of the last compare instruction executed by the computer and in turn can be tested by other central processor instructions (see Branch on Indicator in Section 4).

A comparison is typically executed between a register and a word in memory. If the contents of the register are greater, the G indicator is set; if the contents of memory are greater, the register is less and L is set.

Bit Test Indicator

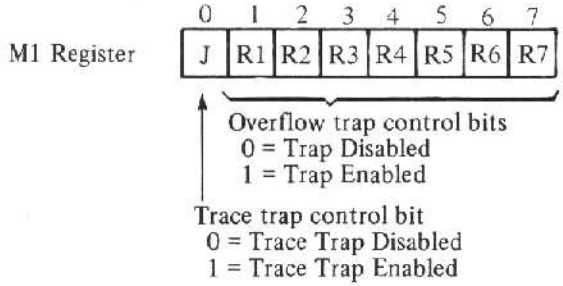
This can be considered a one-bit register loaded by load bit instructions. Not only can it be loaded from any bit in any word of memory, but it can also be set if any of a group of bits in a word in memory is set. The selection is done under the control of a 16-bit mask and is very useful for testing bit patterns in memory.

Input/Output Indicator

The I-bit stores an indication of whether the last input/output command was successful. For example, if the central processor issues a command output to a peripheral channel that is busy, the peripheral channel will issue a "NAK," which in turn will cause the I/O indicator to be cleared. Software can then test this indicator and determine what alternate action to take.

Mode Control Registers

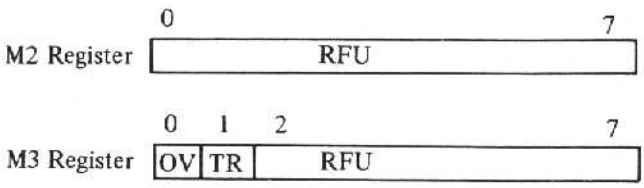
Other control registers are the trap enable/mode control registers (M1–M7). These are all 8-bit registers. Registers M2, M6, and M7 are reserved for future use.



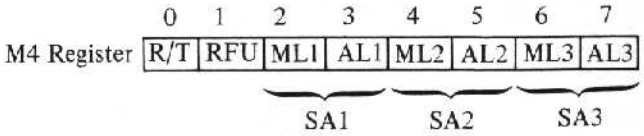
The seven overflow trap control bits are associated with the seven general registers, R1–R7. If overflow in any one of these occurs and if the corresponding trap control bit is set, a trap occurs through trap vector 6. This facility saves the programmer from having to test the result of every arithmetic operation for overflow, and yet also guarantees that overflow will not go undetected.

The other bit in this register, bit zero, is the trace trap enable bit (J-bit). When this is enabled, all jumps and branches that are executed in a program will cause a trap to the trace entry location. This bit therefore allows a programmer to trace a code without having to modify its procedure at all.

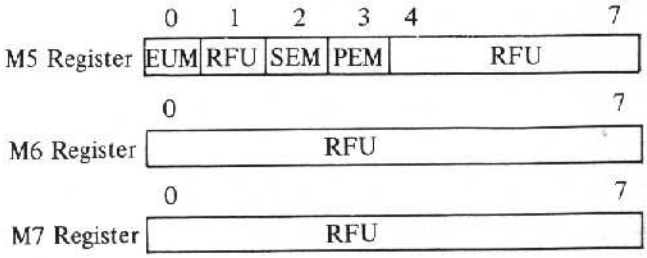
The following M registers are found only in the Models 43 and larger. The M3 register is only used with the Models 47 and 57. The M4 and M5 registers are used only when the optional SIP is installed.



The format of the CIP Control Register (M3) is the same as that of the CIP Mode Register which is described later in this section.

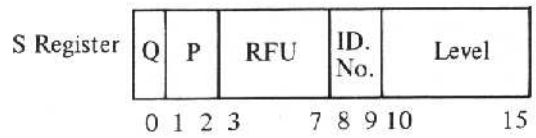


The formats of registers M4 and M5 are the same as those of the SIP Mode Register and the SIP Trap Mask Register respectively. These registers are described later in this section.



Status Register

The status register (S register) is a 16-bit register. This register contains four fields, as shown below: QLT indicator (Q), the privileged state indicator (P), the processor ID, and the interrupt priority level.



QLT Indicator

This 1-bit field indicates whether a unit in the system has successfully completed its Quality Logic Test (QLT) or not.

0 = QLT successfully completed
1 = QLT either still running or failed

Privileged State Indicator

There are two modes of instruction execution: user mode and privileged mode. The privilege field in the S register defines this mode as follows:

P = Privilege State (Ring Number)
11 = Ring 0 (Privilege)
10 = Ring 1 (Privilege)
01 = Ring 2 (User)
00 = Ring 3 (User)

NOTE: Privileges and access rights accorded the various rings are in inverse order to the ring number (i.e., Ring 0 is the most privileged).

If bit 1 is not set, the processor is in the user mode and will automatically trap when certain instructions are attempted (bit 2 is ignored). Input/output command instructions plus the interrupt level change instruction are privileged instructions and can be executed only when the privilege mode bit is set. They will be automatically trapped if attempted in user mode. By utilizing this hardware feature, systems can be protected against unauthorized use of input/output by user routines.

Processor ID

This is a 2-bit field that is fixed during system configuration. It is typically zero (a second processor in the system would have an ID of 01). These 2 bits are used as the least significant bits of the 10-bit channel number for the processor itself. The high 8 bits are always zero. Thus, the processor ID and the processor channel number are for all intents and purposes synonymous.

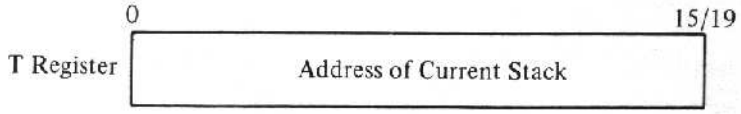
The processor ID in the S register is hard-wired and cannot be changed under program control.

Priority Level

This 6-bit level field defines the interrupt priority level on which the processor is currently executing instructions. Zero is the highest priority level and 63 is the lowest. Upon receiving interrupt requests from other units, it also determines whether the interrupting unit is of higher, equal, or lower priority. Only higher priority interrupt requests are granted. When an interrupt occurs, the level of the interrupting unit replaces the level in the status register. The old level is always automatically stored. (See Interrupts for further details.)

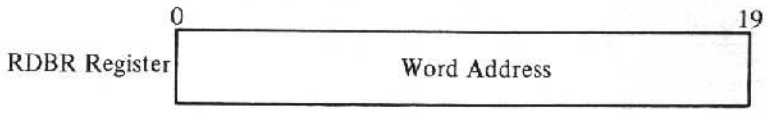
Stack Address Register

This register, which is used only with the Models 43 and larger, points to the first word of the stack header. See also "Stack Management" described later in this section.



Remote Descriptor Base Register

This register, which is used only with the Models 47 and 57, contains the address of a remote descriptor array. An offset of the RDBR and the contents of a label in the CIP instruction are combined to generate the remote descriptor address. See also "Commercial Instructions" in Section 4.



SUMMARY OF PROGRAM VISIBLE REGISTERS

Thus 18 registers are visible to the Model 23 and 33 programmer and 26 to the Model 43, 47, 53, and 57 programmer. The central processor registers are shown in Figure 3-4. Of the various registers, two are automatically saved and restored upon interrupt (the S register and

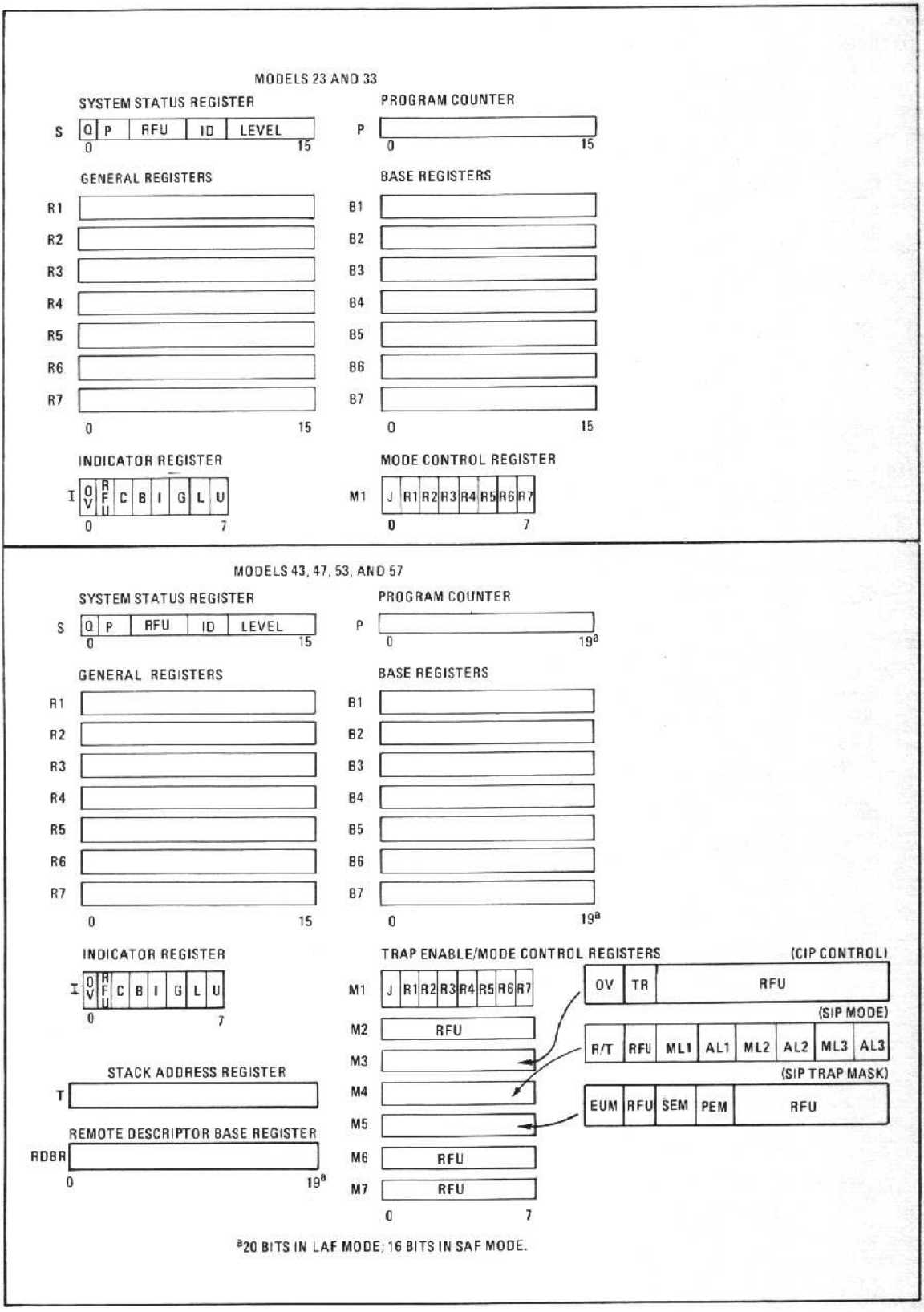
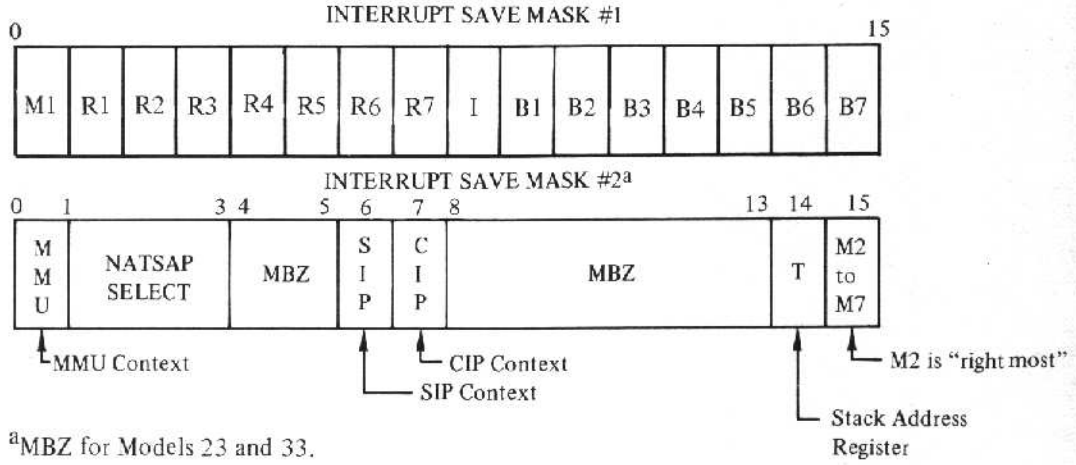


Figure 3-4. Central Processor Register Complement

the P register). The others have their context stored and restored under firmware control according to a 16-/32-bit mask. This mask is set up under program control. Its format is as follows:



[a]MBZ for Models 23 and 33.

INTERRUPTS

There are 64 levels of interrupt, numbered from 0 to 63; level 0 has the highest priority. Clearing the computer puts it into level 0, which in effect makes it uninterruptable.

Associated with each interrupt level is a dedicated memory location which contains the interrupt vector. These locations extending from address 0080 to address 00BF (00FF for LAF) contain a pointer to an interrupt save area, which is a block of memory in which the hardware/firmware saves context when an interrupt occurs. The interrupt save area needs only to be set up for those levels that are active in a particular program. Each interrupt save area has five or six fixed locations and up to 47 variable locations. These locations are as follows:

- DEV – this location contains the channel number (bits 0-9) and level (bits 10-15) of an interrupting device.

  NOTE: DEV is not updated when a level is activated by an internal condition such as a power failure (level 0), watchdog timer (level 1), trap save area overflow (level 2), real-time clock (assigned level), or execution of a LEV instruction. Interrupt levels 1, 2, and the assigned real-time clock level should not be used for devices. If these levels are assigned to devices, or if multiple devices have been assigned to the same interrupt level, software should clear DEV before each I/O operation that will interrupt. This ensures that the interrupted level can determine the interrupt source.

- ISM 1/2 – these two locations contain the 32-bit interrupt save mask. This mask controls which of the registers will be saved in the variable portion of the interrupt save area.
- P – in this word or two, the program counter of the interrupt level is stored. It acts as a pointer to the interrupt handling procedure for a new level, or upon restoration of an interrupted level, to the location of the next instruction to be executed.
- S – this is where the status register is automatically stored. Note that when a new interrupt level is set up, the S register is loaded from this location only as far as the privileged mode field is concerned; the level is generated automatically, and the processor ID is hard-wired.

Words 6–52 are locations for saving the machine registers under control of the interrupt save mask. If the interrupt save mask is all zeros, none of these words will be reserved. The mask bits are scanned from right (bit 15) to left (bit 0), ISM1 first and then ISM2.

It should also be noted that the word(s) prior to the one pointed to by the interrupt vector (that is, the word(s) prior to the device word) are also dedicated. This includes the TSAP which points to a trap save area. (See discussion of Traps.)

Figure 3-5 shows this action of an interrupt. Both levels 20 and 30 have had their interrupt save areas set up – level 20s at AB20, as pointed to by its vector at 0094 or 00A8; and level 30s at 1000, as pointed to by its vector at 009E or 00BC.

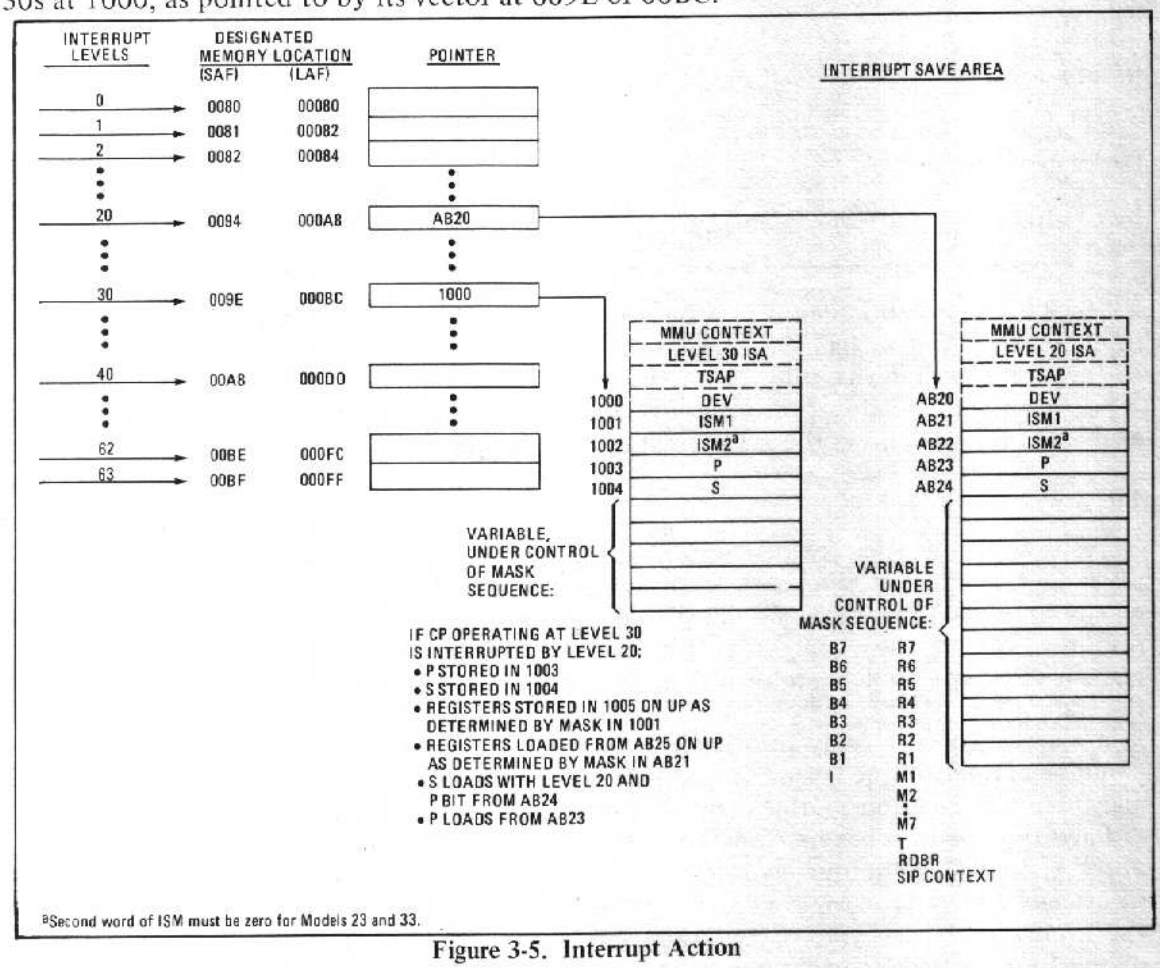


Figure 3-5. Interrupt Action

If level 30 is interrupted by level 20, firmware automatically saves the register contents in the level 30 interrupt save area and loads the registers with the values in the level 20 interrupt save area. One of these values is the starting address of the level 20 interrupt subroutine which is automatically loaded (in this example) from location AB23 into P.

Associated with each interrupt level is a dedicated flag bit which is set when the interrupt is initiated. These bits are stored in four dedicated memory locations, 0020-0023. In the example above, both bits 20 and 30 would be set. At the end of the interrupt routine for level 20, a level change instruction (LEV) would be executed. This would clear the bit for level 20 and then scan the table to determine which is the next highest scheduled level. If no intermediate interrupts (such as level 25) were pending, it would scan the table, find bit 30 set, and therefore return to level 30.

TABLE 3-1. EVENT INTERRUPT LEVEL ASSIGNMENT

Event Causing Interrupt	Level Assignment	Comment
Incipient Power Failure	0	Highest priority
Watchdog Timer Runout	1	—
Overflow of Trap Context Save Area	2	—
Real-Time Clock	—	Level is contained in main memory location 0016 (HEX)
Device Requiring Service	—	Level is dynamically controlled by software
LEV Instruction	—	Level specified in instruction

The LEV instruction can set or clear activity flags, change the current level, inhibit interrupts, or do a "quick change" without saving and restoring context. By changing the current level to level 3 all device interrupts can be inhibited; level 2 (overflow of trap context save area), level 1 (watchdog timer runout), and level 0 (incipient power failure) will still be enabled. Table 3-1 shows the assignment of interrupt levels.

TRAPS

Traps are a special software- or hardware-related condition that may occur during execution of a task. The Level 6 hardware/firmware responds to many trap conditions. The design of any application program should provide that when a trap occurs, the hardware/software response will include calling a dedicated software routine (a trap handler) to react to the trap. When trap handlers are provided, handling the trap is invisible to the task that caused the trap.

A trap can occur at any priority level, and several can be nested at the same level. A trap could be entered at one level, that level interrupted during the execution of the trap routine, and then the same trap routine reentered in the new level. See Table 3-2.

Each type of trap has its own trap vector containing a pointer to the trap-handler procedure. Also utilized are four[3] pointers (NATSAPs) at locations 10 and below to form "pools" of available Trap Save Areas (TSA). The linkage between TSAs in a pool is maintained by the firmware. When a trap occurs, the CP uses the NATSAP selected by ISM2 to access one of the four TSA pools (only one pool is supported by the Models 23 and 33). If the pool is empty, the CP will halt; otherwise, the first available TSA is unlinked from the pool and some (but not all) register contents are automatically stored in the trap save area. The pointer in the first word of the interrupt save area for the current level is adjusted so that it points to the trap save area; the pointer in the trap save area points to any other traps that occurred at the same interrupt level. Thus, several traps may be nested at the same interrupt level. At the end of the execution of the trap-handler procedure, a return from trap (RTT) must be executed; this does a restoration of the partial context that was stored and returns all pointers to their original state.

The relationship of traps and interrupts, and their vector linkage are shown in Figure 3-6.

[3] Only one NATSAP (next available trap save area pointer) is available in the Models 23 and 33 processors.

TABLE 3-2. TRAP VECTORS AND EVENTS

Vector #	Event
Vector #00	No trap event set
Vector #1	Monitor call (MCL instruction)
Vector #2	Trace[a] (debug) or BRK instruction
Vector #3	Scientific operation not in hardware
Vector #4	Reserved for software use
Vector #5	Other operation not in hardware (or undefined)
Vector #6	Integer register overflow[a]
Vector #7	Scientific divide by zero
Vector #8	Scientific exponent overflow
Vector #9	Stack underflow
Vector #10	Stack overflow
Vector #11	Reserved for future use
Vector #12	Recursive Remote Descriptor usage
Vector #13	Unprivileged use of privileged operation
Vector #14	Unauthorized reference to protected memory (with optional protection)
Vector #15	Reference to unavailable resource
Vector #16	Program error
Vector #17	Memory or Bus error (parity or noncorrectable EDAC) detected
Vector #18	Reserved for future use
Vector #19	Scientific exponent underflow[a]
Vector #20	Scientific program error
Vector #21	Scientific significance error[a]
Vector #22	Scientific precision error[a]
Vector #23	Reference to unavailable resource by SIP or CIP
Vector #24	Memory or Bus error seen by SIP or CIP
Vector #25	Commercial Divide by Zero
Vector #26	Commercial Illegal Specification
Vector #27	Commercial Illegal Character
Vector #28	Commercial Truncation
Vector #29	Commercial Overflow
Vector #30	CIP QLT fault
Vector #31	SIP QLT fault
Vector #32 through Vector #46	Reserved for future use

NOTE: The Models 23 and 33 do not use Trap Vectors 18 through 46. Other trap vectors are used/not used depending on the model and options installed.

[a] If enabled.

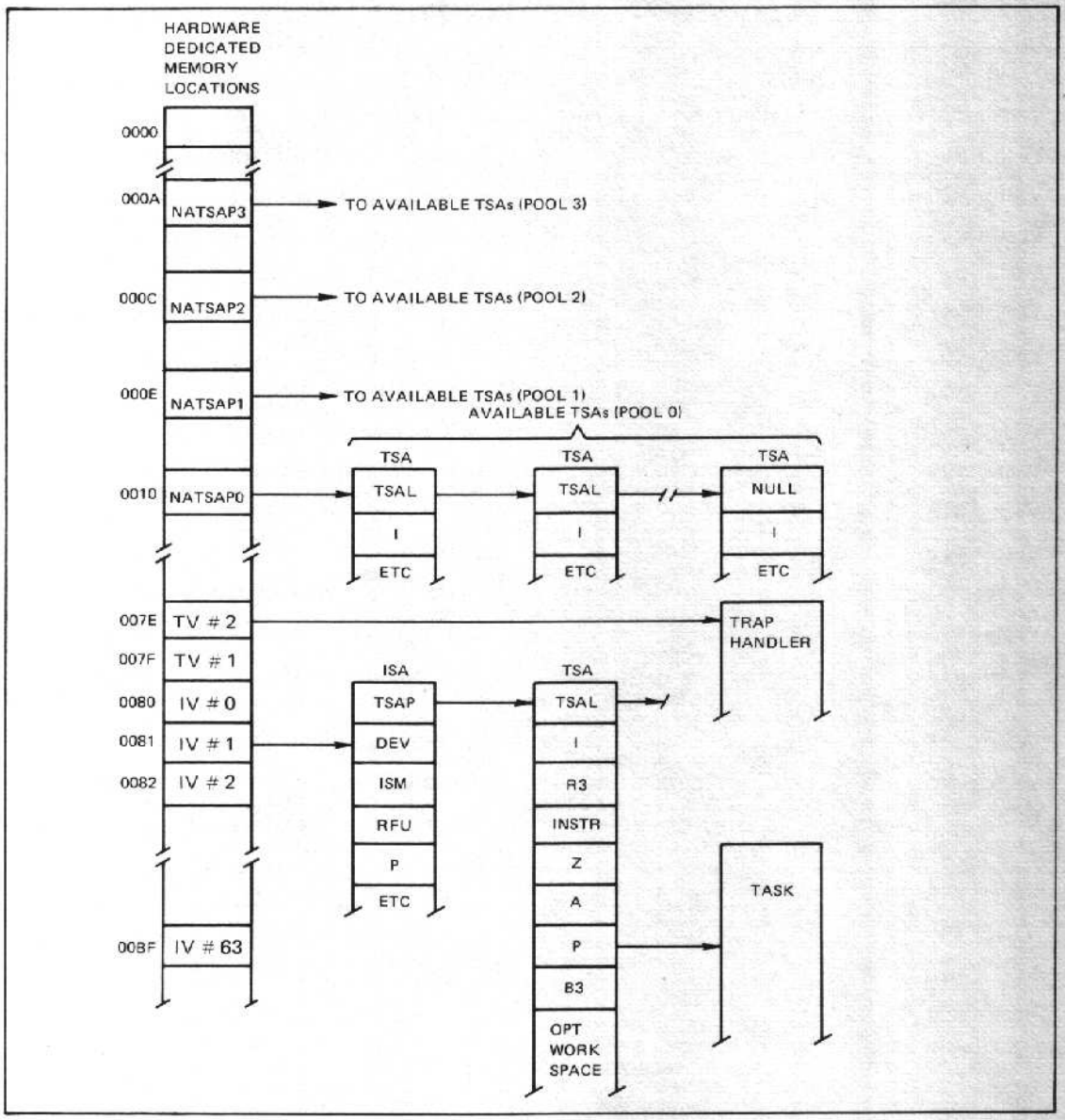


Figure 3-6. Trap Vector and Interrupt Vector Linkage

- o NATSAP – Pointers to Next Available Trap Save Area: Four NATSAPs (NATSAP0 to NATSAP3) are maintained at location 0010 and below. The NATSAP (and TSA pool) that is to be used for the current level is selected by bits 1-3 of ISM2. NATSAP0 at location 0010 is the only TSA pool pointer supported by the Models 23 and 33. Refer to Figure 3-6.
- o TSA – Trap Save Areas: Initially, when processing of an application begins, all trap save areas exist in a linked "pool," which is pointed to by memory location 0010_{16}. All trap save areas remain in this pool until a trap condition occurs within a running task, at which point the hardware/firmware (1) stores information in the first available trap save area in the pool, (2) links this trap save area to the interrupt save area for the priority level of the task that was running when the trap occurred, and (3) unlinks this trap save area from the pool. Later, after the trap handler (if any) has completed its

work, the trap save area is returned to the pool of available trap save areas. Thus, at any time, a given trap save area is either in the pool of available trap save areas or in use because of a trap condition. The trap save areas reside in the system pool.

Trap save areas are 64 words long in SAF mode, and 104 words in LAF mode.

- o TSAL – Trap Save Area Link: When the trap save area resides in the "available" pool, TSAL points to the next trap save area in the pool; the TSAL of the last trap save area in the pool contains a null pointer. When the trap save area is in use (i.e., connected to an interrupt save area), TSAL contains a null pointer (if this is the only or last trap save area connected to this interrupt save area) or it points to the next trap save area connected to this interrupt save area.
- o I – Indicator Register: The contents of this register are saved by hardware/firmware when a trap occurs. This register is then available for use by the trap handler.
- o R3 – R3 Register: The contents of this register are saved by hardware/firmware when a trap occurs. This register is then available for use by the trap handler.
- o INSTR – Instruction: The hardware/firmware stores the instruction associated with the trap. If a multiword instruction is involved, the first word is saved.
- o Z – Z-Word: This word contains miscellaneous information relative to the trap.
- o A – A-Word: In many cases, this word contains an address associated with the trap. The nature of the saved address is governed by the specific trap condition and the specific instruction associated with the trap.
- o P – Program Counter: The contents of the program counter are saved by the hardware/firmware when a trap occurs. This is the address to which a return is made when the trap handler completes. In most cases the program counter will point to the instruction or location following the instruction associated with the trap. However, when an input/output instruction is involved, the program counter may point to an address *within* the instruction; in this case, the trap handler must modify this word before issuing a return to "normal" task processing.
- o B3 – B3 Register: The contents of this register are saved by hardware/firmware when a trap occurs. This register is then available for use by the trap handler; as the trap handler is entered, the B3 register points to the A-word in the trap save area.

Note that when a trap occurs, if the appropriate trap handler is available in the application, the first word (TSAP) of the interrupt save area (for the current priority level) is set to point to the link word (TSAL) of the trap save area in which hardware/firmware has just stored information relative to the trap (see Figure 3-6). TSAP is subsequently used by the trap handler to gain access to the trap save area.

QUEUE MANAGEMENT

The Models 43 and larger provide a queue capability that allows easy maintenance of ordered lists of "frames" (a frame contains a frame priority number, a next frame pointer, and an associated data structure). Each list is identified by a lock frame which contains a lock word and list head and tail pointers (Figure 3-7).

Four generic instructions are provided to enqueue/dequeue frames from the list. The instructions, which are described in Section 4, are as follows:

- o Queue on Head (QOH)
- o Queue on Tail (QOT)
- o Dequeue from Head (DQH)
- o Dequeue by Address (DQA)

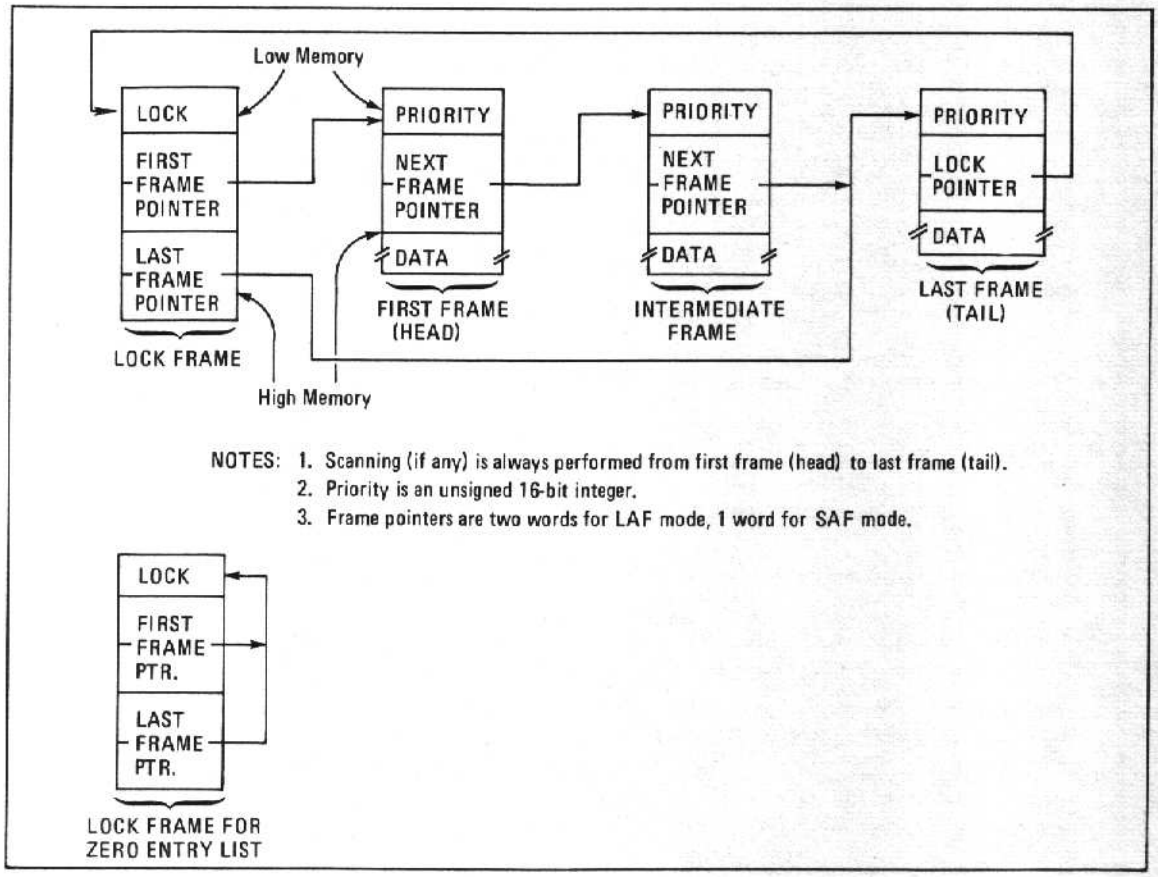


Figure 3-7. Queue Management

The lock word is used to ensure that only one CP is accessing a particular queue at a time. Each queue instruction causes a fetch of the lock word with a Read-Modify-Write (RMW) cycle. If the low-order bit of the lock word is set, the RMW cycle is completed without changing the lock, I (C) is cleared, and the next instruction is fetched. If the low-order bit of lock is cleared, the CP completes the RMW cycle, writing ones into the lock word, and initiates execution of the queue/dequeue instruction. Each queue/dequeue instruction causes a scan of the frames from the head toward the tail.

The scan continues until the conditions of the particular command are met (a hit), the last frame is reached without a hit, or an interrupt occurs.

When a hit occurs or if the last frame is reached without a hit, the frame is linked into or out of the list as appropriate, and I(G) and I(L) are left in a known state. In either case, the CP initiates another RMW cycle, writing zeros into the lock word, sets I(C), then fetches the next instruction. If an interrupt[4] occurs, the CP stops the scan, initiates an RMW, writes zeros into the lock word, clears I(C), leaves I(G) and I(L) undefined and backs up P to point to the queue/dequeue instruction, before servicing the interrupt.

Software must build the lock frame of each list to be used. A list with no entries is a lock frame in which the first and last frame pointers point to LOCK (see Figure 3-7). The CP leaves the lock frame in the same condition when a frame is unlinked from a single frame list.

---

[4] This includes service of the internal timer at a 120-Hz rate.

STACK MANAGEMENT

The Models 43 and larger provide a single stack capability for each interrupt level. The Stack Address Register (T) points to the first word of the stack header (Figure 3-8).

Four generic instructions are provided to manipulate the stack. The instructions, which are described in Section 4, are as follows:

- o Load Stack Address Register (LDT)
- o Store Stack Address Register (STT)
- o Acquire Stack Frame (ACQ)
- o Relinquish Stack Frame (RLQ)

These are all two-word instructions having a common first word. Appropriate checks are made for stack overflow/underflow conditions.

The stack header contains four entries, two of which are not used, but must contain null pointers if upward software compatibility is desired.

MW is the number of words allocated to this stack. MW is written by software when the header is created and referenced (but not altered) by hardware.

CW represents the number of words currently consumed in this stack. CW is written by software when the header is created. Thereafter, the value of CW is a hardware responsibility.

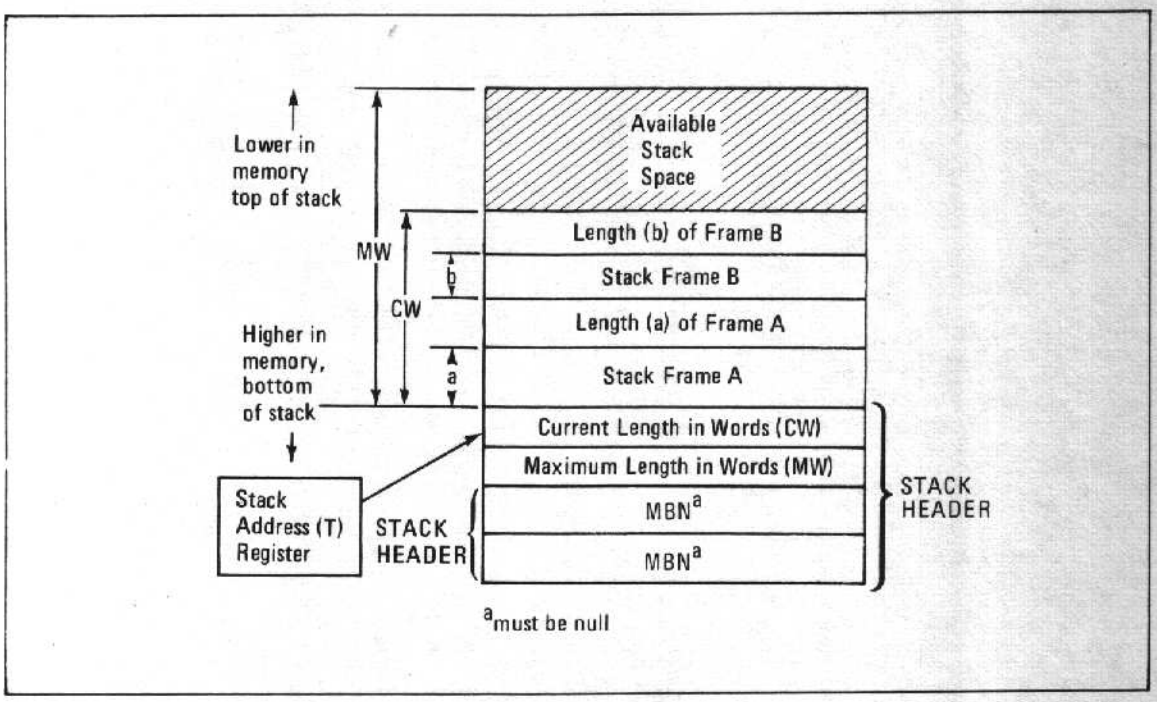


Figure 3-8. Stack Structure

SCIENTIFIC INSTRUCTION PROCESSOR

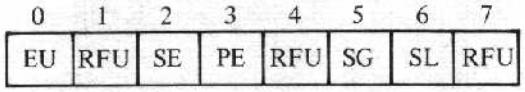
The Scientific Instruction Processor (SIP) is optionally (CPF9503) available on Models 43, 47, 53, and 57. The 30 scientific instructions operate on single-precision and double-precision (respectively, double-word and quadruple-word fields) with floating-point, integer and zero formats. These operands come from main memory, CP registers (following their conversion to two- or four-word floating-point quantities) and scientific accumulator (SA) registers. Prior to command execution involving two operands, the SIP tests for unequal length floating-point values. If the operand lengths are not equal, the destination scientific accumulator's length dominates.

Control Registers

There are three scientific control registers in the SIP. Two of these registers, the SIP Mode Register (M4) and the SIP Trap Mask Register (M5) reside in the associated CP with a copy of their contents stored in the SIP. The third register is the SIP Indicator Register, located in the SIP.

Scientific Indicator (SI) Register

The 8-bit SI-register contains error and status indicators that can be tested with the scientific branch instructions.

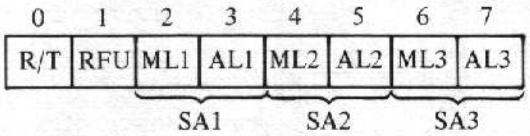


where:

EU = Exponent underflow (trap 19)
SE = Significance error (trap 21)
PE = Precision error (trap 22)
SG = Greater than
SL = Less than

SIP Mode (M4) Register

The SIP mode, or M4, register is an 8-bit control register residing in the SIP, but with a copy in the CP. Both versions are set to 0 upon CP initialization and both may be modified with an MTM instruction. If only the SIP is initialized, the CP copy of the register is not cleared, and the contents of both versions must be reestablished with an MTM.

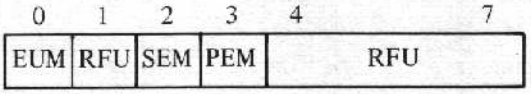


where:

R/T = Round/Truncate Mode
0 = Truncate
1 = Round
SA1 = Scientific Accumulator 1
SA2 = Scientific Accumulator 2
SA3 = Scientific Accumulator 3
ML = Memory Length – Length of main memory data field associated with this SA.
0 = 2 words
1 = 4 words
AL = Accumulator Length – Length of the value in the scientific accumulator data field.
0 = 2 words
1 = 4 words

SIP Trap Mask (M5) Register

The SIP Trap Mask, or M5 register, is an 8-bit control register residing in the SIP but with a copy in the CP. Both versions are set to 0 upon CP initialization and both may be modified with an MTM instruction. If only the SIP is initialized, the CP copy of the register is not cleared, and the contents of both versions must be reestablished with an MTM.



where:

EUM = Exponent Underflow Trap Mask
- 0 = Trap disable
- 1 = Trap enable

SEM = Significance Error Trap Mask
- 0 = Trap disable
- 1 = Trap enable

PEM = Precision Error Trap Mask
- 0 = Trap disable
- 1 = Trap enable

RFU = Reserved for future use (MBZ)

Accumulators

The SIP contains three variable length scientific accumulators (SA1, SA2, SA3) that may contain either single-precision (32-bit) or double-precision (64-bit) floating-point quantities.

Automatic Round/Truncate

Scientific operations requiring right scaling, which is the process of shifting floating-point mantissa digits to the right, inserting zero digits into the most significant mantissa digit positions and increasing the exponent appropriately, may produce results where nonzero data is shifted off the end of the scientific accumulator into guard digit locations. These guard digits will be truncated from the result if bit zero of mode register (M4) is zero. If bit zero is a one, the result is rounded using the significant guard digits.

Scientific Traps

As the SIP is an option of the CP and functions as an extension of it, any SIP exception conditions are reported to the CP in the form of traps rather than interrupts. This trap facility is activated upon detection of specific status conditions during the execution of a scientific instruction. The SIP has nine types of trap vectors.

These traps and the conditions that cause them are shown in Table 3-3. Trap 23, reference to unavailable resource, and Trap 24, bus parity or uncorrected main memory error, are functions of the Megabus. Trap 20, program error, identifies program errors detected by the SIP. Note that program errors detected by the CPU activate Trap 16. If more than one trap condition exists, the SIP sends the trap with the highest priority to the CPU. The other conditions are lost. The priority of the traps is indicated in Table 3-3 by their location; the trap at the top (Trap 31) has the highest priority.

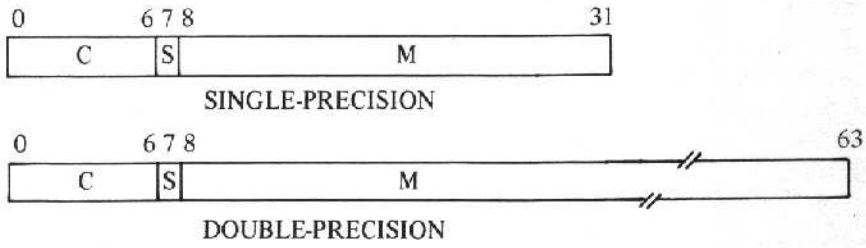
TABLE 3-3. SIP TRAP VECTORS AND EVENTS

Trap Vector Number	Trap Event	C/U[a]	SIP Registers	
			Indicator	Mask
31	SIP QLT Fault	U	–	–
23	Reference to Unavailable Resource	U	–	–
24	Megabus/Memory Error	U	–	–
20	Program Error (SIP)	U	–	–
7	Divide by Zero	U	–	–
8	Exponent Overflow	U	–	–
21	Significance Error	C	SI	M5
19	Exponent Underflow	C	SI	M5
22	Precision Error	C	SI	M5
00	No Trap Event Set	–	–	–

[a]C – Conditioned Trap on Indicator and Mask
U – Unconditioned Trap

Data Formats

Floating-point data appears either as a single-precision (32-bit) or double-precision (64-bit) constant, as follows:



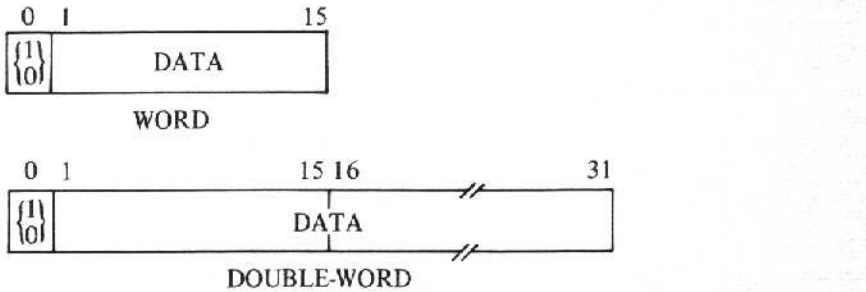
where:

C = the characteristic (excess 64 power-of-16 exponent) of the number. The characteristic represents exponents with a range from -64 to +63. Since the characteristic has no sign bit, the number 64 (decimal) is effectively added to each exponent, thus allowing a characteristic range of 0 to 127 to represent exponents with a range of -64 to +63.

S = sign bit (0 = +; 1 = –) of the mantissa.

M = magnitude of the mantissa.

Signed integer data contains a sign (0 = +; 1 = –) in bit 0 and the data in the remaining bits. Negative numbers appear in twos-complement form. Word and double-word formats are permitted, as follows:



Single-word integers must be in general registers R4, R5 or R6 of the CP in order to be processed by the SIP. Double-word integers must be in registers R6 and R7.

Software Simulation of the SIP

For systems on which a Scientific Information Processor (SIP) is not available, GCOS provides the equivalent of the SIP functions through software simulation. Two trap handlers, the Floating-Point Simulator, entered via trap vector 3, and the Scientific Branch Simulator, entered via trap vector 5, are available. These two simulators are described in the *System Service Macro Calls* manual, Order No. CB08.

COMMERCIAL INSTRUCTION PROCESSOR

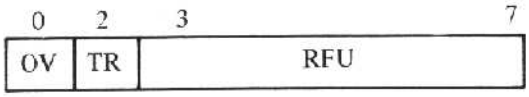
The Commercial Instruction Processor (CIP) is only available with the Models 47 and 57. The CIP operates on a powerful set of 30 instructions including numeric, alphanumeric, edit, and branch instructions. Operands are specified by data descriptors that give operand location, length, type (packed/unpacked decimal or character string), and other necessary information (leading/trailing sign, blank fill/no fill, etc.).

Control Registers

There are two commercial control registers in the CIP, the CIP Mode Register and the CIP Indicator Register.

CIP Mode (M3) Register

The 8-bit CIP mode register is a copy of the M3 register (in the CPU) which is provided for use with the CIP. Both are set to zero at initialization of the CPU. Both registers may be modified with an MTM instruction. If only the CIP is initialized, the M3 register is not cleared, and the contents of both registers must be established with an MTM instruction.



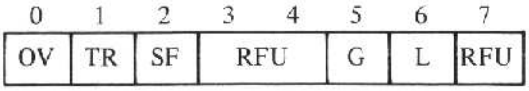
where:

OV = Overflow Trap Mask
  0 = Disable Trap
  1 = Enable Trap
TR = Truncation Trap Mask
RFU = Reserved for future use (MBZ)

Note that, although the contents of the CIP mode register are not saved, the equivalent information in the M3 register is saved or restored as a function of the mask bits in the interrupt save area. When a restore is done, the restored value is sent to the CIP by the CPU.

CIP Indicator Register

The 8-bit CIP indicator register is cleared at initialization. During the execution of an instruction that affects the register, only the bits pertinent to the instruction are preset (set or reset). All other bits remain unchanged. During the execution of a branch instruction, all bits including the one being tested are left unchanged.



where:

OV = Overflow occurred during decimal instruction
TR = Alphanumeric result is truncated
SF = Sign fault (negative operand is stored in unsigned field)
G = Greater than
L = Less than

Commercial Traps

CIP exception conditions are reported to the CP in the form of traps rather than interrupts. This trap facility is activated upon detection of specific status conditions during the execution of commercial instructions. The CIP has eight types of trap vectors. These traps and the conditions that cause them are shown in Table 3-4.

TABLE 3-4. CIP TRAP VECTORS AND EVENTS

Trap Vector Number	Trap Event	C/U[a]	CIP Registers	
			Indicator	Mode
23	Reference to Unavailable Resource	U	–	–
24	Bus or Memory Error	U	–	–
25	Divide by Zero	U	–	–
26	Illegal Specification	U	–	–
27	Illegal Character	U	–	–
28	Truncation	C	CI	CM
29	Overflow	C	CI	CM
30	Inoperative CIP	U	–	–

[a]C = conditional Trap
U = unconditional Trap

Software Simulation of the CIP

For systems on which a Commercial Instruction Processor (CIP) is not available (i.e., Models 23, 33, 43, and 53), GCOS provides the equivalent of the CIP functions through software simulation. The CIP Simulator is entered via trap vector 5.

MEMORY MANAGEMENT UNIT

The Memory Management Unit (MMU) is optionally (CPF9501) available on the Models 43 and 47, and standard on the Models 53 and 57. MMU functionality provides for the separation of memory into 16/31 independent segments, the relocation of each segment independently in physical memory, and the protection of each segment from improper access, based on software-specified attributes.

Segmentation

The MMU option divides the million-word address space into 16 regions of 65,536 words each, numbered 0 through 15. The first of these regions is subdivided into 16 regions of 4096 words each, numbered 0.00 through 0.15. Each of these regions may contain a segment of program address space. Each segment may range from 256 words in size up to the 4K or 64K size of its associated region, in steps of 256 words.

In Short Address Form (SAF), segments 0.00 through 0.15 are available, corresponding to the SAF address space of 64K. In Long Address Form (LAF), segments 1 through 15 are also available, yielding a total of 31 segments corresponding to the LAF address space of one million words. See Figure 3-9.

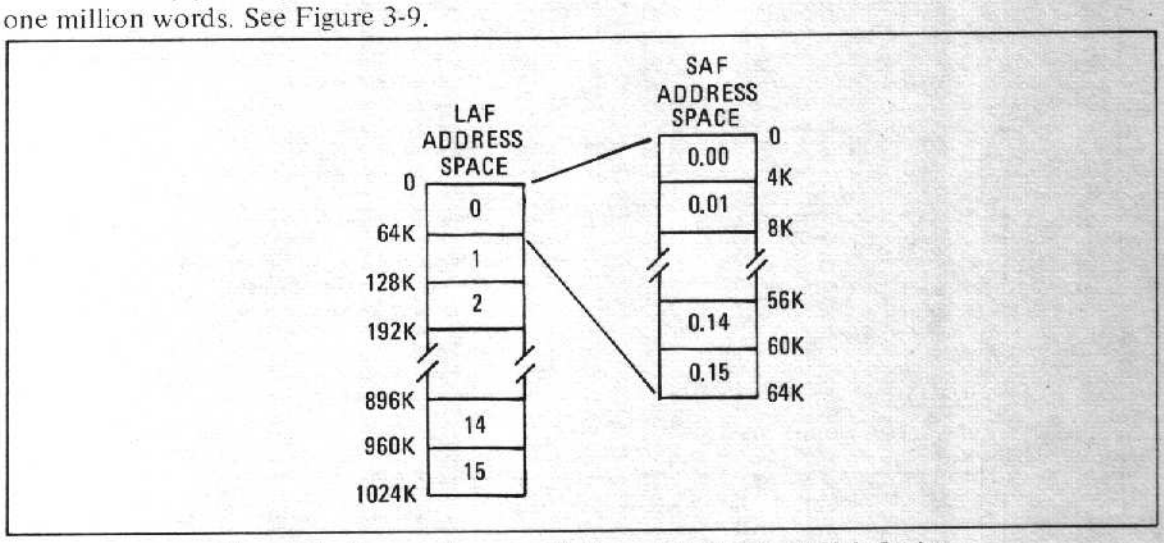


Figure 3-9. Memory Layout with Memory Management Unit Option

Relocation

Through 32-bit "segment descriptors," software can specify the location each segment is to occupy in main memory. This is given as a base (on a 256-word boundary) and a size (in multiples of 256 words). Each segment is relocated independently; segments need not be contiguous, be in order, or have any other particular physical relationship.

Protection

The MMU implements protection using the "ring" approach pioneered by Honeywell's Multics system. At any time, the processor is operating in one of four "rings" of privilege. Each ring, starting with Ring 0 and continuing through Ring 3, is more privileged than the next outer ring. Thus, Ring 0 can be used for the most critical system functions; Ring 1 for less critical system functions; Ring 2 to create a "user supervisor" or for well-checked-out user programs; and Ring 3 for the majority of user software. Programs operating in one ring cannot access code or data reserved for rings of higher privilege.

The MMU controls three forms of access: *read*, *write*, and *execute*. Each form of access is limited to programs operating in a certain ring or lower-numbered ring. Access control is implemented on a segment basis, so each segment has its own protection attributes. Access control can support many different objectives:

- *Data protection* – critical system control data can be placed in a user's address space, so that operating system routines executing on a user's behalf can access it while protecting it from user programs.
- *Source data integrity* – data to be read but not modified can be given read, but not write, access.
- *Software protection* – proprietary routines which a user is permitted to execute can be made inaccessible to "read" operations, thus preventing copying.
- *Program checkout* – by making data segments nonexecutable and code segments nonwritable, many programming errors can be detected quickly.

An additional margin of protection is provided by the segment tables themselves. There is no way a program can attempt to access data not covered by that program's segment descriptors.

Segment Descriptor Format

Each segment is described by a 32-bit segment descriptor (see Figure 3-10).

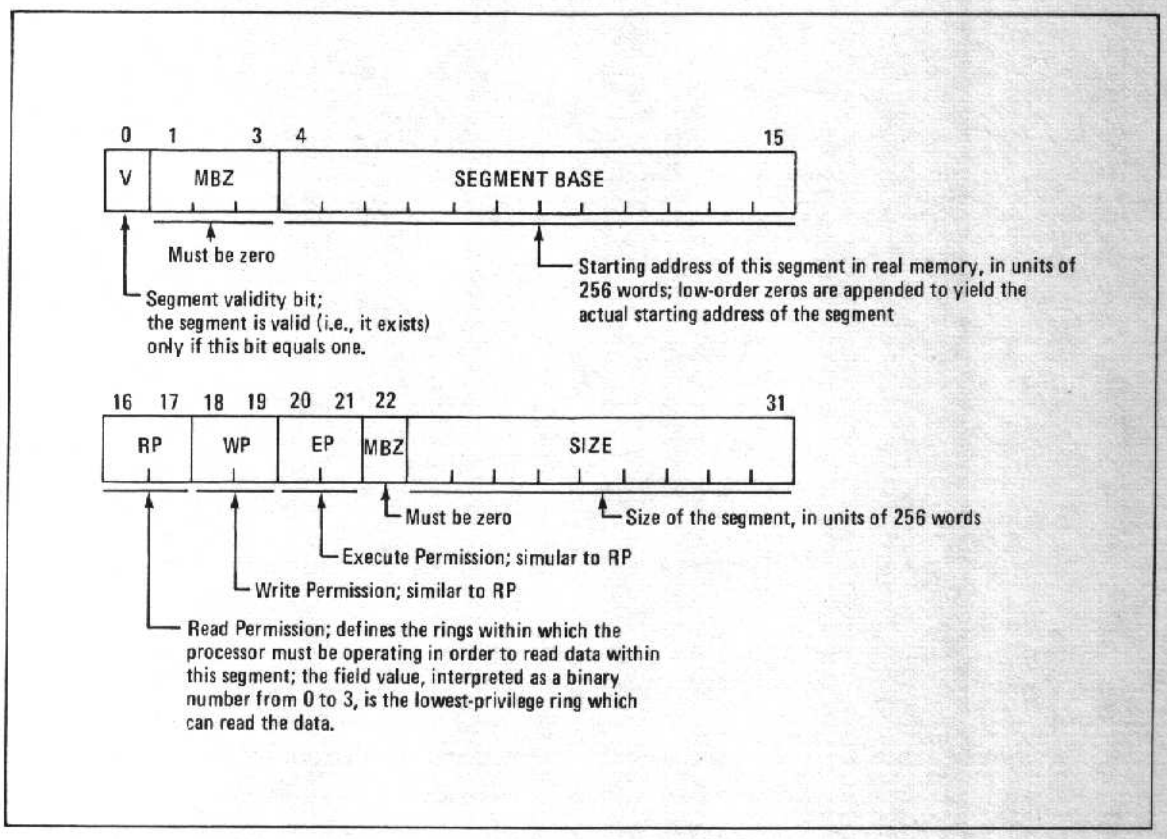


Figure 3-10. Segment Descriptor Format

During program execution, the segment descriptors are contained within the MMU hardware itself. At system initialization, the descriptors are given values which create a "transparent" mode of execution: each virtual address maps into the same real address, and all access modes are permitted in all four rings. Therefore, all programs execute on a system with the MMU exactly as they would on a system without it, until explicit software action changes the MMU tables. This can happen in one of two ways:

- When a level change takes place, if desired. Specifically, bit 0 of the second Interrupt Mask (ISM 2) in the Interrupt Save Area (ISA) of a level determines whether the MMU registers are to be loaded when that level is activated. If this bit is 1, the Address Space Vector in the ISA indicates a 62-word area which contains 31 descriptor images. These images are loaded into the MMU before instruction execution starts at the new level.
- When an Activate Segment Descriptor instruction is executed. This instruction loads a single segment descriptor into the MMU.

Descriptor tables in memory are not referred to during program execution. Therefore, there is no performance degradation with the Memory Management Unit; programs using it operate at the same speed as programs which do not.

As instructions are executed, three types of addressing errors can occur. These are detected by the MMU and result in program traps. They are as follows:

1. Nonexistent segment – an attempt to reference a segment whose validity bit is zero. Causes Trap 15 (unavailable resource).
2. Access out of bounds – an attempt to reference an address beyond the size established by the segment descriptor. Causes Trap 15 (unavailable resource).
3. Protection violation – an attempt to perform a type of access not permitted by the appropriate permission field (RP, WP or EP) in the segment descriptor. Causes Trap 14 (protection violation).

Three instructions invoke MMU functionality, over and above the relocation and protection functions performed on every memory access. They are:

- ASD (Activate Segment Descriptor)
- VLD (Validate)
- LEV (Level Change)

They are described in Section 4 of this handbook.