Concurrent Condition	Type of Ending	PER Event			
		Branch	Instr Fetch	Storage Alter.	GR Alter.
Specification					
Odd instruction address in the PSW	S	No	No	No	No
Instruction access					
First halfword	N or S	No	No	No	No
Second, third halfwords	N or S	No	U	No	No
Specification					
EXECUTE target address odd	S	No	U	No	No
EXECUTE target access	N or S	No	U	No	No
Other nullifying	N	No	Yes	No[1]	No[1]
Other suppressing	S	No	Yes	No[1]	No[1]
Other terminating	T	No	Yes	Yes[2]	Yes[2]
All completing	C	Yes	Yes	Yes	Yes

Explanation:

C The operation, or in the case of the interruptible instructions, the unit of operation is completed.

N The operation or, in the case of the interruptible instructions, the unit of operation is nullified.

S The operation or, in the case of the interruptible instructions, the unit of operation is suppressed.

T The execution of the instruction is terminated.

Yes The PER event is indicated with the other program-interruption condition if the event has occurred; that is, the contents of the designated storage location or general register were altered, or an attempt was made to execute an instruction whose first byte is located in the designated storage area.

No The PER event is not indicated.

U It is unpredictable whether the PER event is indicated.

[1] Although PER events of this type are not indicated for the current unit of operation of an interruptible instruction, PER events of this type that were recognized on completed units of operation of the interruptible instruction are indicated.

[2] This event may be indicated, depending on the model, if the event has not occurred, but would have been indicated if execution had been completed.

Indication of PER Events with Other Concurrent Conditions

Programming Notes

1. The execution of the interruptible instructions MOVE LONG, TEST BLOCK, and COMPARE LOGICAL LONG can cause events for general-register alteration and instruction fetching. Additionally, MOVE LONG can cause the storage-alteration event. The interruption of such an instruction may cause a PER event to be indicated more than once. It may be necessary, therefore, for a program to remove the redundant event indications from the PER data. The following rules govern the indication of the applicable events during execution of these instructions:

a. The instruction-fetching event is indicated whenever the instruction is fetched for execution, regardless of whether it is the initial execution or a resumption.

b. The general-register-alteration event is indicated on the initial execution and on each resumption and does not depend on whether or not the register actually is changed.

c. The storage-alteration event is indicated only when data has been stored in the designated storage area by the portion of the operation starting with the last initiation and ending with the last byte transferred before the interruption. No special indication is provided on premature interruptions as to whether the event will occur again upon the resumption of the operation. When the designated storage area is a single byte location, a storage-alteration event can be recognized only once in the execution of MOVE LONG.

2. The following is an outline of the general action a program must take to delete the redundant entries in the PER data for an interruptible instruction so that only one entry for each complete execution of the instruction is obtained:

   a. Check to see if the PER address is equal to the instruction address in the old PSW and if the last instruction executed was interruptible.

   b. If both conditions are met, delete instruction-fetching and register-alteration events.

   c. If both conditions are met and the event is storage alteration, delete the event if some part of the remaining destination operand is within the designated storage area.

DIRECT CONTROL

The direct-control facility consists of two facilities: (1) a read-write-direct facility, including the two instructions READ DIRECT and WRITE DIRECT and an associated 27-line interface, and (2) an external-signal facility with six signal-in lines. These facilities operate independent of the facilities that perform I/O operations.

READ-WRITE-DIRECT FACILITY

The READ DIRECT and WRITE DIRECT instructions use the 27-line interface to provide timing signals and to transfer a single byte of information, normally for controlling and synchronizing purposes, between CPUs or between a CPU and an external device. The 27 lines are:

Name	Number of Lines	Direction
Write out	1	Output
Read out	1	Output
Hold	1	Input
Signal out	8	Output
Direct out	8	Output
Direct in	8	Input

EXTERNAL-SIGNAL FACILITY

The external-signal facility consists of six signal-in lines and an external-signal mask, which is bit 26 of control register 0. Each of the six signal-in lines, when pulsed, sets up the condition for one of six distinct interruptions (see the section "External Signal" in Chapter 6, "Interruptions").

Note: Some models provide the external-signal facility and not the read-write-direct facility.

For a detailed description, see the System/360 and System/370 Direct Control and External Interruption Features--Original Equipment Manufacturers' Information, GA22-6845.

TIMING

The timing facilities include four facilities for measuring time: the TOD clock, the clock comparator, the CPU timer, and the interval timer.

In a multiprocessing configuration, a single TOD clock may be shared by more than one CPU, or each CPU may have a separate TOD clock. However, each CPU has a separate clock comparator, CPU timer, and interval timer.

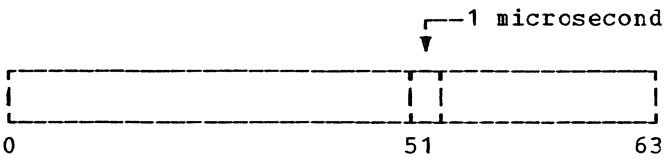
TIME-OF-DAY CLOCK

The time-of-day (TOD) clock provides a high-resolution measure of real time suitable for the indication of date and time of day. The cycle of the clock is approximately 143 years.

In an installation with more than one CPU, each CPU may have a separate TOD clock, or more than one CPU may share a clock, depending on the model. In all cases, each CPU has access to a single clock.

Format

The TOD clock is a binary counter with the format shown in the following illustration. The bit positions of the clock are numbered 0 to 63, corresponding to the bit positions of a 64-bit unsigned binary integer.



In the basic form, the TOD clock is incremented by adding a one in bit position 51 every microsecond. In models having a higher or lower resolution, a different bit position is incremented at such a frequency that the rate of advancing the clock is the same as if a one were added in bit position 51 every microsecond. The resolution of the TOD clock is such that the incrementing rate is comparable to the instruction-execution rate of the model.

When more than one TOD clock exists in the configuration, the stepping rates are synchronized such that all TOD clocks in the configuration are incremented at exactly the same rate.

When incrementing of the clock causes a carry to be propagated out of bit position 0, the carry is ignored, and counting continues from zero. The program is not alerted, and no interruption condition is generated as a result of the overflow.

The operation of the clock is not affected by any normal activity or event in the system. Incrementing of the clock does not depend on whether the wait-state bit of the PSW is one or whether the CPU is in the operating, load, stopped, or check-stop state. Its operation is not affected by CPU, initial-CPU, program, initial-program, or clear resets or by initial program loading. Operation of the clock is also not affected by the setting of the rate control or by an initial-microprogram-loading operation. Depending on the model and the configuration, a TOD clock may or may not be powered independent of a CPU that accesses it.

States

The following states are distinguished for the TOD clock: set, not set, stopped, error, and not operational. The state determines the condition code set by execution of STORE CLOCK. The clock is incremented, and is said to be running, when it is in either the set state or the not-set state.

Not-Set State: When the power for the clock is turned on, the clock is set to zero, and the clock enters the not-set state. The clock is incremented when in the not-set state.

When the clock is in the not-set state, execution of STORE CLOCK causes condition code 1 to be set and the current value of the running clock to be stored.

Stopped State: The clock enters the stopped state when SET CLOCK is executed on a CPU accessing that clock and the clock is set. This occurs when SET CLOCK is executed without encountering any exceptions and any manual TOD-clock control in the configuration is set to the enable-set position. The clock can be placed in the stopped state from the set, not-set, and error states. The clock is not incremented while in the stopped state.

When the clock is in the stopped state, execution of STORE CLOCK on a CPU accessing that clock causes condition code 3 to be set and the value of the stopped clock to be stored.

Set State: The clock enters the set state only from the stopped state. The change of state is under control of the TOD-clock-sync-control bit, bit 2 of control register 0, in the CPU which most recently caused that clock to enter the stopped state. When the bit is zero, or the TOD-clock-synchronization facility is not installed, the clock enters the set state at the completion of execution of SET CLOCK. When the bit is one, it remains in the stopped state until the bit is set to zero on the CPU that placed that clock in the stopped state, until another CPU executes a SET CLOCK instruction affecting the clock, or until any other clock in the configuration is incremented to a value of all zeros in bit positions 32-63. If any clock is set to a value of all zeros in bit positions 32-63 and enters the set state as

the result of a signal from another clock, the updating of bits 32-63 of the two clocks is in synchronism.

Incrementing of the clock begins with the first stepping pulse after the clock enters the set state.

When the clock is in the set state, execution of STORE CLOCK causes condition code 0 to be set and the current value of the running clock to be stored.

Error State: The clock enters the error state when a malfunction is detected that is likely to have affected the validity of the clock value. A timing-facility-damage machine-check-interruption condition is generated on each CPU which has access to that clock whenever it enters the error state.

When STORE CLOCK is executed and the clock accessed is in the error state, condition code 2 is set, and the value stored is unpredictable.

Not-Operational State: The clock is in the not-operational state when its power is off or when it is disabled for maintenance. It depends on the model if the clock can be placed in this state. Whenever the clock enters the not-operational state, a timing-facility-damage machine-check-interruption condition is generated on each CPU that has access to that clock.

When the clock is in the not-operational state, execution of STORE CLOCK causes condition code 3 to be set, and zero is stored.

Changes in Clock State

When the TOD clock accessed by a CPU changes value because of the execution of SET CLOCK or changes state, interruption conditions pending for the clock comparator, CPU timer, interval timer, and TOD-clock sync check may or may not be recognized for up to 1.048576 seconds (2^{20} microseconds) after the change.

Setting and Inspecting the Clock

The clock can be set to a specific value by execution of SET CLOCK if the manual TOD-clock control of any CPU in the configuration is set to the enable-set position. Setting the clock replaces the values in all bit positions from bit position 0 through the rightmost position that is incremented when the clock is running. However, on some models, the rightmost bits starting at or to the right of bit 52 of the specified value are ignored, and zeros are placed in the corresponding positions of the clock.

The TOD clock can be inspected by executing STORE CLOCK, which causes a 64-bit value to be stored. Two executions of STORE CLOCK, possibly on different CPUs in the same configuration, always store different values if the clock is running, or, if separate clocks are accessed, both clocks are running and are synchronized.

The values stored for a running clock always correctly imply the sequence of execution of STORE CLOCK on one or more CPUs for all cases where the sequence can be established by means of the program. Zeros are stored in positions to the right of the bit position that is incremented. In a configuration with more than one CPU, however, when the value of a running clock is stored, nonzero values may be stored in positions to the right of the rightmost position that is incremented. This ensures that a unique value is stored.

In a configuration where more than one CPU accesses the same clock, SET CLOCK is interlocked such that the entire contents appear to be updated at once; that is, if SET CLOCK instructions are issued simultaneously by two CPUs, the final result is either one or the other value. If SET CLOCK is issued on one CPU and STORE CLOCK on the other, the result obtained by STORE CLOCK is either the entire old value or the entire new value. When SET CLOCK is issued by one CPU, a STORE CLOCK issued on another CPU may find the clock in the stopped state even when the TOD-clock-sync-control bit is zero in each CPU. The TOD-clock-sync-control bit is bit 2 of control register 0. Since the clock enters the set state before incrementing, the first STORE CLOCK issued after the clock enters the set state may still find the original value introduced by SET CLOCK.

Programming Notes

1. Bit position 31 of the clock is incremented every 1.048576 seconds; for some applications, reference to the leftmost 32 bits of the clock may provide sufficient resolution.

2. Communication between systems is facilitated by establishing a standard time origin, or standard epoch, which is the calendar date and time to which a clock value of zero corresponds. January 1, 1900, 0 AM Greenwich Mean Time (GMT) is recommended as the

standard epoch for the clock.

3. A program using the clock value as a time-of-day and calendar indication must be consistent with the programming support under which the program is to run. If the programming support uses the standard epoch, bit 0 of the clock remains one through the years 1972-2041. (Bit 0 turned on at 11:56:53.685248 May 11, 1971.) Ordinarily, testing bit 0 for a one is sufficient to determine if the clock value is in the standard epoch.

   In converting to or from the current date or time, the programming support assumes each day to be 86,400 seconds. It does not take into account "leap seconds" inserted or deleted because of time-correction standards.

4. Because of the limited accuracy of manually setting the clock value, the rightmost bit positions of the clock, expressing fractions of a second, are normally not valid as indications of the time of day. However, they permit elapsed-time measurements of high resolution.

5. The following chart shows the time interval between instants at which various bit positions of the TOD clock are stepped. This time value may also be considered as the weighted time value that the bit, when one, represents.

TOD-Clock Bit	Stepping Interval			
	Days	Hours	Minutes	Seconds
51				0.000 001
47				0.000 016
43				0.000 256
39				0.004 096
35				0.065 536
31				1.048 576
27				16.777 216
23			4	28.435 456
19		1	11	34.967 296
15		19	5	19.476 736
11	12	17	25	11.627 776
7	203	14	43	6.044 416
3	3257	19	29	36.710 656

6. The following chart shows the clock setting at the start of various years. The clock settings, expressed in hexadecimal notation, correspond to 0 AM Greenwich Mean Time on January 1 of each year.

Year	Clock Setting (Hex)
1900	0000 0000 0000 0000
1976	8853 BAF0 B400 0000
1980	8F80 9FD3 2200 0000
1984	96AD 84B5 9000 0000
1988	9DDA 6997 FE00 0000
1992	A507 4E7A 6C00 0000
1996	AC34 335C DA00 0000
2000	B361 183F 4800 0000

7. The stepping value of TOD-clock bit position 63, if implemented, is 2^{-12} microseconds, or approximately 244 picoseconds. This value is called a clock unit.

   The following chart shows various time intervals in clock units expressed in hexadecimal notation.

Interval	Clock Units (Hex)
1 microsecond	1000
1 millisecond	3E 8000
1 second	F424 0000
1 minute	39 3870 0000
1 hour	D69 3A40 0000
1 day	1 41DD 7600 0000
365 days	1CA E8C1 3E00 0000
366 days	1CC 2A9E B400 0000
1,461 days[1]	72C E4E2 6E00 0000

[1] Number of days in four years, including a leap year.

8. In a multiprocessing configuration, after the TOD clock is set and begins running, the program should delay activity for 2^{20} microseconds (1.048576 seconds) to ensure that the CPU-timer, clock-comparator, interval timer, and TOD-clock-sync-check interruption conditions are recognized by the CPU.

TOD-CLOCK SYNCHRONIZATION

In an installation with more than one CPU, each CPU may have a separate TOD clock, or more than one CPU may share a TOD clock, depending on the model. In all cases, each CPU has access to a single clock.

The TOD-clock-synchronization facility provides the functions that make it possible to provide, in conjunction with a clock-synchronization program, only one TOD clock, in effect, in a multiprocessing configuration. The result is such that, to all programs storing the clock value, it

appears that all CPUs read the same clock. The TOD-clock-synchronization facility provides these functions in such a way that even though the number of clocks in a multiprocessing configuration is model-dependent, a single model-independent clock-synchronization routine can be written. The following functions are provided:

- Synchronizing the stepping rates for all TOD clocks in the configuration. Thus, if all clocks are set to the same value, they stay in synchronism.
- Comparing the rightmost 32 bits of each clock in the configuration. An unequal condition is signaled by an external interruption with the interruption code 1003 hex, indicating the TOD-clock-sync-check condition.
- Setting a TOD clock to the stopped state.
- Causing a stopped clock, with the TOD-clock-sync-control bit set to one, to start incrementing when bits 32-63 of any running clock in the configuration are incremented to zero. This permits the program to synchronize all clocks to any particular clock without requiring special operator action to select a "master clock" as the source of the clock-synchronization pulses.

Programming Notes

1. TOD-clock synchronization provides for checking and synchronizing only the rightmost bits of the TOD clock. The program must check for synchronization of the leftmost bits and must communicate the leftmost-bit values from one CPU to another in order to correctly set the TOD-clock contents.
2. The TOD-clock-sync-check external interruption can be used to determine the number of TOD clocks in the configuration.

CLOCK COMPARATOR

The clock comparator provides a means of causing an interruption when the TOD-clock value exceeds a value specified by the program.

In a configuration with more than one CPU, each CPU has a separate clock comparator.

The clock comparator has the same format as the TOD clock. In the basic form, the clock comparator consists of bits 0-47, which are compared with the corresponding bits of the TOD clock. In some models, higher resolution is obtained by providing more than 48 bits. The bits in positions provided in the clock comparator are compared with the corresponding bits of the clock. When the resolution of the clock is less than that of the clock comparator, the contents of the clock comparator are compared with the clock value as this value would be stored by executing STORE CLOCK.

The clock comparator causes an external interruption with the interruption code 1004 (hex). A request for a clock-comparator interruption exists whenever either of the following conditions exists:

1. The TOD clock is running and the value of the clock comparator is less than the value in the compared portion of the clock, both values being considered unsigned binary integers. Comparison follows the rules of unsigned binary arithmetic.
2. The TOD clock is in the error state or the not-operational state.

A request for a clock-comparator interruption does not remain pending when the value of the clock comparator is made equal to or greater than that of the TOD clock or when the value of the TOD clock is made less than the clock-comparator value. The latter may occur as a result of the TOD clock either being set or wrapping to zero.

The clock comparator can be inspected by executing the instruction STORE CLOCK COMPARATOR and can be set to a specific value by executing the SET CLOCK COMPARATOR instruction.

The contents of the clock comparator are initialized to zero by initial CPU reset.

Programming Notes

1. An interruption request for the clock comparator persists as long as the clock-comparator value is less than that of the TOD clock or as long as the TOD clock is in the error or not-operational state. Therefore, one of the following actions must be taken after an external interruption for the clock comparator has occurred and before the CPU is again enabled for external interruptions: the value of the clock comparator has to be replaced, the TOD clock has to be set,

the TOD clock has to wrap to zero, or the clock-comparator-subclass mask has to be set to zero. Otherwise, loops of external interruptions are formed.

2. The instruction STORE CLOCK may store a value which is greater than that in the clock comparator, even though the CPU is enabled for the clock-comparator interruption. This is because the TOD clock may be incremented one or more times between when instruction execution is begun and when the clock value is accessed. In this situation, the interruption occurs when the execution of STORE CLOCK is completed.

CPU TIMER

The CPU timer provides a means for measuring elapsed CPU time and for causing an interruption when a specified amount of time has elapsed.

In a configuration with more than one CPU, each CPU has a separate CPU timer.

The CPU timer is a binary counter with a format which is the same as that of the TOD clock, except that bit 0 is considered a sign. In the basic form, the CPU timer is decremented by subtracting a one in bit position 51 every microsecond. In models having a higher or lower resolution, a different bit position is decremented at such a freguency that the rate of decrementing the CPU timer is the same as if a one were subtracted in bit position 51 every microsecond. The resolution of the CPU timer is such that the stepping rate is comparable to the instruction-execution rate of the model.

The CPU timer requests an external interruption with the interruption code 1005 (hex) whenever the CPU-timer value is negative (bit 0 of the CPU timer is one). The request does not remain pending when the CPU-timer value is changed to a nonnegative value.

When both the CPU timer and the TOD clock are running, the stepping rates are synchronized such that both are stepped at the same rate. Normally, decrementing the CPU timer is not affected by concurrent I/O activity. However, in some models the CPU timer may stop during extreme I/O activity and other similar interference situations. In these cases, the time recorded by the CPU timer provides a more accurate measure of the CPU time used by the program than would have been recorded had the CPU timer continued to step.

The CPU timer is decremented when the CPU is in the operating state or the load state. When the manual rate control is set to instruction step, the CPU timer is decremented only during the time in which the CPU is actually performing a unit of operation. However, depending on the model, the CPU timer may or may not be decremented when the TOD clock is in the error, stopped, or not-operational state.

Depending on the model, the CPU timer may or may not be decremented when the CPU is in the check-stop state.

The CPU timer can be inspected by executing the instruction STORE CPU TIMER and can be set to a specific value by executing the SET CPU TIMER instruction.

The CPU timer is set to zero by initial CPU reset.

Programming Notes

1. The CPU timer in association with a program may be used both to measure CPU-execution time and to signal the end of a time interval on the CPU.

2. The time measured for the execution of a sequence of instructions may depend on the effects of such things as I/O interference, page faults, and instruction retry. Hence, repeated measurements of the same sequence on the same installation may differ.

3. The fact that a CPU-timer interruption does not remain pending when the CPU timer is set to a positive value eliminates the problem of an undesired interruption. This would occur if, between the time when the old value is stored and a new value is set, the CPU is disabled for CPU-timer interruptions and the CPU timer value goes from positive to negative.

4. The fact that CPU-timer interruptions are requested whenever the CPU timer is negative (rather than just when the CPU timer goes from positive to negative) eliminates the requirement for testing a value to ensure that it is positive before setting the CPU timer to that value.

   As an example, assume that a program being timed by the CPU timer is interrupted for a cause other than the CPU timer, external interruptions are disallowed by the new PSW, and the CPU-timer value is then saved by STORE CPU TIMER. This value could be negative if the CPU timer went from

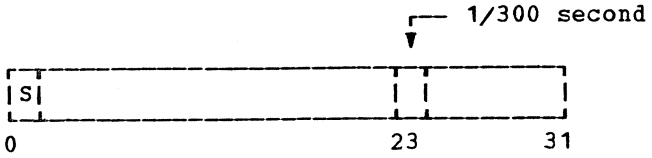
positive to negative since the interruption. Subsequently, when the program being timed is to continue, the CPU timer may be set to the saved value by SET CPU TIMER. A CPU-timer interruption occurs immediately after external interruptions are again enabled if the saved value was negative.

The persistence of the CPU-timer-interruption request means, however, that after an external interruption for the CPU timer has occurred, the value of the CPU timer has to be replaced, the value in the CPU timer has to wrap for its maximum positive value, or the CPU-timer-subclass mask has to be set to zero before the CPU is again enabled for external interruptions. Otherwise, loops of external interruptions are formed.

5. The instruction STORE CPU TIMER may store a negative value even though the CPU is enabled for the interruption. This is because the CPU-timer value may be decremented one or more times between when instruction execution is begun and when the CPU timer is accessed. In this situation, the interruption occurs when the execution of STORE CPU TIMER is completed.

INTERVAL TIMER

The interval timer is a binary counter that occupies a word at real storage location 80 and has the following format:



The interval timer is treated as a 32-bit signed binary integer. In the basic form, the contents of the interval timer are decremented by one in both bit positions 21 and 22 every 1/50 of a second, or the interval timer contents are reduced by one in both bit positions 21 and 23 every 1/60 of a second. Higher resolution of timing may be obtained in some models by counting with higher frequency in other bit positions. In each case, the frequency is adjusted so that bits to the left of bit position 23 change as if bit position 23 were being decremented by one every 1/300 of a second. The cycle of the interval timer is approximately 15.5 hours.

In a configuration with more than one CPU, each CPU has an interval timer.

The interval timer causes an external interruption, with bit 8 of the interruption code set to one and bits 0-7 set to zeros. Bits 9-15 of the interruption code are zeros unless set to ones for another condition that is concurrently indicated.

A request for an interval-timer interruption is generated whenever the interval-timer value is decremented from a positive or zero number to a negative number. The request is preserved and remains pending in the CPU until it is cleared by an interval-timer interruption or a CPU reset. The overflow occurring as the interval-timer value is decremented from a large negative number to a large positive number is ignored.

The interval timer is not necessarily synchronized with the TOD clock.

The interval-timer contents are updated at the appropriate frequency whenever other machine activity permits. The updating occurs only between instruction executions, except that the interval timer may be updated between units of operation of an interruptible instruction, such as MOVE LONG. An updated interval-timer value is normally available at the end of each instruction execution. When the execution of an instruction, I/O data transmission, or other machine activity causes updating to be delayed by more than one period, the contents of the interval timer may be decremented by more than one unit in a single updating cycle. Interval-timer updating may be omitted when such delay is extreme. The program is not alerted when omission of updating causes the real-time count to be lost.

When the contents of the interval timer are fetched by a channel or by another CPU, or when they are used as the source of an instruction, the result is unpredictable. Similarly, storing by a channel, or by another CPU, into the interval timer causes the contents of the interval timer to be unpredictable. This unpredictability is true even for the case of COMPARE AND SWAP or COMPARE DOUBLE AND SWAP when issued by another CPU.

The interval timer is not decremented when the manual interval-timer control is set to disable. The interval timer is also not decremented when the CPU is not in the operating state or when the manual rate control is set to instruction step.

Depending on the model, the interval timer may or may not be decremented when the TOD clock is in the error, stopped, or not-operational state.

When the TOD clock accessed by a CPU is set

or changes state, interruption conditions pending for the interval timer may or may not be recognized for up to 1.048576 seconds after the change.

Programming Notes

1. The value of the interval timer is accessible by fetching the word at real location 80 as an operand, provided the location is not protected against fetching. It may be changed at any time by storing a word at location 80. When real location 80 is protected, any attempt by the program to change the value of the interval timer causes a program interruption for protection exception.

2. The value of the interval timer may be changed without losing the real-time count by storing the new value at real locations 84-87 and then shifting real bytes 80-87 to real locations 76-83 by means of the MOVE (MVC) instruction. Thus, in a single operation, the new interval-timer value is placed at real locations 80-83, and the old value is made available at real locations 76-79.

   If any means other than the instruction MOVE (MVC) are used to interrogate and then replace the value of the interval timer, including MOVE LONG or two separate instructions, the program may lose a time increment when an updating cycle occurs between fetching and storing.

   Logical locations 84-87 are used as the trace-table designation by DAS tracing. If the above means for updating the interval timer by using MOVE are used in a system which also uses DAS tracing, and if location 84 is mapped to real location 84, then the program must restore the contents of the word at real location 84 after updating the interval timer.

3. When the value of the interval timer is to be recorded on an I/O device, the program should first store the interval-timer value in a temporary storage location to which the I/O operation subsequently refers. When the channel fetches the interval-timer value directly from location 80, the value obtained is unpredictable.

EXTERNALLY INITIATED FUNCTIONS

SERVICE SIGNAL

The service-signal facility permits the service processor to communicate with the CPU. Communications to the service processor are model-dependent and are accomplished by means of the DIAGNOSE instruction. When the service processor has completed all or part of a function requested by means of the DIAGNOSE instruction, a service-signal external interruption is generated. The service-signal external interruption is a floating interruption condition and can be accepted by any CPU in the configuration. The service-signal request causes an external interruption with the interruption code 2401 (hex). A 32-bit parameter is also stored at location 128. The subclass mask for service signal is bit 22 of control register 0.

RESETS

Seven reset functions are provided:

- CPU reset
- Initial CPU reset
- Subsystem reset
- Program reset
- Initial program reset
- Clear reset
- Power-on reset

CPU reset provides a means of clearing equipment-check indications and any resultant unpredictability in the CPU state with the least amount of information destroyed. In particular, it is used to clear check conditions when the CPU state is to be preserved for analysis or resumption of the operation.

Initial CPU reset provides the functions of CPU reset together with initialization of the current PSW, CPU timer, clock comparator, prefix, and control registers.

Subsystem reset provides a means for clearing floating interruption conditions and for initializing channel-set connections as well as for invoking

I/O-system reset.

Program reset and initial program reset cause CPU reset and initial CPU reset, respectively, to be performed and cause I/O-system reset to be performed (see the section "I/O-System Reset" in Chapter 12, "Input/Output Operations").

Clear reset causes initial CPU reset and subsystem reset to be performed and, additionally, clears or initializes all storage locations and registers in all CPUs in the configuration, with the exception of the TOD clock. Such clearing is useful in debugging programs and in ensuring user privacy. Clearing does not affect external storage, such as direct-access storage devices used by the control program to hold the contents of unaddressable pages.

The power-on-reset sequences for the TOD clock, main storage, and channels may be included as part of the CPU power-on sequence, or the power-on sequence for these units may be initiated separately.

CPU reset, initial CPU reset, subsystem reset, and clear reset may be initiated manually by using the operator facilities (see Chapter 13, "Operator Facilities"). Initial CPU reset is part of the initial-program-loading function. The figure "Manual Initiation of Resets" summarizes how these four resets are manually initiated. Power-on reset is performed as part of turning power on. The reset actions are tabulated in the figure "Summary of Reset Actions." For information concerning what resets can be performed by the SIGNAL PROCESSOR instruction, see the section "Signal-Processor Orders" in this chapter.

Key Activated	Function Performed on[1]: CPU on which Key was Activated	Other CPUs in Config	Remainder of Configuration
System-reset-normal key			
• without store-status facility	Initial CPU reset	*	Subsystem reset
• with store-status facility	CPU reset	CPU reset	Subsystem reset
System-reset-clear key	Clear reset[2]	Clear reset[2]	Clear reset[3]
Load-normal key	Initial-CPU reset, followed by IPL	CPU reset	Subsystem reset
Load-clear key	Clear reset[2], followed by IPL	Clear reset[2]	Clear reset[3]

Explanation:

* This situation cannot occur, since the store-status facility is provided in a CPU equipped for multiprocessing.

[1] Activation of a system-reset or load key may change the configuration, including the connection with I/O, storage units, and other CPUs.

[2] Only the CPU elements of this reset apply.

[3] Only the non-CPU elements of this reset apply.

Manual Initiation of Resets

Area Affected	Reset Function						
	Sub-system Reset	CPU Reset	Program Reset	Initial CPU Reset	Initial Program Reset	Clear Reset	Power-on Reset
CPU	U	S	S	S[1]	S	S[1]	S
PSW[2]	U	U/V	U/V	C*[1]	C*	C*[1]	C*
Prefix	U	U/V	U/V	C	C	C	C
CPU timer	U	U/V	U/V	C	C	C	C
Clock comparator	U	U/V	U/V	C	C	C	C
Control registers	U	U/V	U/V	I	I	I	I
General registers	U	U/V	U/V	U/V	U/V	C/V	C/X
Floating-point registers	U	U/V	U/V	U/V	U/V	C/V	C/X
Storage keys	U	U	U	U	U	C	C/X[4]
Volatile main storage	U	U	U	U	U	C	C/X[4]
Nonvolatile main storage	U	U	U	U	U	C	U
TOD clock	U[3]	U[3]	U[3]	U[3]	U[3]	U[3]	T[4]
Channel-set connection	I	U	U	U	U	I	I[5]
Floating interruption conditions	C	U	U	U	U	C	C[4]
Channels in the configuration	RA	U	RC	U	RC	RA	RA[4]

Explanation:

S The CPU is reset; current operations, if any, are terminated; the TLB is cleared of entries; interruption conditions in the CPU are cleared; and the CPU is placed in the stopped state.

RA I/O-system reset is performed in all the channels in the configuration and pending I/O-interruption conditions are cleared. As part of this reset, system reset is signaled to the I/O control units and devices attached to the channels being reset.

RC I/O-system reset is performed in those channels connected to the CPU performing the program reset or initial-program reset. As part of this reset, system reset is signaled to the I/O control units and devices attached to the channels being reset.

U The state, condition, or contents of the field remain unchanged. However, the resulting value is unpredictable if an operation is in progress that changes the state, condition, or contents of the field at the time of reset.

U/V The contents remain unchanged, provided the field is not being accessed at the time the reset function is performed. However, on some models the checking-block code of the contents may be made valid. The subsequent contents of a field are unpredictable if it is accessed at the time of the reset.

C The condition or contents are cleared. If the area affected is a field, the contents are cleared to zero with valid checking-block code.

C/V The checking-block code of the contents is made valid. The contents normally are cleared to zeros but in some models may be left unchanged.

C/X The checking-block code of the contents is made valid. The contents normally are cleared to zeros but in some models may be left unpredictable.

I The state or contents are initialized. If the area affected is a field, the contents are set to their initial values with valid checking-block code.

T The TOD clock is initialized to zero and validated; it enters the not-set state.

Summary of Reset Actions (Part 1 of 2)

Explanation (Continued):

* Clearing the contents of the PSW to zero causes the CPU to assume the BC-mode format.

1 When the IPL sequence follows the reset function on that CPU, the CPU does not enter the stopped state, and the PSW is not necessarily cleared to zeros.

2 For a BC-mode PSW, the ILC and interruption-code fields are unpredictable in the current PSW.

3 Access to the TOD clock by means of STORE CLOCK at the time a reset function is performed does not cause the value of the TOD clock to be affected.

4 When these units are separately powered, the action is performed only when the power for the unit is turned on.

5 When these units are separately powered, the action is model-dependent.

Summary of Reset Actions (Part 2 of 2)

CPU Reset

CPU reset causes the following actions:

1. The execution of the current instruction or other processing sequence, such as an interruption, is terminated, and all program-interruption and supervisor-call-interruption conditions are cleared.
2. Any pending external-interruption conditions which are local to the CPU are cleared. Floating external-interruption conditions are not cleared.
3. Any pending machine-check-interruption conditions and error indications which are local to the CPU and any check-stop states are cleared. Floating machine-check-interruption conditions are not cleared. A broadcast machine check which has been made pending to a CPU is said to be local to the CPU.
4. All copies of prefetched instructions or operands are cleared. Additionally, any results to be stored because of the execution of instructions in the current checkpoint interval are cleared.
5. The translation-lookaside buffer is cleared of entries.
6. The CPU is placed in the stopped state after actions 1-5 have been completed. When the IPL sequence follows the reset function on that CPU, the CPU enters the load state at the completion of the reset function and does not necessarily enter the stopped state during the execution of the reset operation.

Registers, storage contents, and the state of conditions external to the CPU remain unchanged by CPU reset. However, the subsequent contents of the register, location, or state are unpredictable if an operation is in progress that changes the contents at the time of the reset.

When the reset function in the CPU is initiated at the time the CPU is executing an I/O instruction or is performing an I/O interruption, the current operation between the CPU and the channel may or may not be completed, and the resultant state of the associated channel may be unpredictable.

Programming Note

Most operations which would change a state, a condition, or the contents of a field cannot occur when the CPU is in the stopped state. However, some signal-processor functions and some operator functions may change these fields. To eliminate the possibility of losing a field when CPU reset is issued, the CPU should be stopped, and no operator functions should be in progress.

Initial CPU Reset

Initial CPU reset combines the CPU reset functions with the following clearing and initializing functions:

1. The contents of the current PSW,

prefix, CPU timer, and clock comparator are set to zero. When the IPL sequence follows the reset function on that CPU, the PSW is not necessarily cleared to zeros.

2. All assigned control-register positions are set to their initial values.

These clearing and initializing functions include validation.

Setting the current PSW tc zero causes the PSW to assume the BC-mode format. The instruction-length code and interruption code are unpredictable, because these values are not retained when a new PSW is introduced.

Subsystem Reset

Subsystem reset operates only on those elements in the configuration which are not CPUs. It performs the following actions:

1. I/O-system reset is performed in each channel in the configuration.

2. All floating interruption conditions in the configuration are cleared.

3. Channel-set connections are initialized to connect each channel set to its home CPU if one exists, is operational, and is in the configuration, or else to make the channel set disconnected.

As part of I/O-system reset, pending I/O-interruption conditions are cleared, and system reset is signaled to all control units and devices attached to the channel (see the section "I/O-System Reset" in Chapter 12, "Input/Output Operations"). The effect of system reset on I/O control units and devices and the resultant control-unit and device state are described in the appropriate SL publication for the control unit or device. A system reset, in general, resets only those functions in a shared control unit or device that are associated with the particular channel signaling the reset.

Program Reset

For program reset, CPU reset is performed, and I/O-system reset is performed in each channel connected to this CPU.

Initial Program Reset

Initial program reset combines the program-reset functions with the clearing and initializing functions of initial CPU reset.

Clear Reset

Clear reset combines the initial-CPU-reset function with an initializing function which causes the following actions:

1. In most models, the contents of the general and floating-point registers are set to zero, but in some models the contents may be left unchanged except that the checking-block code is made valid.

2. The contents of the main storage and the storage keys in the configuration are set to zero with valid checking-block code.

3. A subsystem reset is performed.

Validation is included in setting registers and in clearing storage.

Programming Notes

1. For the CPU-reset cr program-reset operation not to affect the contents of fields that are to be left unchanged, the CPU must not be executing instructions and must be disabled for all interruptions at the time of the reset. Except for the operation of the TOD clock, interval timer, and CPU timer and for the possibility of taking a machine-check interruption, all CPU activity can be quiesced by placing the CPU in the wait state and by disabling it for I/O and external interruptions. To avoid the possibility of causing a reset at the time the timing facilities are being updated or a machine-check interruption occurs, the CPU must be in the stopped state.

2. CPU reset, initial CPU reset, program reset, initial program reset, and clear reset do not affect the value and state of the TOD clock.

3. The conditions under which the CPU enters the check-stop state are model-dependent and include malfunctions that preclude the completion of the current operation.

Hence, if CPU reset, initial CPU reset, program reset, or initial program reset is executed while the CPU is in the check-stop state, the contents of the PSW, registers, and storage locations, including the storage keys and the storage location accessed at the time of the error, may have unpredictable values, and, in some cases, the contents may still be in error after the check-stop state is cleared by these resets. In such a case, a clear reset is required to clear the error.

4. Clear reset causes all bit positions of the interval timer to be cleared to zeros.

Power-On Reset

The power-on-reset function for a component of the system is performed as part of the power-on sequence for that component.

The power-on sequences for the TOD clock, main storage, and channels may be included as part of the CPU power-on sequence, or the power-on sequence for these units may be initiated separately. The following sections describe the power-on resets for the CPU, TOD clock, main storage, and channels. See also Chapter 12, "I/O Operations," and the appropriate System Library (SL) publication for channels, control units, and I/O devices.

CPU Power-On Reset: The power-on reset causes initial CPU reset to be performed and may or may not cause I/O-system reset to be performed in the channels connected to the CPU. The contents of general registers and floating-point registers normally are cleared to zeros, but in some models may be left unpredictable, with valid checking-block code.

TOD-Clock Power-On Reset: The power-on reset causes the value of the TOD clock to be set to zero and causes the clock to enter the not-set state.

Main-Storage Power-On Reset: For volatile main storage (one that does not preserve its contents when power is off) and for storage keys, power-on reset causes valid checking-block code to be placed in these fields. In most models, the contents are cleared to zeros, but, in some models, the contents may be left unpredictable except for the checking-block code. The contents of nonvolatile main storage, including the checking-block code, remain unchanged.

Channel Power-On Reset: The channel power-on reset causes I/O-system reset to be performed. (See the section "I/O-System Reset" in Chapter 12, "Input/Output Operations.")

INITIAL PROGRAM LOADING

Initial program loading (IPL) is provided to initiate processing when the contents of storage or of the PSW are not suitable for processing.

Initial program loading is initiated manually by designating an input device with the load-unit-address controls and subsequently activating the load-normal or load-clear key. The load-normal key causes an initial-CPU-reset and a subsystem-reset operation to be performed, and the load-clear key causes a clear-reset operation to be performed. Other CPUs in the configuration perform CPU reset and clear reset, respectively. The CPU enters the load state. Subsequently, a read operation is initiated from the selected input device. The CPU does not necessarily enter the stopped state during the execution of the reset operation. The load indicator is on while the CPU is in the load state.

The read operation is performed as if a START I/O instruction were executed that specified the channel, subchannel, and I/O device designated by the load-unit-address controls. The operation uses an implied channel-address word (CAW) containing a subchannel key of zero, a suspend-control bit of zero, and a channel-command-word (CCW) address of 0, but the CAW at real location 72 is not accessed. The load-unit-address controls provide the 12 rightmost bits of the I/O address; zeros are implied for the leftmost bits.

Although the absolute location of the first CCW to be executed is specified by the CCW address as 0, the first CCW actually executed is an implied CCW, containing, in effect, a read command with the modifier bits set to zeros, a data address of 0, a byte count of 24, the chain-command flag set to one, the SLI flag set to one, the chain-data flag set to zero, the skip flag set to zero, the indirect-data-address flag set to zero, the suspend flag set to zero, and the PCI flag set to zero. The CCW fetched, as a result of command chaining, from absolute location 8 or 16, as well as any subsequent CCW in the IPL sequence, is interpreted the same as a CCW in any I/O operation, except that any PCI flags that are specified in CCWs used for the IPL sequence are ignored.

When the I/O device provides channel-end status for the last operation of the IPL chain and no exceptional conditions are detected in the operation, a new PSW is obtained from storage locations 0-7. When this PSW specifies the EC mode, the I/O address that was used for the IPL operation is stored at locations 186-187, and zeros are stored at location 185; when the BC mode is specified, the I/O address is stored at locations 2-3. The CPU leaves the load state and enters the operating state, with CPU operation proceeding under the control of the new PSW, provided the rate control is set to process; if the rate control is set to instruction step, the CPU enters the stopped state after the new PSW has been obtained.

When channel-end status for the IPL operation is presented, either separate from or along with device-end status, no I/O-interruption condition is generated. Similarly, any PCI flags specified by the program in the CCWs used for the IPL sequence are ignored. If the device-end status for the IPL operation is provided separately after channel-end status, it causes an I/O interruption condition to be generated.

If the IPL I/O operation or the PSW loading is not completed satisfactorily, the CPU remains in the load state, and the load indicator remains on. This occurs when the device designated by the load-unit-address controls is not operational, when the device or channel signals any condition other than channel end, device end, or status modifier during or at the completion of the IPL I/O operation, or when the PSW loaded from location 0 has a PSW-format error that is recognized during the loading procedure. The address of the I/O device used in the IPL operation is not stored. The contents of storage locations 0-7 are unpredictable. The contents of other storage locations remain unchanged, except possibly for those locations due to be changed by the read operations.

When fewer than eight bytes are read into locations 0-7, the PSW fetched from location 0 at the conclusion of the IPL operation is unpredictable.

Programming Notes

1. The information read and placed at locations 8-15 and 16-23 may be used as CCWs for reading additional information during the IPL sequence: the CCW at location 8 may specify reading additional CCWs elsewhere in storage, and the CCW at location 16 may specify the transfer-in-channel command, causing transfer to these CCWs.

   The status-modifier bit has its normal effect during the IPL operation, causing the channel to fetch and chain to the CCW whose address is 16 higher than that of the current CCW. This applies also to the initial chaining that occurs after completion of the read operation specified by the implicit CCW.

   The PSW that is loaded at the completion of the IPL procedure may be provided by the first eight bytes of the IPL I/C operation or may be placed at locations 0-7 by a subsequent CCW.

2. When the PSW in location 0 has bit 14 set to one, the CPU is placed in the wait state after the IPL procedure is completed; at that point, the load and manual indicators are off, and the wait indicator is on.

3. Activating the load-normal key permits an IPL program to be loaded with a minimum disturbance of storage contents. This function may be useful in debugging. When the power is turned on or the load-clear key is activated, the IPL program starts with a cleared machine in a known state, except that information on external storage remains unchanged.

STORE STATUS

The store-status facility includes:

1. A change to the operation of the system-reset-normal key. With the store-status facility installed, activating the system-reset-normal key causes a CPU-reset operation and a subsystem-reset operation to be performed; without this facility, an initial-CPU-reset operation and subsystem-reset operation are performed.

2. An operator-initiated store-status function.

The store-status operation places the contents of the CPU registers, except for the TOD clock, in assigned storage locations.

The figure "Assigned Storage Locations for Store Status" lists the fields that are stored, their length, and their location in main storage.

Field	Length in Bytes	Absolute Address
CPU timer*	8	216
Clock comparator*	8	224
Current PSW+	8	256
Prefix*	4	264
Model-dependent feat.*	4	268
Fl-pt registers 0-6*	32	352
General registers 0-15	64	384
Control registers 0-15	64	448

Explanation

* If the facility is not installed, the contents of the field in storage remain unchanged.

\+ In the BC mode, the ILC is unpredictable, and the interruption code is stored as zeros.

Assigned Storage Locations for Store Status

In the BC mode, the instruction-length code in the PSW is unpredictable, and an interruption code of zero is stored. The information provided for uninstalled or unassigned control-register positions is unpredictable. If the CPU timer, clock comparator, prefix register or floating-point facility is not installed, the contents of the corresponding locations in storage remain unchanged.

The word beginning at absolute location 268 is reserved for storing additional status as required by certain model-dependent facilities. If no facility requiring this location is installed, the contents of the field remain unchanged upon execution of the store-status function.

The contents of the registers are not changed. If an error is encountered during the operation, the CPU enters the check-stop state.

The store-status operation can be initiated manually by use of the store-status key (see Chapter 13, "Operator Facilities"). The store-status operation can also be initiated at the addressed CPU by executing SIGNAL PROCESSOR, specifying the stop-and-store-status order.

MULTIPROCESSING

The multiprocessing facility provides for the interconnection of CPUs, via a common main storage, in order to enhance system availability and to share data and resources. The multiprocessing facility includes the following facilities:

- Shared main storage
- Prefixing
- CPU-address identification
- CPU signaling and response
- TOD-clock synchronization

TOD-clock synchronization is described earlier in this chapter. Prefixing is described in Chapter 3, "Storage." Shared main storage, CPU-address identification, and CPU signaling and response are described in the sections which follow.

Associated with these facilities are four extensions to the external interruption (external call, emergency signal, TOD-clock-sync check, and malfunction alert), which are described in Chapter 6, "Interruptions"; control-register positions for the TOD-clock-sync-control bit and for the masks for the external-interruption conditions, which are listed in the section "Control Registers" in this chapter; and the instructions SET PREFIX, SIGNAL PROCESSOR, STORE CPU ADDRESS, and STORE PREFIX, which are described in Chapter 10, "Control Instructions."

Channels in a multiprocessing configuration are connected to a particular CPU. Only that CPU which is connected to a channel can initiate I/O operations at that channel, and all interruption conditions are directed to that CPU. When channel-set switching is installed, the channel-CPU connection can be changed by means of the program.

SHARED MAIN STORAGE

The shared-main-storage facility permits more than one CPU to have access to common main-storage locations. All CPUs having access to a common main-storage location have access to the entire 2K-byte block containing that location and to the associated storage key. When the storage-key 4K-byte-block facility is installed, all CPUs having access to a common main-storage location have access to the entire 4K-byte block containing that location and to the associated single key or double keys

in that block. All CPUs and all channels refer to a shared main-storage location using the same absolute address.

CPU-ADDRESS IDENTIFICATION

Each CPU in a multiprocessing configuration has a number assigned, called its CPU address. A CPU address uniquely identifies one CPU within a configuration. The CPU is designated by specifying this address in the CPU-address field of SIGNAL PROCESSOR. The CPU signaling a malfunction alert, emergency signal, or external call is identified by storing this address in the CPU-address field with the interruption. The CPU address is assigned during system installation and is not changed as a result of reconfiguration changes. The program can determine the address of the CPU by using STORE CPU ADDRESS.

CPU SIGNALING AND RESPONSE

The CPU-signaling-and-response facility consists of SIGNAL PROCESSOR and a mechanism to interpret and act on several order codes. The facility provides for communications among CPUs, including transmitting, receiving, and decoding a set of assigned order codes; initiating the specified operation; and responding to the signaling CPU. If a CPU has the CPU-signaling-and-response facility installed, it can address SIGNAL PROCESSOR to itself. SIGNAL PROCESSOR is described in Chapter 10, "Control Instructions."

SIGNAL-PROCESSOR ORDERS

The signal-processor orders are specified in bit positions 24-31 of the second-operand address of SIGNAL PROCESSOR and are encoded as shown in the figure "Encoding of Orders."

Code	Order
00	Unassigned
01	Sense
02	External call
03	Emergency signal
04	Start
05	Stop
06	Restart
07	Initial program reset
08	Program reset
09	Stop and store status
0A	Initial microprogram load
0B	Initial CPU reset
0C	CPU reset
0D-FF	Unassigned

Encoding of Orders

The orders are defined as follows:

Sense: The addressed CPU presents its status to the issuing CPU (see the section "Status Bits" in this chapter for a definition of the bits). No other action is caused at the addressed CPU. The status, if not all zeros, is stored in the general register designated by the R_1 field, and condition code 1 is set; if all status bits are zeros, condition code 0 is set.

External Call: An external-call external-interruption condition is generated at the addressed CPU. The interruption condition becomes pending during the execution of SIGNAL PROCESSOR. The associated interruption occurs when the CPU is enabled for that condition and does not necessarily occur during the execution of SIGNAL PROCESSOR. The address of the CPU sending the signal is provided with the interruption code when the interruption occurs. Only one external-call condition can be kept pending in a CPU at a time. The order is effective only when the addressed CPU is in the stopped or operating state.

Emergency Signal: An emergency-signal external-interruption condition is generated at the addressed CPU. The interruption condition becomes pending during the execution of SIGNAL PROCESSOR. The associated interruption occurs when the CPU is enabled for that condition and does not necessarily occur during the execution of SIGNAL PROCESSOR. The address of the CPU sending the signal is provided with the interruption code when the interruption occurs. At any one time the receiving CPU can keep pending one emergency-signal condition for each CPU in the configuration, including the receiving CPU itself. The order is effective only when the addressed CPU is in the stopped or

operating state.

Start: The addressed CPU performs the start function (see the section "Stopped, Operating, Load, and Check-Stop States" in this chapter). The CPU does not necessarily enter the operating state during the execution of SIGNAL PROCESSOR. The order is effective only when the addressed CPU is in the stopped state, and the effect is unpredictable when the stopped state has been entered by reset.

Stop: The addressed CPU performs the stop function (see the section "Stopped, Operating, Load, and Check-Stop States" in this chapter). The CPU does not necessarily enter the stopped state during the execution of SIGNAL PROCESSOR. The order is effective only when the CPU is in the operating state.

Restart: The addressed CPU performs the restart operation (see the section "Restart Interruption" in Chapter 6, "Interruptions"). The CPU does not necessarily perform the operation during the execution of SIGNAL PROCESSOR. The order is effective only when the addressed CPU is in the stopped or operating state.

Initial Program Reset: The addressed CPU performs initial program reset (see the section "Resets" in this chapter). The execution of the reset does not affect other CPUs. The reset operation is not necessarily completed during the execution of SIGNAL PROCESSOR.

Program Reset: The addressed CPU performs program reset (see the section "Resets" in this chapter). The execution of the reset does not affect other CPUs. The reset operation is not necessarily completed during the execution of SIGNAL PROCESSOR.

Stop and Store Status: The addressed CPU performs the stop function, followed by the store-status function (see the section "Store Status" in this chapter). The CPU does not necessarily complete the operation, or even enter the stopped state, during the execution of SIGNAL PROCESSOR. The order is effective only when the addressed CPU is in the stopped or operating state.

Initial Microprogram Load (IML): The addressed CPU performs initial program reset and then initiates the IML function. The IML function is the same as that which is performed as part of manual initial microprogram loading. If the IML function is not provided on the addressed CPU, the order code is treated as unassigned and invalid. The operation is not necessarily completed during the execution of SIGNAL PROCESSOR.

Initial CPU Reset: The addressed CPU performs initial CPU reset (see the section "Resets" in this chapter). The execution of the reset does not affect other CPUs and does not cause I/O to be reset. If the initial-CPU-reset order is not provided on the addressed CPU, the order is treated as unassigned and invalid. The reset operation is not necessarily completed during the execution of SIGNAL PROCESSOR.

CPU Reset: The addressed CPU performs CPU reset (see the section "Resets" in this chapter). The execution of the reset does not affect other CPUs and does not cause I/O to be reset. If the CPU-reset order is not provided on the addressed CPU, the order is treated as unassigned and invalid. The reset operation is not necessarily completed during the execution of SIGNAL PROCESSOR.

CONDITIONS DETERMINING RESPONSE

Conditions Precluding Interpretation of the Order Code

The following situations preclude the initiation of the order. The sequence in which the situations are listed is the order of priority for indicating concurrently existing situations:

1. The access path to the addressed CPU is busy because a concurrently issued SIGNAL PROCESSOR is using the CPU-signaling-and-response facility. The concurrently issued instruction may or may not have been issued by or to the addressed CPU and may or may not have been issued to this CPU. The order is rejected. Condition code 2 is set.

2. The addressed CPU is not operational; that is, the addressed CPU is not installed, is not in the configuration, is in certain customer-engineer test modes, or does not have power on. The order is rejected. Condition code 3 is set. This condition cannot arise as a result of a SIGNAL PROCESSOR by a CPU addressing itself.

3. One of the following conditions exists at the addressed CPU:

   a. A previously issued start, stop, restart, or stop-and-store-status order has been accepted by the addressed CPU, and execution of the function requested by the

order has not yet been completed.

b. A manual start, stop, restart, or store-status function has been initiated at the addressed CPU, and the function has not yet been completed. This condition cannot arise as a result of a SIGNAL PROCESSOR by a CPU addressing itself.

c. A manual initial-program-load function has been initiated at the addressed CPU, and the reset portion, but not the program-load portion, of the function has been completed. This condition cannot arise as a result of a SIGNAL PROCESSOR by a CPU addressing itself.

If the currently specified order is sense, external call, emergency signal, start, stop, restart, or stop-and-store-status, then the order is rejected, and condition code 2 is set. If the currently specified order is an IML, one of the reset orders, or an unassigned or not-implemented order, the order code is interpreted as described in the section "Status Bits" in this chapter.

4. One of the following conditions exists at the addressed CPU:

a. A previously issued initial-program-reset, program-reset, IML, initial-CPU-reset, or CPU-reset order has been accepted by the addressed CPU, and execution of the function requested by the order has not yet been completed.

b. A manual-reset or IML function has been initiated at the addressed CPU, and the function has not yet been completed. The term "manual-reset function" includes the reset portion of IPL. This condition cannot arise as a result of a SIGNAL PROCESSOR by a CPU addressing itself.

If the currently specified order is sense, external call, emergency signal, start, stop, restart, or stop-and-store-status, then the order is rejected, and condition code 2 is set. If the currently specified order is an IML, one of the reset orders, or an unassigned or not-implemented order, either the order is rejected and condition code 2 is set or the order code is interpreted as described in the section "Status Bits" in this chapter.

When any of the conditions described in items 3 and 4 exists, the addressed CPU is referred to as "busy." Busy is not indicated if the addressed CPU is in the check-stop state or when the operator-intervening condition exists. A CPU-busy condition is normally of short duration; however, the conditions described in item 3 may last indefinitely because of a string of interruptions, because of an incomplete READ DIRECT operation, or because of an invalid address in the prefix register. In this situation, however, the CPU does not appear busy to any of the reset orders or to an IML.

When the conditions described in items 1 and 2 above do not apply and operator-intervening and receiver-check status conditions do not exist at the addressed CPU, reset orders may be accepted regardless of whether the addressed CPU has completed a previously accepted order. This may cause the previous order to be lost when it is only partially completed, making unpredictable whether the results defined for the lost order are obtained.

Status Bits

Various status conditions are defined whereby the issuing and addressed CPUs can indicate their response to the designated order. The status conditions and their bit positions in the general register designated by the R_1 field of the SIGNAL PROCESSOR instruction are shown in the figure "Status Conditions."

Bit Position	Status Condition
0	Equipment check
1-23	Unassigned; zeros stored
24	External-call pending
25	Stopped
26	Operator intervening
27	Check stop
28	Not ready
29	Unassigned; zero stored
30	Invalid order
31	Receiver check

Status Conditions

The status condition assigned to bit position 0 is generated by the CPU executing SIGNAL PROCESSOR. The remaining status conditions are generated by the addressed CPU.

When the equipment-check condition exists, bit 0 of the general register designated by the R_1 field of the SIGNAL PROCESSOR

instruction is set to one, unassigned bits of the status register are set to zeros, and the contents of other status bits are unpredictable. In this case, condition code 1 is set independent of whether the access path to the addressed CPU is busy and independent of whether the addressed CPU is not operational, is busy, or has presented zero status.

When the access path to the addressed CPU is not busy and the addressed CPU is operational and does not indicate busy to the currently specified order, the addressed CPU presents its status to the issuing CPU. These status bits are of two types:

1. Status bits 24-28 indicate the presence of the corresponding conditions in the addressed CPU at the time the order code is received. Except in response to the sense order, each condition is indicated only when the condition precludes the successful execution of the designated order. In the case of sense, all existing status conditions are indicated; the operator-intervening and not-ready conditions each are indicated if these conditions preclude the execution of any installed order.

2. Status bits 30 and 31 indicate that the corresponding conditions were detected by the addressed CPU during reception of the order.

If the presented status is all zeros, the addressed CPU has accepted the order, and condition code 0 is set at the issuing CPU; if the presented status is not all zeros, the order has been rejected, the status is stored at the issuing CPU in the general register designated by the R_1 field of the SIGNAL PROCESSOR instruction, zeros are stored in the unassigned bit positions of the register, and condition code 1 is set.

The status conditions are defined as follows:

Equipment Check: This condition exists when the CPU executing the instruction detects equipment malfunctioning that has affected only the execution of this instruction and the associated order. The order code may or may not have been transmitted and may or may not have been accepted, and the status bits provided by the addressed CPU may be in error.

External Call Pending: This condition exists when an external-call interruption condition is pending in the addressed CPU because of a previously issued SIGNAL PROCESSOR. The condition exists from the time an external-call order is accepted until the resultant external interruption has been completed or a CPU reset occurs. The condition may be due to the issuing CPU or another CPU. The condition, when present, is indicated only in response to sense and to external call.

Stopped: This condition exists when the addressed CPU is in the stopped state. The condition, when present, is indicated only in response to sense. This condition cannot be reported as a result of a SIGNAL PROCESSOR by a CPU addressing itself.

Operator Intervening: This condition exists when the addressed CPU is executing certain operations initiated from local or remote operator facilities. The particular manually initiated operations that cause this condition to be present depend on the model and on the order specified. On machines which do not implement the IML order, the conditions described under "Not Ready" may be indicated as an operator-intervening condition. The operator-intervening condition, when present, can be indicated in response to all orders. Operator intervening is indicated in response to sense if the condition is present and precludes the acceptance of any of the installed orders. The condition may also be indicated in response to unassigned or uninstalled orders. This condition cannot arise as a result of a SIGNAL PROCESSOR by a CPU addressing itself.

Check Stop: This condition exists when the addressed CPU is in the check-stop state. The condition, when present, is indicated only in response to sense, external call, emergency signal, start, stop, restart, and stop and store status. The condition may also be indicated in response to unassigned or uninstalled orders. This condition cannot be reported as a result of a SIGNAL PROCESSOR by a CPU addressing itself.

Not Ready: This condition exists when the addressed CPU uses reloadable control storage to perform an order and the required microprogram is not loaded. The not-ready condition may be indicated in response to all orders except IML. This condition cannot arise as a result of a SIGNAL PROCESSOR by a CPU addressing itself.

Invalid Order: This condition exists during the communications associated with the execution of SIGNAL PROCESSOR when an unassigned or uninstalled order code is decoded.

Receiver Check: This condition exists when the addressed CPU detects malfunctioning of equipment during the communications associated with the execution of SIGNAL PROCESSOR. When this condition is indicated, the order has not been

initiated, and, since the malfunction may have affected the generation of the remaining receiver status bits, these bits are not necessarily valid. A machine-check condition may or may not have been generated at the addressed CPU.

The following chart summarizes which status conditions are presented to the issuing CPU in response to each order code.

	Receiver check≠	Invalid order	Not ready	Check stop	Operator intervening#	Stopped	External call pending
Sense	X	X	X	X	X	0	X
External call	X	0	X	X	X	0	X
Emergency signal	0	0	X	X	X	0	X
Start	0	0	X	X	X	0	X
Stop	0	0	X	X	X	0	X
Restart	0	0	X	X	X	0	X
Initial program reset	0	0	X	0	X	0	X
Program reset	0	0	X	0	X	0	X
Stop and store status	0	0	X	X	X	0	X
IML*	0	0	X	0	0	0	X
Initial CPU reset*	0	0	X	0	X	0	X
CPU reset*	0	0	X	0	X	0	X
Unassigned order	0	0	X	0/X	X	1	X

Explanation:

- 0 — A zero is presented in this bit position regardless of the current state of this condition.
- 1 — A one is presented in this bit position.
- X — A zero or a one is presented in this bit position, reflecting the current state of the corresponding condition.
- 0/X — Either a zero or the current state of the corresponding condition is indicated.
- # — The current state of the operator-intervening condition may depend on the order code that is being interpreted.
- ≠ — If a one is presented in the receiver-check bit position, the values presented in the other bit positions are not necessarily valid.
- * — If the order code is implemented, use the line entry for the order code; if the order code is not implemented, use the line entry labeled "Unassigned Order."

If the presented status bits are all zeros, the order has been accepted, and the issuing CPU sets condition code 0. If one or more ones are presented, the order has been rejected, and the issuing CPU stores the status in the general register specified by the R_1 field of the SIGNAL PROCESSOR instruction and sets condition code 1.

Programming Notes

1. A CPU can obtain the following functions by addressing SIGNAL PROCESSOR to itself:
   a. Sense indicates whether an external-call condition is pending.
   b. External call and emergency signal cause the corresponding interruption conditions to be generated. External call can be rejected because of a previously generated external-call condition.
   c. Start sets condition code 0 and has no other effect.
   d. Stop causes the CPU to set condition code 0, take pending interruptions for which it is enabled, and enter the stopped state.
   e. Restart provides a means to store the current PSW.
   f. Stop and store status causes the machine to stop and store all current status.
2. Two CPUs can simultaneously execute SIGNAL PROCESSOR, with each CPU addressing the other. When this occurs, one CPU, but not both, can find the access path busy because of the transmission of the order code or status bits associated with SIGNAL PROCESSOR that is being executed by the other CPU. Alternatively, both CPUs can find the access path available and transmit the order codes to each other. In particular, two CPUs can simultaneously stop, restart, or reset each other.

CHANNEL-SET SWITCHING

The channel-set-switching facility permits a collection of channels to be switched from one CPU to another. The collection of

channels which are switched as a group is called a channel set. A CPU can be connected to only one channel set at a time, and a channel set can be connected to only one CPU at a time. The switching operation controls only the execution of I/O instructions and I/O interruptions. Other channel activity, such as chaining and data-transfer operations, is not controlled by the switching.

When a channel set is switched to a particular CPU, it is said to be connected to that CPU. Channel-set switching permits any channel set in the configuration to be connected to any CPU in the configuration. However, a channel set can be connected to no more than one CPU at a time, and vice versa. When a channel set is not connected to a CPU, it is said to be disconnected. On a particular CPU, all I/O instructions executed address only the channels within the channel set which is currently connected to that CPU. Initial program reset and program reset issued to a CPU result in the resetting of the CPU and of only those channels which are currently connected to that CPU. Similarly, I/O interruptions caused by a channel which is part of a particular channel set occur on the CPU to which the channel set is currently connected. Chaining and data-transfer operations by the channel continue, independent of whether the channel set is connected to a CPU.

Channel sets can be connected and disconnected by means of two instructions, CONNECT CHANNEL SET and DISCONNECT CHANNEL SET, which are defined in Chapter 10, "Control Instructions." These instructions select a particular channel set by means of a 16-bit channel-set address. When the addressed channel set is not operational, execution of these instructions results in a setting of condition code 3. A channel set is not operational when it is not provided in the installation, has power off, is not in the configuration, or is in certain customer-engineer test modes. Depending on the model, a channel set may be not operational when all of the channels in the channel set are not operational.

When a channel set is connected to a CPU and the CPU becomes not operational, the channel set may also become not operational, or it may become disconnected and remain in the configuration. A CPU can become not operational because of certain customer-engineer test modes being set, because model-dependent reconfiguration controls remove it from the configuration, or because its power is off.

The number of CPUs and channel sets in a particular configuration is not necessarily the same.

When system reset normal, system reset clear, load normal, or load clear is activated on any CPU in the configuration, in the absence of any override by model-dependent reconfiguration controls, then:

- All channels within all channel sets in the configuration perform system reset,
- Each channel set which has a home CPU which is operational and in the configuration is connected to its home CPU, and
- Each channel set which does not have a home CPU which is operational and in the configuration is disconnected.

By definition, the CPU to which a channel set is connected after subsystem reset is called the home CPU for that channel set. The address of the channel set may or may not be the same as the address of its home CPU.

When no channel set is connected to a particular CPU, the execution of any I/O instruction results in a setting of condition code 3. When a channel set is connected to a particular CPU, condition code 3 to an I/O instruction normally indicates that the addressed channel, subchannel, or device is not operational. The I/O instructions are described in Chapter 12, "Input/Output Operations." The connection or disconnection of a channel set is not considered to be a change in the channel state for purposes of setting to one the machine-check external-damage-code bit 3, channel not operational. The setting of this bit, even when a channel set is disconnected, indicates only those changes from the operational state to the not-operational state which would be seen if the channel set were connected to a CPU.

CHAPTER 5. PROGRAM EXECUTION

Normally, operation of the CPU is controlled by instructions in storage that are executed sequentially, one at a time, left to right in an ascending sequence of storage addresses. A change in the sequential operation may be caused by branching, LOAD PSW, interruptions, SIGNAL PROCESSOR orders, or manual intervention.

INSTRUCTIONS

Each instruction consists of two major parts:

- An operation code (op code), which specifies the operation to be performed
- The designation of the operands that participate

OPERANDS

Operands can be grouped in three classes: operands located in registers, immediate operands, and operands in storage. Operands may be either explicitly or implicitly designated.

Register operands can be located in general, floating-point, or control registers, with the type of register identified by the op code. The register containing the operand is specified by identifying the register in a four-bit field, called the R field, in the instruction. For some instructions, an operand is located in an implicitly designated register, the register being implied by the op code.

Immediate operands are contained within the instruction, and the eight-bit field containing the immediate operand is called the I field.

Operands in storage may have an implied length; be specified by a bit mask; be specified by a four-bit or eight-bit length specification, called the L field, in the instruction; or have a length specified by the contents of a general register. The addresses of operands in storage are specified by means of a format that uses the contents of a general register as part of the address. This makes it possible to:

1. Specify a complete address by using an abbreviated notation
2. Perform address manipulation using instructions which employ general registers for operands

3. Modify addresses by program means without alteration of the instruction stream

4. Operate independent of the location of data areas by directly using addresses received from other programs

The address used to refer to storage either is contained in a register designated by the R field in the instruction or is calculated from a base address, index, and displacement, designated by the B, X, and D fields, respectively, in the instruction.

To describe the execution of instructions, operands are designated as first and second operands and, in some cases, third operands.

In general, two operands participate in an instruction execution, and the result replaces the first operand. However, CONVERT TO DECIMAL, TEST BLOCK, and instructions with "store" in the instruction name (other than STORE THEN AND SYSTEM MASK and STORE THEN OR SYSTEM MASK) use the second-operand address to designate a location in which to store. Except when otherwise stated, the contents of all registers and storage locations participating in the addressing or execution part of an operation remain unchanged.

INSTRUCTION FORMAT

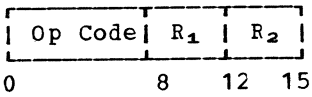
An instruction is one, two, or three halfwords in length and must be located in storage on a halfword boundary. Each instruction is in one of eight basic formats: RR, RRE, RX, RS, SI, S, SSE, and SS, with two variations of SS. (See the figure "Basic Instruction Formats.")

Some instructions contain fields that vary slightly from the basic format, and in some instructions the operation performed does not follow the general rules stated in this section. All of these exceptions are explicitly identified in the individual instruction descriptions.

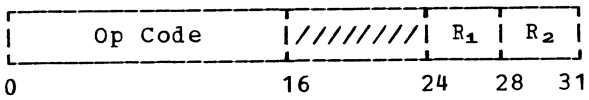
The format names indicate, in general terms, the classes of operands which participate in the operation:

- RR denotes a register-and-register operation.
- RRE denotes a register-and-register operation having an extended op-code field.
- RX denotes a register-and-indexed-storage operation.
- RS denotes a register-and-storage operation.
- SI denotes a storage-and-immediate operation.
- S denotes an operation using an implied operand and storage.
- SS denotes a storage-and-storage operation.
- SSE denotes a storage-and-storage operation having an extended op-code field.

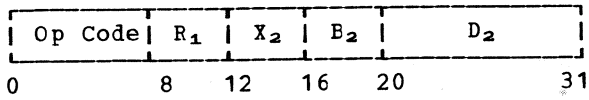
RR Format



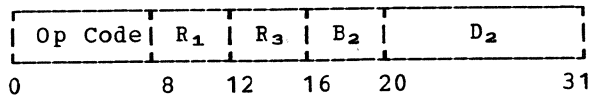
RRE Format



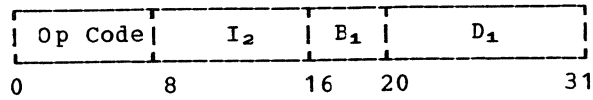
RX Format



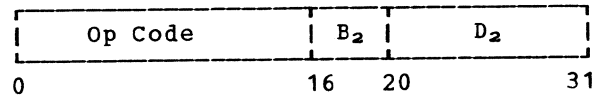
RS Format



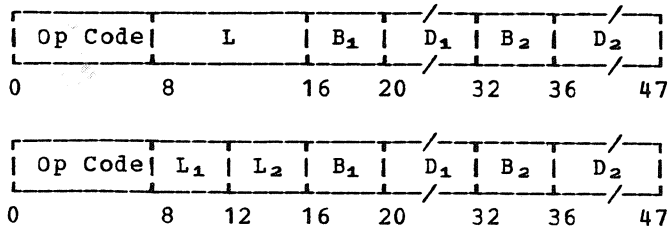
SI Format



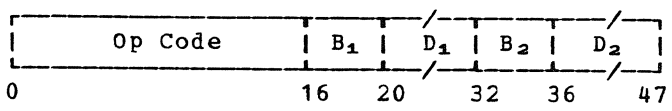
S Format



SS Format



SSE Format



Basic Instruction Formats

The first byte or, in the RRE, S, and SSE formats, the first two bytes of an instruction contain the op code. For some instructions in the S format, all or a portion of the second byte is ignored.

The first two bits of the first or only byte of the op code specify the length and format of the instruction, as follows:

Bit Positions (0-1)	Instruction Length	Instruction Format
00	One halfword	RR
01	Two halfwords	RX
10	Two halfwords	RRE/RS/RX/S/SI
11	Three halfwords	SS/SSE

In the format illustration for each individual instruction description, the op-code field shows the op code as hexadecimal digits within single quotes. The hexadecimal representation uses 0-9 for the binary codes 0000-1001 and A-F for the binary codes 1010-1111.

The remaining fields in the format illustration for each instruction are designated by code names, consisting of a letter and possibly a subscript number. The subscript number denotes the operand to which the field applies.

Register Operands

In the RR, RRE, RX, and RS formats, the contents of the register designated by the R_1 field are called the first operand. The register containing the first operand is sometimes referred to as the "first-operand location." In the RR and RRE formats, the R_2 field designates the register containing the second operand, and the R_2 field may designate the same register as R_1. In the RS format, the use of the R_3 field depends on the instruction.

The R field designates a general register in the general and control instructions and a floating-point register in the floating-point instructions. In the instructions LOAD CONTROL and STORE CONTROL, the R field designates a control register.

Unless otherwise indicated in the individual instruction description, the register operand is one register in length (32 bits for a general register or a control register and 64 bits for a floating-point register), and the second operand is the same length as the first.

Immediate Operands

In the SI format, the contents of the eight-bit immediate-data field, the I_2 field of the instruction, are used directly as the second operand. The B_1 and D_1 fields designate the first operand, which is one byte in length.

Storage Operands

In the SI, SSE, and SS formats, the contents of the general register designated by the B_1 field are added to the contents of the D_1 field to form the first-operand address. In the S, RS, SSE, and SS formats, the contents of the general register designated by the B_2 field are added to the contents of the D_2 field to form the second-operand address. In the RX format, the contents of the general registers designated by the X_2 and B_2 fields are added to the contents of the D_2 field to form the second-operand address.

In the SS format, with two length fields given, L_1 specifies the number of additional operand bytes to the right of the byte designated by the first-operand address. Therefore, the length in bytes of the first operand is 1-16, corresponding to a length code in L_1 of 0-15. Similarly, L_2 specifies the number of additional operand bytes to the right of the location designated by the second-operand address. Results replace the first operand and are never stored outside the field specified by the address and length. If the first operand is longer than the second, the second operand is extended on the left with zeros up to the length of the first

operand. This extension does not modify the second operand in storage.

In the SS format with a single, eight-bit length field, L specifies the number of additional operand bytes to the right of the byte designated by the first-operand address. Therefore, the length in bytes of the first operand is 1-256, corresponding to a length code in L of 0-255. Storage results replace the first operand and are never stored outside the field specified by the address and length. In this format, the second operand has the same length as the first operand, except for the following instructions: EDIT, EDIT AND MARK, TRANSLATE, and TRANSLATE AND TEST.

ADDRESS GENERATION

Execution of instructions by the CPU involves generation of the addresses of instructions and operands. This section describes address generation as it applies to most instructions. In some instructions, the operation performed does not follow the general rules stated in this section. All of these exceptions are explicitly identified in the individual instruction descriptions.

SEQUENTIAL INSTRUCTION-ADDRESS GENERATION

When an instruction is fetched from the location designated by the current PSW, the instruction address is increased by the number of bytes in the instruction, and the instruction is executed. The same steps are then repeated by using the new value of the instruction address to fetch the next instruction in the sequence.

Instruction addresses wrap around, with the halfword at instruction address $2^{24} - 2$ being followed by the halfword at instruction address 0. Thus, any carry out of PSW bit position 40, as a result of updating the instruction address, is lost.

OPERAND-ADDRESS GENERATION

An operand address that refers to storage either is contained in a register designated by an R field in the instruction or is calculated from the sum of three binary numbers: base address, index, and displacement.

The base address (B) is a 24-bit number contained in a general register specified by the program in a four-bit field, called the B field, in the instruction. Base addresses can be used as a means of independently addressing each program and data area. In array-type calculations, it can specify the location of an array, and, in record-type processing, it can identify the record. The base address provides for addressing the entire storage. The base address may also be used for indexing.

The index (X) is a 24-bit number contained in a general register designated by the program in a four-bit field, called the X field, in the instruction. It is included only in the address specified by the RX-format instructions. The RX-format instructions permit double indexing; that is, the index can be used to provide the address of an element within an array.

The displacement (D) is a 12-bit number contained in a field, called the D field, in the instruction. The displacement provides for relative addressing of up to 4,095 bytes beyond the location designated by the base address. In array-type calculations, the displacement can be used to specify one of many items associated with an element. In the processing of records, the displacement can be used to identify items within a record.

In forming the address, the base address and index are treated as 24-bit unsigned binary integers. The displacement is similarly treated as a 12-bit unsigned binary integer, and 12 zeros are appended on the left. The three are added as 24-bit binary numbers, ignoring overflow. The sum is always 24 bits long. The bits of the generated address are numbered 8-31, corresponding to the numbering of the base-address and index bits in the general register.

A zero in any of the B_1, B_2, or X_2 fields indicates the absence of the corresponding address component. For the absent component, a zero is used in forming the address, regardless of the contents of general register 0. A displacement of zero has no special significance.

When an instruction description specifies that the contents of a general register designated by an R field are used to address an operand in storage, bit positions 8-31 of the register provide the operand address. For the instructions INSERT STORAGE KEY EXTENDED, RESET REFERENCE BIT EXTENDED, SET STORAGE KEY EXTENDED, and TEST BLOCK, bits 1-31 of the register provide the address.

An instruction can designate the same general register both for address computation and as the location of an

operand. Address computation is completed before registers, if any, are changed by the operation.

Unless otherwise indicated in an individual instruction definition, the generated operand address designates the leftmost byte of an operand in storage.

Programming Note

Negative values may be used in index and base-address registers. Bits 0-7 of these values are always ignored.

BRANCH-ADDRESS GENERATION

For branch instructions, the address of the next instruction to be executed when the branch is taken is called the branch address. Depending on the branch instruction, the instruction format may be RR, RS, or RX.

In the RS and RX formats, the branch address is designated by a base address, a displacement, and, for RX, an index. In the RS and RX formats, the branch address generation follows the normal rules for operand-address generation.

In the RR format, the contents of bit positions 8-31 of the general register designated by the R_2 field are used as the branch address. General register 0 cannot be designated as containing a branch address. A value of zero in the R_2 field causes the instruction to be executed without branching.

For several branch instructions, branching depends on satisfying a specified condition. When the condition is not satisfied, the branch is not taken, normal sequential instruction execution continues, and the branch address is not used. When a branch is taken, bits 8-31 of the branch address replace bits 40-63 of the current PSW. The branch address is not used to reference storage as part of the branch operation.

A specification exception due to an odd branch address and access exceptions due to fetching of the instruction at the branch location are not recognized as part of the branch operation but instead are recognized as exceptions associated with the execution of the instruction at the branch location.

A branch instruction, such as BRANCH AND LINK, can designate the same general register for branch-address computation and as the location of an operand. Branch-address computation is completed before the remainder of the operation is executed.

INSTRUCTION EXECUTION AND SEQUENCING

The program-status word (PSW), described in Chapter 4, "Control," contains information required for proper program execution. The PSW is used to control instruction sequencing and to hold and indicate the status of the CPU in relation to the program currently being executed. The active or controlling PSW is called the current PSW.

Branch instructions perform the functions of decision-making, loop control, and subroutine linkage. A branch instruction affects instruction sequencing by introducing a new instruction address into the current PSW.

DECISION-MAKING

Facilities for decision-making are provided by BRANCH ON CONDITION. This instruction inspects a condition code that reflects the result of a majority of the arithmetic, logical, and I/O operations. The condition code, which consists of two bits, provides for four possible condition-code settings: 0, 1, 2, and 3.

The specific meaning of any setting depends on the operation that sets the condition code. For example, the condition code reflects such conditions as zero, nonzero, first operand high, equal, overflow, and channel busy. Once set, the condition code remains unchanged until modified by an instruction that causes a different condition code to be set. See Appendix C, "Condition-Code Settings," for a summary of the instructions which set the condition code.

LOOP CONTROL

Loop control can be performed by the use of BRANCH ON CONDITION to test the outcome of address arithmetic and counting operations. For some particularly frequent combinations of arithmetic and tests, BRANCH ON COUNT, BRANCH ON INDEX HIGH, and BRANCH ON INDEX LOW OR EQUAL are provided. These branches, being specialized, provide increased performance for these tasks.

SUBROUTINE LINKAGE

Subroutine linkage is provided by the BRANCH AND LINK and BRANCH AND SAVE instructions, which permit not only the introduction of a new instruction address but also the preservation of the return address and associated information. Linkage between a program and the supervisor or monitoring program is provided by means of the SUPERVISOR CALL and MONITOR CALL instructions.

The instructions PROGRAM CALL and PROGRAM TRANSFER provide the facility for linkage between programs of different authority and in different address spaces. PROGRAM CALL permits linkage to a number of preassigned programs that may be in either problem or supervisor state and may be in either the same address space or an address space different from that of the caller. In general, it is used to transfer control to a program of higher authority. PROGRAM TRANSFER permits a change of instruction address and address space. PROGRAM TRANSFER also permits a reduction in PSW-key-mask authority and a change from supervisor to problem state. In general, it is used to transfer control from one program to another of equal or lower authority. PROGRAM TRANSFER can be used to return from a program called by PROGRAM CALL.

The operation of PROGRAM CALL is controlled by means of an entry-table entry, which is located as part of a table-lookup process during the execution of the instruction. The instruction causes the primary address space to be changed only when the ASN in the entry-table entry is nonzero. When the primary address space is changed, the operation is called PROGRAM CALL with space switching (PC-ss). When the primary address space is not changed, the operation is called PROGRAM CALL to current primary (PC-cp).

PROGRAM TRANSFER specifies an address space which is to become the new primary address space. When the primary address space is changed, the operation is called PROGRAM TRANSFER with space switching (PT-ss). When the primary address space is not changed, the operation is called PROGRAM TRANSFER to current primary (PT-cp).

The linkage instructions provided and the functions performed by each are summarized in the figure "Linkage-Instruction Summary."

Instruction	Format	Instruction Address PSW Bits 40–63		Problem State PSW Bit 15		PASN CR4 Bits 16–31		PSW-Key Mask Changed in CR3
		Save	Set	Save	Set	Save	Set	
BALR	RR	Yes*	R_2[1]	–	–	–	–	–
BAL	RX	Yes*	Yes	–	–	–	–	–
BASR	RR	Yes	R_2[1]	–	–	–	–	–
BAS	RX	Yes	Yes	–	–	–	–	–
MC#	SI	Yes[2]	Yes[2]	Yes	Yes	–	–	–
PC–cp	S	Yes	Yes	Yes	Yes	–	–	"OR" EKM
PC–ss	S	Yes	Yes	Yes	Yes	Yes	Yes	"OR" EKM
PT–cp	RRE	–	R_2	–	R_2**	–	–	"AND" R_1
PT–ss	RRE	–	R_2	–	R_2**	–	Yes	"AND" R_1
SVC	RR	Yes[2]	Yes[2]	Yes	Yes	–	–	–

Explanation:

– No

[1] The action takes place only if the R_2 field in the instruction is not zero.

[2] MC and SVC, as part of the interruption, save the entire PSW and load a new one.

* BAL and BALR also save the instruction-length code, condition code, and program mask.

** A change from supervisor to problem state is permissible; a privileged-operation exception is recognized when a change from problem to supervisor state is specified.

\# Monitor mask bits provide a means of disallowing linkage or enabling linkage for selected classes of events.

Linkage-Instruction Summary

INTERRUPTIONS

Interruptions permit the CPU to change state as a result of conditions external to the system, in channels or input/output (I/O) devices, in other CPUs, or in the CPU itself. Details are to be found in Chapter 6, "Interruptions."

Six classes of interruption conditions are provided: external, I/O, machine check, program, restart, and supervisor call. Each class has two related PSWs, called old and new, in permanently assigned real locations. In all classes, an interruption involves storing information identifying the cause of the interruption, storing the current PSW at the old-PSW location, and fetching the PSW at the new-PSW location, which becomes the current PSW.

The old PSW contains CPU-status information necessary for resumption of the interrupted program. At the conclusion of the program invoked by the interruption, the instruction LOAD PSW may be used to restore the current PSW to the value of the old PSW.

TYPES OF INSTRUCTION ENDING

Instruction execution ends in one of five ways: completion, nullification, suppression, termination, and partial completion.

Partial completion of instruction execution occurs only for interruptible instructions; it is described in the section "Interruptible Instructions" later in this chapter.

Completion

Completion of instruction execution provides results as called for in the definition of the instruction. When an interruption occurs after the completion of the execution of an instruction, the instruction address in the old PSW designates the next instruction to be executed.

Suppression

Suppression of instruction execution causes the instruction to be executed as if it specified "no operation." The contents of any result fields, including the condition code, are not changed. The instruction address in the old PSW on an interruption after suppression designates the next sequential instruction.

Nullification

Nullification of instruction execution has the same effect as suppression, except that when an interruption occurs after the execution of an instruction has been nullified, the instruction address in the old PSW designates the instruction whose execution was nullified instead of the next sequential instruction.

Termination

Termination of instruction execution causes the contents of any fields due to be changed by the instruction to be unpredictable. The operation may replace all, part, or none of the contents of the designated result fields and may change the condition code if such change is called for by the instruction. Unless the interruption is caused by a machine-check condition, the validity of the instruction address in the PSW, the interruption code, and the ILC are not affected, and the state or the operation of the machine is not affected in any other way. The instruction address in the old PSW on an interruption after termination designates the next sequential instruction.

INTERRUPTIBLE INSTRUCTIONS

Point of Interruption

For most instructions, the entire execution of an instruction is one operation. An interruption is permitted between operations; that is, an interruption can occur after the performance of one operation and before the start of a subsequent operation.

For the following instructions, referred to as interruptible instructions, an interruption is permitted after partial completion of the instruction:

COMPARE LOGICAL LONG
MOVE LONG
TEST BLOCK

The execution of an interruptible instruction is considered to consist in the execution of a number of units of operation, and an interruption is permitted between units of operation. The amount of data processed in a unit of operation depends on the particular instruction and may depend on the model and on the particular condition that causes the execution of the instruction to be interrupted.

Whenever points of interruption that include those occurring within the execution of an interruptible instruction are discussed, the term "unit of operation" is used. For a noninterruptible instruction, the entire execution consists, in effect, in the execution of one unit of operation.

When an instruction consists of a number of units of operation and an interruption occurs after some, but not all, units of operation have been completed, the instruction is said to be partially completed. In this case, the type of ending (completion, suppression, nullification) is associated with the unit of operation. In the case of termination, the entire instruction is terminated, not just the unit of operation.

Execution of Interruptible Instructions

The execution of an interruptible instruction is completed when all units of operation associated with that instruction are completed. When an interruption occurs after completion, nullification, or suppression of a unit of operation, all preceding units of operation have been completed.

On completion of a unit of operation other than the last one (and on nullification of any unit of operation), the instruction address in the old PSW designates the interrupted instruction, and the operand parameters are adjusted such that the execution of the interrupted instruction is resumed from the point of interruption when the old PSW stored on the interruption is made the current PSW. It depends on the instruction how the operand parameters are adjusted.

When a unit of operation is suppressed, the instruction address in the old PSW designates the next sequential instruction. The operand parameters, however, are adjusted so as to indicate the extent to which instruction execution has been completed. If the instruction is reexecuted after the conditions causing the suppression have been removed, the execution is resumed from the point of interruption. As in the case of completion and nullification, it depends on the instruction how the operand parameters are adjusted.

When an exception which causes termination occurs as part of a unit of operation of an interruptible instruction, the entire operation is terminated, and the contents, in general, of any fields due to be changed by the instruction are unpredictable. On such an interruption, the instruction address in the old PSW designates the next sequential instruction.

Programming Notes

1. Any interruption, other than supervisor call and some program interruptions, can occur after a partial execution of an interruptible instruction. In particular, interruptions for external, I/O, machine-check, restart, and program interruptions for access exceptions and PER events can occur between units of operation.

2. The amount of data processed in a unit of operation of an interruptible instruction depends on the model and may depend on the type of condition which causes the execution of the instruction to be interrupted or stopped. Thus, when an interruption occurs at the end of the current unit of operation, the length of the unit of operation may be different for different types of interruptions. Also, when the stop function is requested during the execution of an interruptible instruction, the CPU enters the stopped state at the completion of the execution of the current unit of operation. Similarly, in the instruction-step mode, only a single unit of operation is performed, but the unit of operation for the various cases of stopping may be different.

EXCEPTIONS TO NULLIFICATION AND SUPPRESSION

In certain unusual situations, the result fields of an instruction having a store-type operand are changed in spite of the occurrence of an exception which would normally result in nullification or suppression. These situations are exceptions to the general rule that the operation is treated as a no-operation when an exception requiring nullification or suppression is recognized. Each of these situations may result in the turning on of the change bit associated with the store-type operand, even though the final result in storage may appear unchanged. Depending on the particular situation, additional effects may be observable the extent of which is described for each of the situations.

All of these situations are limited to the extent that a store access does not occur and the change bit is not set when the store access is prohibited. For the CPU, a store access is prohibited whenever an access exception exists for that access, or whenever an exception exists which is of higher priority than the priority of an access exception for that access.

When, in these situations, an interruption for an exception requiring suppression occurs, the instruction address in the old PSW designates the next sequential instruction. When an interruption for an exception requiring nullification occurs, the instruction address in the old PSW designates the instruction causing the exception even though partial results may have been stored.

Storage Change and Restoration for DAT-Associated Access Exceptions

In this section, the term "DAT-associated access exceptions" is used to refer to those exceptions which may occur as part of the dynamic-address-translation process. These exceptions are page translation, segment translation, translation specification, and addressing due to a DAT-table entry being specified at a location that is not available in the configuration. The first two of these exceptions normally cause nullification, and the last two normally cause suppression. Protection exceptions, including those due to segment protection, are not considered to be DAT-associated access exceptions.

For DAT-associated access exceptions, on some models, a channel may observe the effects on storage described in the following case.

When, for an instruction having a store-type operand, a DAT-associated access exception is recognized for any operand of the instruction, that portion, if any, of the store-type operand which would not cause an exception may change to an intermediate value but is then restored to the original value.

The accesses associated with storage change and restoration for DAT-associated access exceptions are only observable by a channel and are not observable by another CPU in a multiprocessing configuration. Except for multiple-access operands, the intermediate value, if any, is always equal to what would have been the final value if the DAT-associated access exception had not occurred.

Programming Notes

1. Storage change and restoration for DAT-associated access exceptions occur in two main situations:

   a. The exception is recognized for a portion of a store-type operand which crosses a page boundary, and the other portion has no access exception.

   b. The exception is recognized for one operand of an instruction having two storage operands (for example, an SS-format instruction or MOVE LONG), and the other operand, which is a store-type operand, has no access exception.

2. To avoid letting the channel observe intermediate operand values due to storage change and restoration for DAT-associated access exceptions (especially when a CCW chain is modified), the program should do one of the following:

   - Operate on one storage page at a time
   - Perform preliminary testing to ensure that no exceptions occur for any of the required pages
   - Operate with DAT off

Modification of DAT-Table Entries

When a valid and attached DAT-table entry is changed to a value which would cause an exception, and when, before the TLB is cleared of copies of that entry, an attempt is made to refer to storage by using a virtual address requiring that entry for translation, the contents of any fields due to be changed by the instruction are unpredictable. Results, if any, associated with the virtual address whose DAT-table entry was changed may be placed in those real locations originally associated with the address. Furthermore, it is unpredictable whether or not an interruption occurs for an access exception that was not initially applicable.

Trial Execution for TRANSLATE and EDIT

For the instructions TRANSLATE, EDIT, and EDIT AND MARK, the portions of the operands that are actually used in the operation may be established in a trial execution for operand accessibility that is performed before the execution of the instruction is started. This trial execution consists in an execution of the instruction in which results are not stored. If the first operand of TRANSLATE or either operand of EDIT or EDIT AND MARK is changed by a channel or by another CPU, after the initial trial execution but before completion of execution, the contents of any fields due to be changed by the instruction are unpredictable. Furthermore, it is unpredictable whether or not an interruption occurs for an access exception that was not initially applicable.

Interlocked Update for Nullification and Suppression

When an exception which is defined to cause suppression or nullification is recognized for an instruction with a store-type operand, an interlocked update which does not change the contents of the location may occur for that portion of the store-type operand, if any, for which no access exception exists. The interlocked update can occur only if the priority of the exception is equal to or lower than the priority of an access exception for the store-type operand.

When the exception is a specification exception for a store-type operand which requires alignment on integral boundaries, the interlocked update which may occur is limited to the single byte at the location specified by the operand address.

Programming Note

The interlocked update is only observable by channels; it is not observable by other CPUs. Examples of when an interlocked update may occur to the destination-operand location in storage are:

- Decimal-divide exception for DIVIDE DECIMAL
- Specification exception for an odd register number for COMPARE DOUBLE AND SWAP
- Data exception for an invalid decimal sign for ADD DECIMAL

DUAL-ADDRESS-SPACE CONTROL

The dual-address-space (DAS) facility consists of a number of interrelated functions. Many of these functions are described in this chapter, specifically in these sections: "DAS-Authorization Mechanisms," "PC-Number Translation," "ASN Translation," and "ASN Authorization." Additionally, address spaces are described in Chapter 3, "Storage"; DAS tracing in Chapter 4, "Control"; interruptions in Chapter 6, "Interruptions"; and the instructions in Chapter 10, "Control Instructions."

A complete list of the functions, control-register fields, and instructions that are part of DAS is included in Appendix D, "Facilities."

SUMMARY

These major functions are provided:

1. Two address spaces for immediate use by the program
2. Means for changing to other spaces
3. Three instructions for moving information
4. A table-based subroutine-linkage mechanism
5. The use of multiple access keys for key-controlled protection by problem programs
6. Aids for program-problem analysis

Additionally, control and authority mechanisms are incorporated to control these functions.

These functions are intended for use by programs considered to be semiprivileged, that is, programs which are executed in the problem state but which may be authorized to use additional capabilities. With these authorization controls, a hierarchy of programs may be established, with programs at a higher level having a greater degree of privilege or authority than programs at a lower level. The range of functions available at each level, and the ability to transfer control from a lower to a higher level, are prescribed in tables which are managed by the control program.

The 11 instructions which are included as part of DAS are described in Chapter 10, "Control Instructions." DAS includes the privileged instruction LOAD ADDRESS SPACE PARAMETERS and the following semiprivileged instructions:

```
EXTRACT PRIMARY ASN
EXTRACT SECONDARY ASN
INSERT ADDRESS SPACE CONTROL
INSERT VIRTUAL STORAGE KEY
MOVE TO PRIMARY
MOVE TO SECONDARY
MOVE WITH KEY
PROGRAM CALL
PROGRAM TRANSFER
SET ADDRESS SPACE CONTROL
SET SECONDARY ASN
```

In addition, when DAS is installed, two instructions which are not part of DAS are changed to be semiprivileged. These instructions are:

```
INSERT PSW KEY
SET PSW KEY FROM ADDRESS
```

The changes to the operation of these

instructions are under the control of mode bits in the PSW or in control registers. Whenever a program in the problem state attempts to execute one of these instructions at a time when the required control registers have not been set up, a program exception is indicated which is also available on machines without DAS.

DAS FUNCTIONS

Using Two Address Spaces

Primary and Secondary Address Spaces: DAS makes two address spaces available for use by a semiprivileged program. The use of control register 1 to contain the designation of a segment table for one address space, called the primary address space, is the same as when DAS is not installed. Control register 1 is used when translating primary virtual addresses. For the other address space, called the secondary address space, a segment-table designation is contained in control register 7. Control register 7 is used when translating secondary virtual addresses. DAT applies in the same way to both address spaces.

Address-Space Control: When the address-space-control bit, bit 16 of the EC-mode PSW, is one and DAT is on, the CPU is said to be in secondary-space mode. In secondary-space mode, those operand addresses defined to be logical refer to the secondary address space. In secondary-space mode, it is unpredictable whether instructions are fetched from the primary address space or from the secondary address space. Programs which are executed in this mode are expected to reside in a portion of an address space which is shared between the primary address space and the secondary address space.

The instruction SET ADDRESS SPACE CONTROL provides the semiprivileged program with the capability of selecting either the primary-space mode or the secondary-space mode when DAT is on. Since logical addresses are translated as primary virtual addresses when in primary-space mode and as secondary virtual addresses when in secondary-space mode, the semiprivileged program can use the entire set of unprivileged and semiprivileged instructions to access information in either of the two address spaces. The instruction INSERT ADDRESS SPACE CONTROL provides the program with the ability to inspect the state of the address-space-control bit.

In addition to the function of accessing operands in one address space or the other, the instructions MOVE TO PRIMARY and MOVE TO SECONDARY provide a means of moving data from either of the two address spaces to the other.

Changing to Other Spaces

Address-Space Numbers: DAS provides for changing both the primary address space and the secondary address space. Each address space is designated by a 16-bit value, called the address-space number, or ASN. The ASN can be used as a primary ASN (PASN) or a secondary ASN (SASN). These two values are not used directly to access an address space but are used as symbolic identifiers of the address space.

Bits 16-31 of control register 4 contain the PASN. The PASN can be loaded by means of a PROGRAM CALL with space switching, a PROGRAM TRANSFER with space switching, or LOAD ADDRESS SPACE PARAMETERS. The PASN can be inspected by EXTRACT PRIMARY ASN. When the PASN is loaded by means of the DAS instructions, the corresponding segment-table-designation (STD) value is placed in the primary segment-table designation (PSTD), bits 0-31 of control register 1. The PASN can also be loaded by means of LOAD CONTROL, in which case no translation occurs to convert the PASN to an STD value.

Bits 16-31 of control register 3 contain the SASN. The SASN can be loaded by means of the SET SECONDARY ASN instruction and LOAD ADDRESS SPACE PARAMETERS. The SASN can be inspected by EXTRACT SECONDARY ASN. When the SASN is loaded by means of the DAS instructions, the corresponding STD value is placed in the secondary segment-table designation (SSTD), bits 0-31 of control register 7. The SASN can also be loaded by means of LOAD CONTROL, in which case no translation occurs to convert the SASN to an STD value.

Address-Space-Number Translation: By using the instructions SET SECONDARY ASN and PROGRAM TRANSFER, the program can specify, by reference to a general register containing an ASN, a particular address space which is to be accessed. The ASN specified by the program is used in a table-lookup process, which locates the address-space-control parameters that in turn are used to permit controlled access to the address space. The table lookup includes an authorization test to ensure that the program is authorized to use the

specified address space. The table lookup, including the authorization test and the conversion to system-usable form, is called ASN translation. The same table lookup, but without the authorization test, is performed by the PROGRAM CALL instruction on the ASN specified in the entry-table entry in order to effect the space-switching operation. The instruction LOAD ADDRESS SPACE PARAMETERS also uses ASN translation.

To obtain the segment-table designation and other information for the new address space, the ASN is translated by using a set of tables whose origin is contained in control register 14. A two-level lookup is used. The ASN value is partitioned into two indexes. The first index selects an entry in the table designated by control register 14, called the ASN first table, or AFT. This entry designates another table, called the ASN second table, or AST, an entry of which is selected by the second index. An entry in the second table contains several parameters about the new address space. The information in a second-table entry includes:

- A validity indicator, generally used to indicate whether the associated address space is immediately accessible. This is useful for managing unassigned numbers and swapped-out spaces.
- The origin and length of a table which provides control over whether three of the DAS instructions are authorized to use the new ASN. This table is called the authority table (AT).
- The authorization index (AX), or level, of the new space.
- The origin and length of the segment table to be used by DAT when the new address space is accessed.
- A control over whether a signal, in the form of a space-switch-event program interruption, is given for two of the DAS instructions after a change to a new primary address space is completed.
- The origin of a set of tables which describe the entry points associated with a new primary space. These tables are used by the linkage mechanism provided with DAS. A two-level table structure is provided. The first level is the linkage table (LT), whose entries provide the origins of entry tables (ET).

Changing the Secondary Address Space: The SET SECONDARY ASN instruction causes the secondary address space to be changed to the address space associated with the ASN specified by the instruction. The ASN itself is placed in control register 3 and is called the secondary ASN, or SASN. The ASN is translated to obtain the segment-table designation for the space. This designation is placed in control register 7 as the secondary segment-table designation (SSTD). Instruction execution is disallowed if the translation is not authorized. The translation is authorized by a bit in the authority table at an offset determined by the authorization index in control register 4. The instruction LOAD ADDRESS SPACE PARAMETERS also can change the secondary address space.

Moving Information

DAS provides three instructions for moving information under the control of two access keys.

The instructions MOVE TO PRIMARY and MOVE TO SECONDARY permit the program to move data from either of the two current address spaces to the other. These instructions are defined such that a second access key can be specified in addition to the PSW key. The PSW key in these two instructions is used as the access key for the storage references to the primary address space. Accesses to the secondary address space are made by using a key specified in a general register designated by the instruction. Thus, the program can use the instruction to move data between a calling program area and the semiprivileged-program area and to specify the appropriate key to be used in each area.

A third move instruction, MOVE WITH KEY (MVCK), is used for moving data between differently protected areas, but with both operands appearing in the same address space. MVCK gives a semiprivileged program the capability of moving information to (or from) a caller-specified area from (or to) a semiprivileged-program area. The instruction uses the PSW key for the store accesses associated with the first operand and uses a program-specified key for the fetch accesses associated with the second operand. Thus, a program may set up the PSW key and specify the source key so as to provide appropriate authority checking on a caller-specified address whether it be a source or a target.

For all three move instructions, the number of bytes to be moved is expressed as a true length. A zero length is allowed, with no movement performed. Up to 256 bytes are moved each time one of these instructions is executed, and a condition code is set to

indicate whether the number of bytes moved did or did not exhaust the true length. These capabilities make the instructions suitable for use in a simple program to move a number of bytes from zero up. This is particularly useful when the number of bytes to be moved must be calculated by the program.

Transferring Program Control

DAS permits programs operating at different levels of authority to be linked directly without the use of the SUPERVISOR CALL or MONITOR CALL instruction. The instructions PROGRAM CALL and PROGRAM TRANSFER provide a protected mechanism for transferring control between programs operating at different levels, or the same level, of control.

The PROGRAM CALL instruction specifies a 20-bit index, the PC number, which is used to locate the information associated with the program to be called. This information, called the entry information, includes an authorization key mask, an entry key mask to be ORed into the PSW-key mask in control register 3, the information to be loaded into the problem-state and instruction-address portions of the current PSW, and a parameter which is made available to the called program in general register 4. The entry information can also cause an optional space-switching operation to occur. The space-switching operation is specified when a nonzero address-space number (ASN) is provided as part of the entry information. When space switching occurs, the operation is called PROGRAM CALL with space switching (PC-ss). When no space switching occurs, the operation is called PROGRAM CALL to current primary (PC-cp).

The information associated with the program to be called is obtained by means of a two-level lookup:

- The first lookup consists in indexing into the linkage table to obtain a linkage-table entry, which contains an entry-table address.
- The second lookup consists in indexing into the entry table to obtain an entry-table entry, which contains the entry information.

Since the information loaded into the PSW and control registers is obtained from tables set up by the control program, system integrity is maintained because the problem program cannot load arbitrary values. The current values of the PSW-key mask and the PASN are saved in general register 3. The problem-state status and instruction address are saved in general register 14.

A program can use PC-ss to call a program in another address space. In addition to isolating programs in address spaces, this operation provides for a change to a higher level of privilege and authority. Thus, the called program is entered with an authorization index that can permit access to address spaces which are not authorized to the caller, and with a different linkage table. The called program can then perform services for the calling program by having easy access to these other address spaces, without the requirement that the calling program also have access to these address spaces, and it may use program services which are not available to the calling program. A hierarchy of control can be established and the integrity of the address spaces maintained.

PROGRAM TRANSFER may be used to restore the information saved by PROGRAM CALL. It ANDs information into the PSW-key mask in control register 3 and loads the problem-state status, and instruction address into the current PSW. However, PROGRAM TRANSFER cannot be used to change from problem to supervisor state. Like PROGRAM CALL, PROGRAM TRANSFER is described in terms of two cases: PROGRAM TRANSFER with space switching (PT-ss) and PROGRAM TRANSFER to current primary (PT-cp). PT-ss occurs when the specified ASN is different from the current PASN.

PT-ss provides the return function to be used by a program which has been called by means of PC-ss. The authorization checking provided on PT-ss permits a table structure to be set up which prohibits a program from increasing its authority. PT-ss can also be used to transfer control from one address space to another of the same authority.

PROGRAM CALL and PROGRAM TRANSFER are valid only when the CPU is in primary-space mode. They cause a special-operation exception to be recognized when the CPU is in secondary-space mode or real mode.

Handling Storage Keys and the PSW Key

The handling of keys is facilitated by instructions for changing and extracting the PSW key in problem state. A semi-privileged instruction is provided for obtaining the storage key associated with a virtual-storage location.

INSERT PSW KEY, which is changed by DAS to be semiprivileged,; permits a

semiprivileged program to save the current PSW key for later restoration.

INSERT VIRTUAL STORAGE KEY instruction permits the semiprivileged program to determine the storage key associated with any particular virtual-storage location. It may be used, for example, when one program, with authority to more than one key, calls another program and passes the address of a location to be used as either an input or output buffer. The called program must determine the key needed to access the buffer.

INSERT VIRTUAL STORAGE KEY is also useful to the control program since the instruction uses a virtual rather than a real address. The sequence LOAD REAL ADDRESS followed by INSERT STORAGE KEY or INSERT STORAGE KEY EXTENDED does not necessarily produce a valid result if the program is enabled for interruptions or running in a multiprocessing configuration. This could be the case, for example, in a multiprocessing configuration if another CPU executed INVALIDATE PAGE TABLE ENTRY, followed by a reassignment of the page.

SET PSW KEY FROM ADDRESS, which is changed by DAS to be semiprivileged, provides the semiprivileged program with the capability of changing the PSW key, under control of the PSW-key mask, and thus permits the program to access different data areas protected by different keys.

Increased flexibility in key handling is controlled by a 16-bit PSW-key mask in control register 3. The PSW-key mask permits the semiprivileged program to operate with more than one key without having authorization to all keys. This mask controls the semiprivileged-program use of keys in MOVE TO PRIMARY, MOVE TO SECONDARY, MOVE WITH KEY, and SET PSW KEY FROM ADDRESS. Each bit position corresponds to a key value. The bit in the mask must be one in order for the corresponding key to be used.

Program-Problem Analysis

To aid program-problem analysis, the option is provided of having a trace entry made implicitly for three DAS instructions. When tracing is activated, a trace entry is made each time PROGRAM CALL, PROGRAM TRANSFER, or SET SECONDARY ASN is executed.

As a further analysis aid, PROGRAM CALL and PROGRAM TRANSFER are also recognized for PER purposes as successful branching events. Additionally, for these two instructions, a bit called the space-switch-event-control bit is provided both in control register 1 and in the second-table entry used during ASN translation. When either bit is one, a program interruption for a space-switch event occurs at the completion of the instruction. The effect is to provide for an interruption when a primary-space switch occurs, allowing recognition that a space has been entered, left, or both.

DAS AUTHORIZATION MECHANISMS

The DAS authorization mechanisms permit the control program to establish the degree of function which is provided to a particular program. The authorization mechanisms which are provided by DAS are as follows. (A summary of the authorization mechanisms is given in the figure "Summary of DAS Authorization Mechanisms.")

Mode Requirements

Most of the DAS instructions can be executed only with DAT on. PROGRAM CALL and PROGRAM TRANSFER are valid only in primary-space mode. When a DAS instruction is executed in an invalid translation mode, a special-operation exception is recognized.

PROGRAM TRANSFER specifies a new value for the problem-state bit in the PSW. If a program in the problem state attempts to execute PROGRAM TRANSFER and set the supervisor state, a privileged-operation exception is recognized.

Extraction-Authority Control

The extraction-authority-control bit is located in bit position 4 of control register 0. In the problem state, bit 4 must be one for these instructions:

EXTRACT PRIMARY ASN
EXTRACT SECONDARY ASN
INSERT ADDRESS SPACE CONTROL
INSERT PSW KEY
INSERT VIRTUAL STORAGE KEY

Otherwise, a privileged-operation exception is recognized, and the operation is suppressed. The extraction-authority-control bit is not examined in the supervisor state.

Instr	Mode Requirement: Priv Op	Mode Requirement: Trans Mode	Authorization Mechanism: Subsystem-Linkage Control (CR5.0)	Authorization Mechanism: Secondary-Space Control (CR0.5)	Authorization Mechanism: ASN-Translation-Control (CR14.12): Uncd	Authorization Mechanism: ASN-Translation-Control (CR14.12): Cond	Authorization Mechanism: Extraction-Authority Control (CR0.4)	Authorization Mechanism: PSW-Key Mask (CR3.0-15): Bit Test	Authorization Mechanism: PSW-Key Mask (CR3.0-15): AND AKM[1]	Authorization Mechanism: Authorization Index (CR4.0-15)	Space-Switch-Event-Control Bit (CR1.31)
EPAR		SO-PS					Q				
ESAR		SO-PS					Q				
IAC		SO-PS					Q				
IPK							Q				
IVSK		SO-PS					Q				
LASP	P				SO					CC	CC
MVCP		SO-PS		SO				Q			
MVCS		SO-PS		SO				Q			
MVCK								Q			
PC-cp		SO-P	SO						Q		
PC-ss		SO-P	SO			SO			Q		X
PT-cp	Q[2]	SO-P	SO								
PT-ss	Q[2]	SO-P	SO			SO				PA	X
SAC		SO-PS		SO							
SPKA								Q			
SSAR-cp		SO-PS			SO						
SSAR-ss		SO-PS			SO					SA	

Explanation:

[1] The PSW-key mask is ANDed with the authorization key mask in the entry-table entry.

[2] The exception is recognized on an attempt to set supervisor state when in problem state.

CC Space-switch-event-control bit and authorization index tests cause a condition code to be set.

CR Control register.

P Privileged-operation exception for privileged instruction.

PA Authority checked in problem and supervisor states; violation causes a primary-authority exception.

Q Privileged-operation exception for semiprivileged instruction. Authority checked only in problem state.

SO Authority checked in problem and supervisor states; violation causes a special-operation exception.

SO-PS CPU must be in primary-space mode or secondary-space mode; if the CPU is in real mode, a special-operation exception is recognized in both problem and supervisor states.

SO-P CPU must be in primary-space mode; if the CPU is in secondary-space mode or in real mode, a special-operation exception is recognized in both problem and supervisor states.

SA Authority checked in problem and supervisor states; violation causes a secondary-authority exception.

X When bit 31 of control register 1 is one, a space-switch event is recognized. The operation is completed. The event is recognized in both the problem and supervisor states.

Summary of DAS Authorization Mechanisms

PC-NUMBER TRANSLATION

PC-number translation is the process of translating the 20-bit PC number to locate an entry-table entry as part of the execution of the PROGRAM CALL instruction. To perform this translation, the 20-bit PC number is divided into two fields. Bits 12-23 are the linkage index (LX), and bits 24-31 are the entry index (EX). The format of the effective address, from which the

address space. A program may be authorized to establish the address space as a secondary-address space, as a primary-address space, or both.

Associated with each address space is an authority table. The authorization index is used to select an entry in the authority table. Each entry contains two bits, which indicate whether the program with that authorization index is permitted to establish the address space as a primary address space, as a secondary address space, or both.

The instruction SET SECONDARY ASN with space switching uses the authorization index to test the secondary-authority bit in the authority-table entry to determine if the address space can be established as a secondary address space. The tested bit must be one; otherwise, a secondary-authority exception is recognized, and the operation is nullified.

The instruction PROGRAM TRANSFER with space switching uses the authorization index to test the primary authority bit in the authority-table entry to determine if the address space can be established as a primary address space. If the bit is zero, a primary-authority exception is recognized, and the operation is nullified.

The instruction PROGRAM CALL with space switching causes a new authorization index to be loaded from the ASN-second-table entry. This permits the program which is called to be given an authorization index which authorizes it to access more address spaces than those authorized for the calling program.

Space-Switch-Event-Control Bit

The space-switch event permits the control program to gain control whenever a program enters or leaves a particular address space. Bit 31 of control register 1 is the space-switch-event-control bit. This bit is loaded into control register 1, along with the remaining bits of the primary segment-table designation, whenever control register 1 is loaded. If control register 1 is loaded as a result of a PC-ss or PT-ss operation and either the old or new value of the space-switch-event-control bit is one or both values are ones, then a space-switch-event program interruption occurs after the operation has been completed. A space-switch-event program interruption also occurs at the completion of a PC-ss or PT-ss if any PER event is to be indicated.

Programming Notes

1. The space-switch event may be useful in obtaining programmed authorization checking, in causing additional trace information to be recorded, or in enabling or disabling the CPU for PER or tracing.

2. Bit 95 of the ASN-second-table entry (ASTE) is loaded into bit position 31 of control register 1 as part of the PC-ss and PT-ss operations. If bit 95 of the ASTE for a particular address space is set to one, then a space-switch event is recognized when a program enters or leaves the address space by means of either a PC-ss or a PT-ss.

3. The space-switch event causes the old PASN to be stored at real locations 146-147 and the old space-switch-event-control bit to be stored in bit position 0 at real location 144. This information permits the control program to determine the address space from which the instruction was fetched.

4. The occurrence of a space-switch event at the completion of a PC-ss or PT-ss when any PER event is indicated permits the control program to identify the space from which the instruction causing the PER event was fetched.

PSW-Key Mask

The PSW-key mask consists of bits 0-15 in control register 3. These bits are used in the problem state to control which keys and entry points are authorized for the program. The PSW-key mask is modified by PROGRAM CALL and PROGRAM TRANSFER and is loaded by LOAD ADDRESS SPACE PARAMETERS. The PSW-key mask is used in the problem state to control the following.

- The PSW-key values that can be set by means of the instruction SET PSW KEY FROM ADDRESS.
- The PSW-key values that are valid for the three move instructions that specify a second access key: MOVE WITH KEY, MOVE TO PRIMARY, and MOVE TO SECONDARY.
- The entry points which can be called by means of PROGRAM CALL. In this case, the PSW-key mask is ANDed with the authorization key mask in the entry-table entry, and, if the result is zero, the program is not authorized.

When an instruction in the problem state attempts to use a key not authorized by the PSW-key mask, a privileged-operation exception is recognized, and the operation is suppressed. The same action is taken when an instruction in the problem state attempts to call an entry not authorized by the PSW-key mask. The PSW-key mask is not examined in the supervisor state, all keys and entry points being valid.

Secondary-Space Control

Bit 5 of control register 0 is the secondary-space-control bit. This bit provides a mechanism whereby the control program can indicate whether or not the secondary segment table has been established. Bit 5 must be one for these instructions:

MOVE to PRIMARY
MOVE TO SECONDARY
SET ADDRESS SPACE CONTROL

Otherwise, a special-operation exception is recognized, and the operation is suppressed. The bit is examined in both the problem and supervisor states.

Subsystem-Linkage Control

Bit 0 of control register 5 is the subsystem-linkage-control bit. Bit 0 must be one for these instructions:

PROGRAM CALL
PROGRAM TRANSFER

Otherwise, a special-operation exception is recognized, and the operation is suppressed. The bit is active in both the problem and supervisor states and controls both the space switching and current primary versions of the instructions.

ASN-Translation Control

Bit 12 of control register 14 is the ASN-translation-control bit. This bit provides a mechanism whereby the control program can indicate whether the ASNs are permitted to be translated while a particular program is being executed. Bit 12 must be one for these instructions:

LOAD ADDRESS SPACE PARAMETERS
SET SECONDARY ASN
PROGRAM CALL with space switching
PROGRAM TRANSFER with space switching

Otherwise, a special-operation exception is recognized, and the operation is suppressed. The bit is active in both the problem and supervisor states.

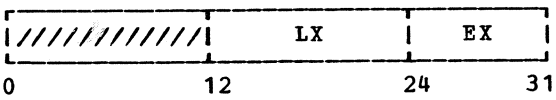
Programming Note

The ASN-translation-control bit is defined such that if zeros are placed in the previously unassigned register positions of control register 14, a problem program attempting to use the ASN-control instructions causes a special-operation exception to be recognized, and the operation is suppressed.

Authorization Index

The authorization index is contained in bits 0-15 of control register 4. The authorization index is associated with the primary address space and is loaded along with the PASN when PROGRAM CALL with space switching, PROGRAM TRANSFER with space switching, or LOAD ADDRESS SPACE PARAMETERS is executed. The authorization index is used to determine whether a program is authorized to establish a particular

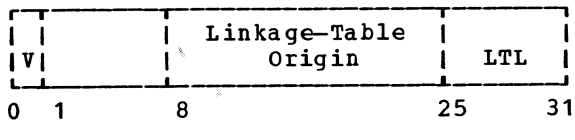
PC-number is taken, is shown in the following diagram:



The translation is performed by means of two tables: a linkage table and an entry table. Both of these tables reside in real storage. The linkage-table designation resides in control register 5. The entry table is designated by means of a linkage-table entry.

PC-NUMBER TRANSLATION CONTROL

PC-number translation is controlled by means of the linkage-table designation in control register 5. The register has the following format:



Subsystem-Linkage Control (V): Bit 0 of control register 5 is the subsystem-linkage-control bit. Bit 0 must be one for these instructions:

PROGRAM CALL
PROGRAM TRANSFER

Otherwise, a special-operation exception is recognized, and the operation is suppressed. The bit is active in both the problem and supervisor states and controls both the space switching and current primary versions of the instructions.

Linkage-Table Origin: Bits 8-24 of control register 5, with seven zeros appended on the right, form a 24-bit real address that designates the beginning of the linkage table. With extended-real addressing, the linkage-table origin is still a 24-bit real address and is extended on the left with zeros.

Linkage-Table Length (LTL): Bits 25-31 of control register 5 designate the length of the linkage table in units of 128 bytes, thus making the length of the linkage table variable in multiples of 32 four-byte entries. The length of the linkage table, in units of 128 bytes, is one more than the value in bit positions 25-31. The linkage-table length is compared against the leftmost seven bits of the linkage-index portion of the PC number to determine whether the linkage index designates an entry within the linkage table.

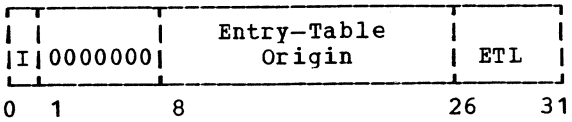
PC-NUMBER TRANSLATION TABLES

The PC-number translation process consists in a two-level lookup using two tables: a linkage table and an entry table. These tables reside in real storage.

Linkage-Table Entries

The linkage-index portion of the PC number is used to select a linkage-table entry. The entry fetched from the linkage table designates the availability, origin, and length of the corresponding entry table.

An entry in the linkage table has the following format:



The fields in the linkage-table entry are allocated as follows:

LX Invalid Bit (I): Bit 0 controls whether the entry table associated with the linkage-table entry is available.

When the bit is zero, PC-number translation proceeds by using the linkage-table entry. When the bit is one, an LX-translation exception is recognized, and the operation is nullified.

Entry-Table Origin: Bits 8-25, with six zeros appended on the right, form a 24-bit real address that designates the beginning of the entry table. With extended real addressing, the entry-table origin is still a 24-bit real address and is extended on the left with zeros.

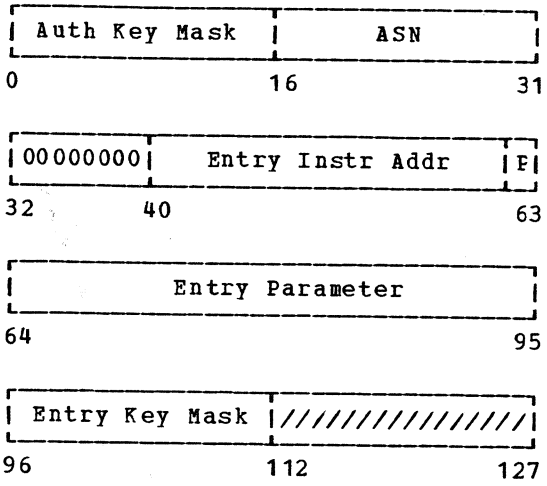
Entry-Table Length (ETL): Bits 26-31 designate the length of the entry table in units of 64 bytes, thus making the entry table variable in multiples of four 16-byte entries. The length of the entry table, in units of 64 bytes, is one more than the value in bit positions 26-31. The entry-table length is compared against the leftmost six bits of the entry index to determine whether the entry index designates an entry within the entry table.

Bits 1-7 of the linkage-table entry must be zeros; otherwise, a PC-translation-specification exception is recognized, and

the operation is suppressed.

Entry-Table Entries

The entry fetched from the entry table is 16 bytes in length and has the following format:



The fields in the entry-table entry are allocated as follows:

Authorization Key Mask: Bits 0-15 are used to verify whether the program issuing the PROGRAM CALL instruction, when in the problem state, is authorized to call this entry point. The authorization key mask and the current PSW-key mask in control register 3 are ANDed, and the result is checked for all zeros. If the result is all zeros, a privileged-operation exception is recognized, and the operation is suppressed. The test is not performed in the supervisor state.

ASN: Bits 16-31 specify whether a PC-ss or PC-cp is to occur. When bits 16-31 are zeros, a PC-cp is specified. When bits 16-31 are not all zeros, a PC-ss is specified, and the bits contain the ASN that replaces the primary ASN.

Entry Instruction Address: Bits 40-62, with a zero appended on the right, form the instruction address which replaces the instruction address in the PSW as part of the PROGRAM CALL operation.

Entry Problem State (P): Bit 63 replaces the problem-state bit, bit 15 of the current PSW, as part of the PROGRAM CALL operation.

Entry Parameter: Bits 64-95 are placed in general register 4.

Entry Key Mask: Bits 96-111 are ORed into the PSW key mask in control register 3 as part of the PROGRAM CALL operation.

Bits 32-39 of the entry-table entry must be zeros; otherwise, a PC-translation-specification exception is recognized, and the operation is suppressed.

Programming Note

The entry parameter is intended to provide the called program with an address which can be depended upon and used as the basis of addressability in locating necessary information which may be environment-dependent. The parameter may be appropriately changed for each environment by setting up different entry tables. The alternative -- obtaining this information from the calling program -- may require extensive validity checking or may present an integrity exposure.

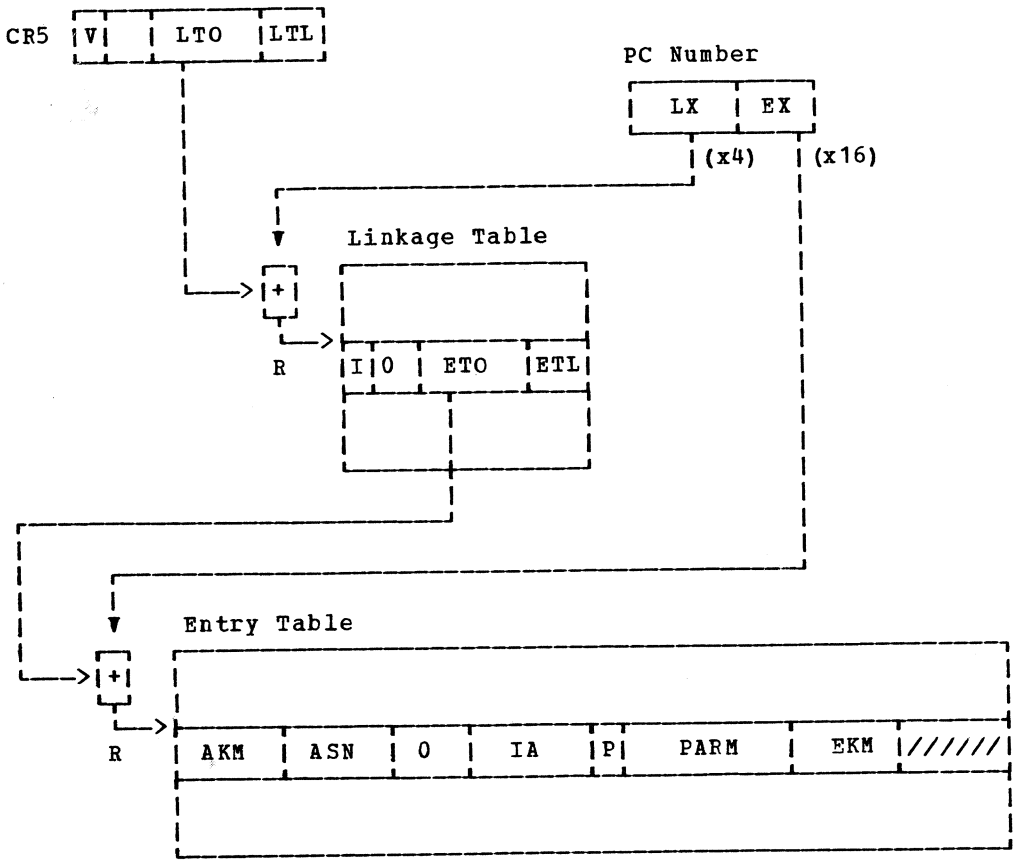
PC-NUMBER-TRANSLATION PROCESS

The translation of the PC number is performed by means of a linkage table and entry table both of which reside in real storage.

For the purposes of PC-number translation, the 20-bit PC number is divided into two parts: the leftmost 12 bits are called the linkage index (LX), and the rightmost eight bits are called the entry index (EX). The LX is used to select an entry from the linkage table, the starting address and length of which are specified by the contents of the linkage-table designation in control register 5. This entry designates the entry table to be used. The EX field of the PC number is then used to select an entry from the entry table.

When, for the purposes of PC-number translation, accesses are made to main storage to fetch entries from the linkage table and entry table, key-controlled protection does not apply.

The PC-number-translation process is shown in the figure "PC-Number Translation."



R: Address is real

PC-Number Translation

Linkage-Table Lookup

The linkage-index (LX) portion of the PC number is used to select an entry from the linkage table. The real address of the linkage-table entry is obtained by appending seven zeros on the right to the contents of bit positions 8-24 of control register 5 and adding the linkage index to this value. For this addition, the linkage index is extended with two rightmost and 10 leftmost zeros.

A carry, if any, into bit position 7 is ignored, and the result is a 24-bit real address.

As part of the linkage-table-lookup process, the leftmost seven bits of the linkage index are compared against the linkage-table length, bits 25-31 of control register 5, to establish whether the addressed entry is within the linkage table. If the value in the linkage-table-length field is less than the value in the seven leftmost bits of the linkage index, an LX-translation exception is recognized, and the operation is nullified.

All four bytes of the linkage-table entry are fetched concurrently. The fetch access is not subject to protection. When the storage address which is generated for fetching the linkage-table entry refers to a location which is not available in the configuration, an addressing exception is recognized, and the operation is suppressed.

Bit 0 of the linkage-table entry specifies whether the entry table corresponding to the linkage index is available. This bit is inspected, and, if it is one, an LX-translation exception is recognized, and the operation is nullified.

When no exceptions are recognized in the process of linkage-table lookup, the entry fetched from the linkage table designates the origin and length of the corresponding entry table.

Entry-Table Lookup

The entry-index (EX) portion of the PC number, in conjunction with the entry-table origin contained in the linkage-table entry, is used to select an entry from the entry table.

The address of the entry-table entry is obtained by appending six zeros on the right to the entry-table origin and adding the entry index, with four rightmost and 12 leftmost zeros appended. A carry, if any, into bit position 7 is ignored, and the result is a 24-bit real address.

As part of the entry-table-lookup process, the six leftmost bits of the entry index are compared against the entry-table length, bits 26-31 of the linkage-table entry, to establish whether the addressed entry is within the table. If the value in the entry-table length field is less than the value in the six leftmost bits of the entry index, an EX-translation exception is recognized, and the operation is nullified.

The 16-byte entry-table entry is fetched by using the real address. The entry is fetched left to right, a doubleword at a time. The fetch access is not subject to protection. When the storage address which is generated for fetching the entry-table entry refers to a location which is not available in the configuration, an addressing exception is recognized, and the operation is suppressed.

The use that is made of the information fetched from the entry-table entry is described in definition of the PROGRAM CALL instruction.

Recognition of Exceptions During PC-Number Translation

The exceptions which can be encountered during the PC-number-translation process and their priority are described in the definition of the PROGRAM CALL instruction.

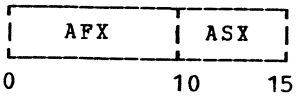
ASN TRANSLATION

ASN translation is the process of translating the 16-bit ASN to locate the address-space-control parameters. ASN translation is performed as part of PROGRAM CALL with space switching (PC-ss), PROGRAM TRANSFER with space switching (PT-ss), and SET SECONDARY ASN with space switching (SSAR-ss). ASN translation is also performed as part of LOAD ADDRESS SPACE PARAMETERS. For PC-ss and PT-ss, the ASN which is translated replaces the primary ASN in control register 4. For SSAR-ss, the ASN which is translated replaces the secondary ASN in control register 3. These two translation processes are called primary ASN translation and secondary ASN translation, respectively, and both can occur for LOAD ADDRESS SPACE PARAMETERS. The ASN-translation process is the same for both primary and secondary ASN translation; only the uses of the results of the process are different.

The ASN-translation process uses two tables, the ASN first table and the ASN second table. They are used to locate the address-space-control parameters and a third table, the authority table, which is used when ASN authorization is performed.

For the purposes of this translation, the 16-bit ASN is considered to consist of two parts: the ASN-first-table index (AFX) is the leftmost 10 bits of the ASN, and the ASN-second-table index (ASX) is the six rightmost bits.

ASN

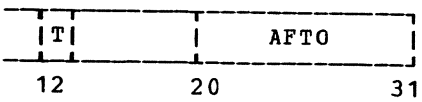


The AFX is used to select an entry from the ASN first table (AFT) the origin of which is designated by the AFTO in control register 14. The AFT entry contains the origin of the ASN second table (AST). The ASX is used to select an entry from the AST. This entry contains the address-space-control parameters.

ASN-TRANSLATION CONTROLS

ASN translation is controlled by the ASN-translation-control bit and the ASN-first-table origin, both of which reside in control register 14.

Control Register 14



ASN-Translation Control (T): Bit 12 of control register 14 is the ASN-translation-control bit. This bit provides a mechanism whereby the control program can indicate whether the ASNs are permitted to be translated while a particular program is being executed. Bit 12 must be one for

these instructions:

LOAD ADDRESS SPACE PARAMETERS
SET SECONDARY ASN
PROGRAM CALL with space switching
PROGRAM TRANSFER with space switching

Otherwise, a special-operation exception is recognized, and the operation is suppressed. The bit is active in both the problem and supervisor states.

ASN-First-Table Origin (AFTO): Bits 20-31 of control register 14, with 12 zeros appended on the right, form a 24-bit real address that designates the beginning of the ASN first table. With extended real addressing, the ASN-first-table origin is still a 24-bit real address and is extended on the left with zeros.

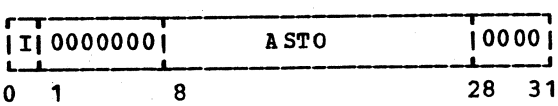
ASN-TRANSLATION TABLES

The ASN-translation process consists in a two-level lookup using two tables: an ASN first table and an ASN second table. These tables reside in real storage.

ASN-First-Table Entries

The entry fetched from the ASN first table (AFT) designates the availability and origin of the corresponding ASN second table.

An entry in the ASN first table has the following format:



The fields in the entry are allocated as follows:

AFX-Invalid Bit (I): Bit 0 controls whether the ASN second table associated with the ASN-first-table entry is available. When bit 0 is zero, ASN translation proceeds by using the designated ASN second table. When the bit is one, the ASN translation cannot continue.

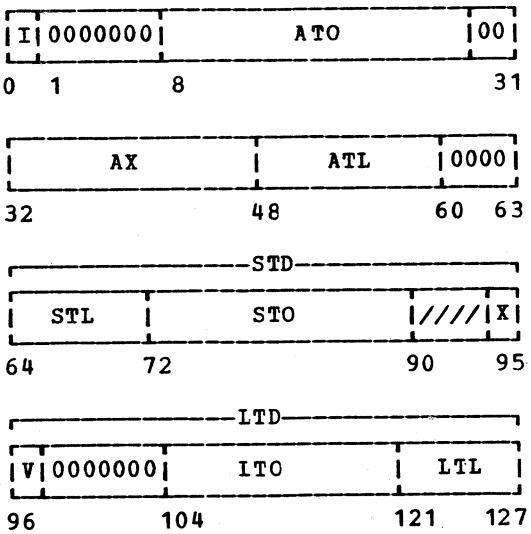
ASN-Second-Table Origin (ASTO): Bits 8-27, with four zeros appended on the right, are used to form a 24-bit real address that designates the beginning of the ASN second table. With extended real addressing, the ASN-second-table origin is still a 24-bit real address and is extended on the left with zeros.

Bits 1-7 and 28-31 of the AFT entry must be zeros; if they are not zeros, an ASN-translation-specification exception is recognized as part of the execution of the instruction using that entry for ASN translation, and the operation is suppressed.

ASN-Second-Table Entries

The entry fetched from the ASN second table indicates the availability of the address space and contains the address-space-control parameters if the address space is available.

The ASN second-table entry has the following format:



The fields in the entry are allocated as follows:

ASX-Invalid Bit (I): Bit 0 controls whether the address space associated with the AST entry is available. When bit 0 is zero, ASN translation proceeds. When the bit is one, the ASN translation cannot continue.

Authority-Table Origin (ATO): Bits 8-29, with two zeros appended on the right, are used to form a 24-bit real address that designates the beginning of the authority table. With extended real addressing, the authority-table origin is still a 24-bit real address and is extended on the left with zeros.

Authorization Index (AX): Bits 32-47 are used as a result of primary ASN translation by PROGRAM CALL and PROGRAM TRANSFER and may be used by LOAD ADDRESS SPACE PARAMETERS. The AX field is ignored for secondary ASN translation.

Authority-Table Length (ATL): Bits 48-59 specify the length of the authority table in units of four bytes, thus making the authority table variable in multiples of 16 entries. The length of the authority table, in units of four bytes, is one more than the ATL value. The contents of the ATL field are used to establish whether the entry designated by a particular AX falls within the authority table.

Segment-Table Designation (STD): Bits 64-95 are used as a result of ASN translation to replace the primary-segment-table designation (PSTD) or the secondary-segment-table designation (SSTD). For SET SECONDARY ASN, the STD field is placed in the SSTD, bits 0-31 of control register 7. For PROGRAM CALL, the STD field is placed in the PSTD, bits 0-31 of control register 1. Each of these actions may occur independently for LOAD ADDRESS SPACE PARAMETERS. For PROGRAM TRANSFER, the STD field is placed in both the PSTD and SSTD, bits 0-31 of control registers 1 and 7, respectively. The contents of the entire STD field are placed in the appropriate control registers without being inspected for validity.

Bit 31 of the segment-table designation is the space-switch-event-control bit. When, in PC-ss or PT-ss, this bit is one in control register 1 either before or after the execution of the PC-ss or PT-ss, a program interruption for a space-switch event occurs after the execution of the instruction is completed. When, in LOAD ADDRESS SPACE PARAMETERS, this bit is one during primary ASN translation, this fact is indicated by the condition code.

Linkage-Table Designation (LTD): Bits 96 and 104-127 are used as a result of primary ASN translation. The linkage-table-designation field contains the subsystem-linkage-control bit (V) (bit 96), the linkage-table origin (LTO) (bits 104-120), and the linkage-table length (LTL) (bits 121-127). The contents of the LTD field are placed in control register 5 as a result of primary ASN translation.

Bits 1-7, 30, 31, 60-63, and 97-103 of the AST entry must be zeros; if these bits are not zeros, an ASN-translation-specification exception is recognized as part of the execution of the instruction using that entry for ASN translation, and the operation is suppressed.

Programming Note

The unused portion of the STD field, bits 90-94 of the AST entry, which corresponds to bits 26-30 of the PSTD and SSTD, should be set to zeros. These bits are reserved for future expansion, and programs which place nonzero values in these bit positions may not operate compatibly on future machines.

ASN-TRANSLATION PROCESS

This section describes the ASN-translation process as it is performed during the execution of PROGRAM CALL with space switching, PROGRAM TRANSFER with space switching, and SET SECONDARY ASN with space switching. ASN translation for LOAD ADDRESS SPACE PARAMETERS is the same, except that AFX-translation and ASX-translation exceptions do not occur; such situations are instead indicated in the condition code. Translation of an ASN is performed by means of two tables, an ASN first table and an ASN second table, both of which reside in main storage.

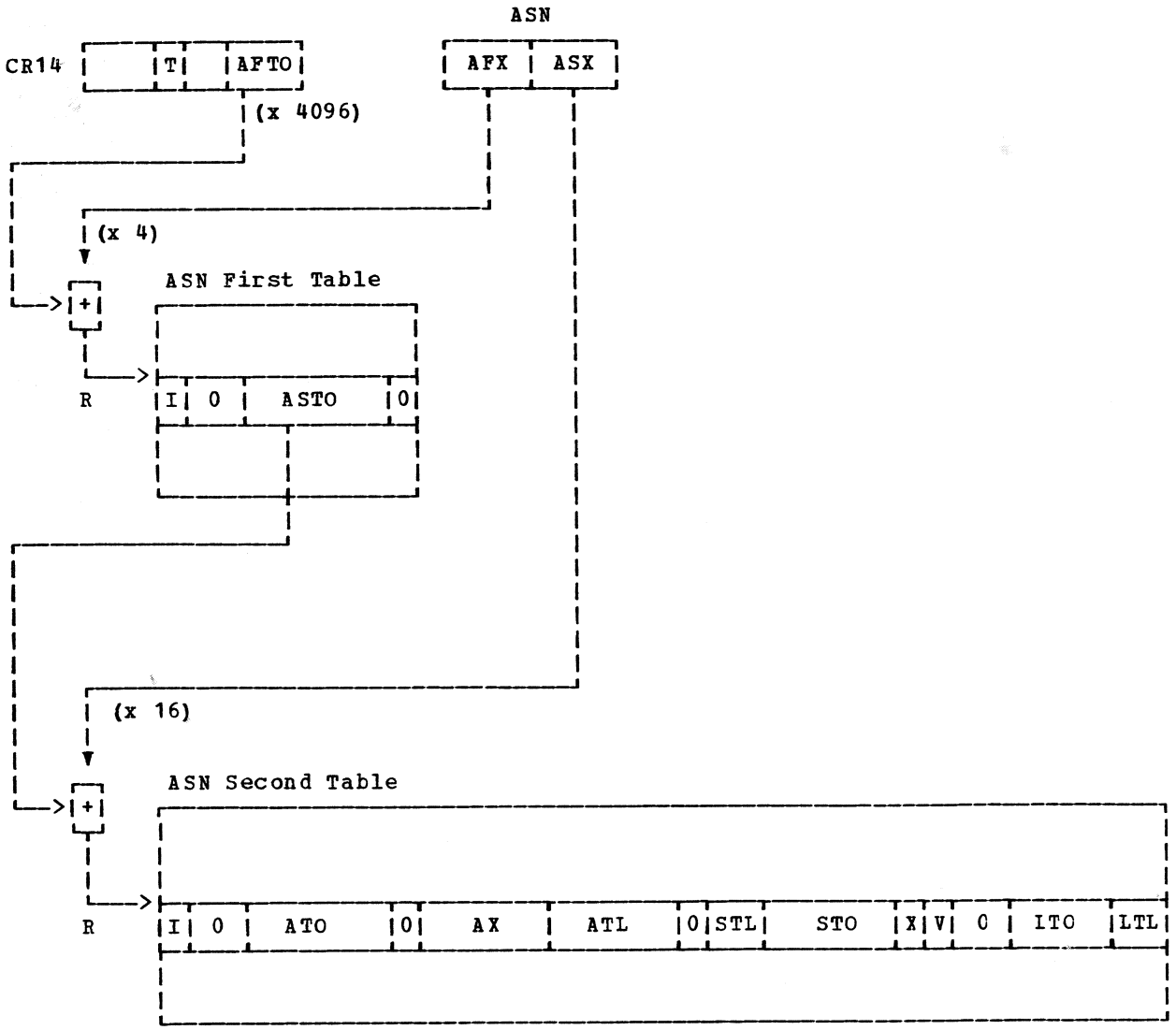
The ASN first index is used to select an entry from the ASN first table. This entry designates the ASN second table to be used.

The ASN second index is used to select an entry from the ASN second table. This entry contains the address-space-control parameters.

If the I bit is one in either the ASN-first-table entry or ASN-second-table entry, the entry is invalid, and the ASN-translation process cannot be completed. An AFX-translation exception or ASX-translation exception is recognized, and the operation is nullified.

Whenever access to main storage is made during the ASN translation process for the purpose of fetching an entry from an ASN first table or ASN second table, key-controlled protection does not apply.

The ASN translation process is shown in the figure "ASN Translation."



R: Address is real

ASN Translation

ASN-First-Table Lookup

The AFX portion of the ASN is used to select an ASN-first-table entry that designates the second table (ASN second table) to be used for the second lookup. The 24-bit real address of the ASN first table is obtained by appending 12 zeros on the right to the AFT origin contained in bit positions 20-31 of control register 14. The real address of the AFT entry is obtained by appending two rightmost zeros and 12 leftmost zeros to the AFX and adding this 24-bit value to the real address of the AFT. This addition cannot cause a carry into bit position 7. With extended real addressing, this 24-bit real address is extended on the left with zeros. All four bytes of the ASN-first-table entry are fetched concurrently. The fetch access is not subject to protection. When the storage address which is generated for fetching the ASN-first-table entry refers to a location which is not available in the configuration, an addressing exception is recognized, and the operation is suppressed.

Bit 0 of the four-byte AFT entry specifies whether the corresponding AST is available. If this bit is one, an AFX-translation exception is recognized, and the operation is nullified. If bit positions 1-7 and 28-31 of the AFT entry dc not contain zeros, an ASN-translation-specification exception is recognized, and the operation is suppressed. When no exceptions are

recognized, the entry fetched from the AFT is used to access the AST.

ASN-Second-Table Lookup

The ASX portion of the ASN, in conjunction with the ASN-second-table origin derived from the ASN-first-table entry, is used to select an entry from the ASN second table (AST). Bits 8-27 of the ASN-first-table entry, with four zeros appended on the right, form the 24-bit real address of the ASN second table. The address of the ASN-second-table entry is obtained by appending four rightmost zeros and 14 leftmost zeros to the ASX and adding this 24-bit value to the real address of the AST, ignoring any carry into bit position 7. With extended real addressing, this 24-bit real address is extended on the left with zeros. Thus, the ASN-second table can wrap from $2^{24} - 1$ to zero.

The 16 bytes of the ASN-second-table entry are fetched left to right, a doubleword at a time. The fetch access is not subject to protection. When the storage address which is generated for fetching the ASN-second-table entry refers to a location which is not available in the configuration, an addressing exception is recognized, and the operation is suppressed.

Bit 0 of the 16-byte ASN-second-table entry specifies whether the address space is accessible. If this bit is one, an ASX-translation exception is recognized, and the operation is nullified. If bit positions 1-7, 30, 31, 60-63, and 97-103 of the ASN-second-table entry do nct contain zeros, an ASN-translation-specification exception is recognized, and the operation is suppressed.

Recognition of Exceptions During ASN Translation

The exceptions which can be encountered during the ASN-translation prccess are collectively referred to as ASN-translation exceptions. A list of these exceptions and their priorities is given in Chapter 6, "Interruptions."

ASN AUTHORIZATION

ASN authorization is the process of testing whether the program associated with the current authorization index is permitted to establish a particular address space. The ASN authorization is performed as part of PROGRAM TRANSFER with space switching (PT-ss) and SET SECONDARY ASN with space switching (SSAR-ss) and may be performed as part of LOAD ADDRESS SPACE PARAMETERS. ASN authorization is performed after the ASN-translation process for these instructions.

When performed as part of PT-ss, the ASN authorization tests whether the ASN can be established as the primary ASN and is called primary-ASN authorization. When performed as part of LOAD ADDRESS SPACE PARAMETERS or SSAR-ss, the ASN authorization tests whether the ASN can be established as the secondary ASN and is called secondary-ASN authorization.

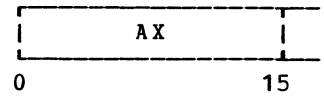
The ASN authorization is performed by means of an authority table in real storage which is designated by the authority-table-origin and authority-table-length fields in the ASN-second-table entry.

ASN-AUTHORIZATICN CONTROLS

ASN authorization uses the authority-table origin and the authority-table length from the ASN-second-table entry, together with an authorization index.

Control Register 4

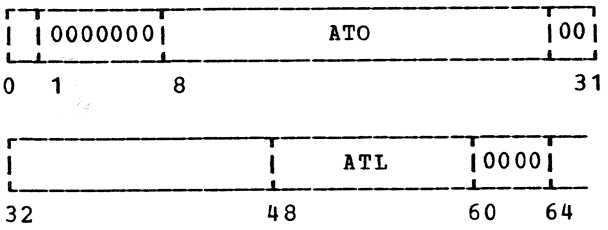
For PT-ss and SSAR-ss, the current contents of control register 4 contain the authorization index. For LOAD ADDRESS SPACE PARAMETERS, the value which will become the new contents of control register 4 are used. The register has the following format:



Authorization Index (AX): Bits 0-15 of control register 4 are used as an index to locate the authority bits in the authority table.

ASN-Second-Table Entry

The ASN-second-table entry which is fetched as part of the ASN translation process contains information which is used to designate the authority table.

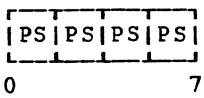


Authority-Table Origin (ATO): Bits 8-29, with two zeros appended on the right, are used to form a 24-bit real address that designates the beginning of the authority table. With extended real addressing, the authority-table origin is still a 24-bit real address and is extended on the left with zeros.

Authority-Table Length (ATL): Bits 48-59 designate the length of the authority table in units of four bytes, thus making the authority table variable in multiples of 16 entries. The length of the authority table, in units of four bytes, is equal to one more than the ATL value. The contents of the length field are used to establish whether the entry designated by the authorization index falls within the authority table.

Authority-Table Entries

The authority table consists of entries of two bits each; accordingly, each byte of the authority table contains four entries:



The fields are allocated as follows:

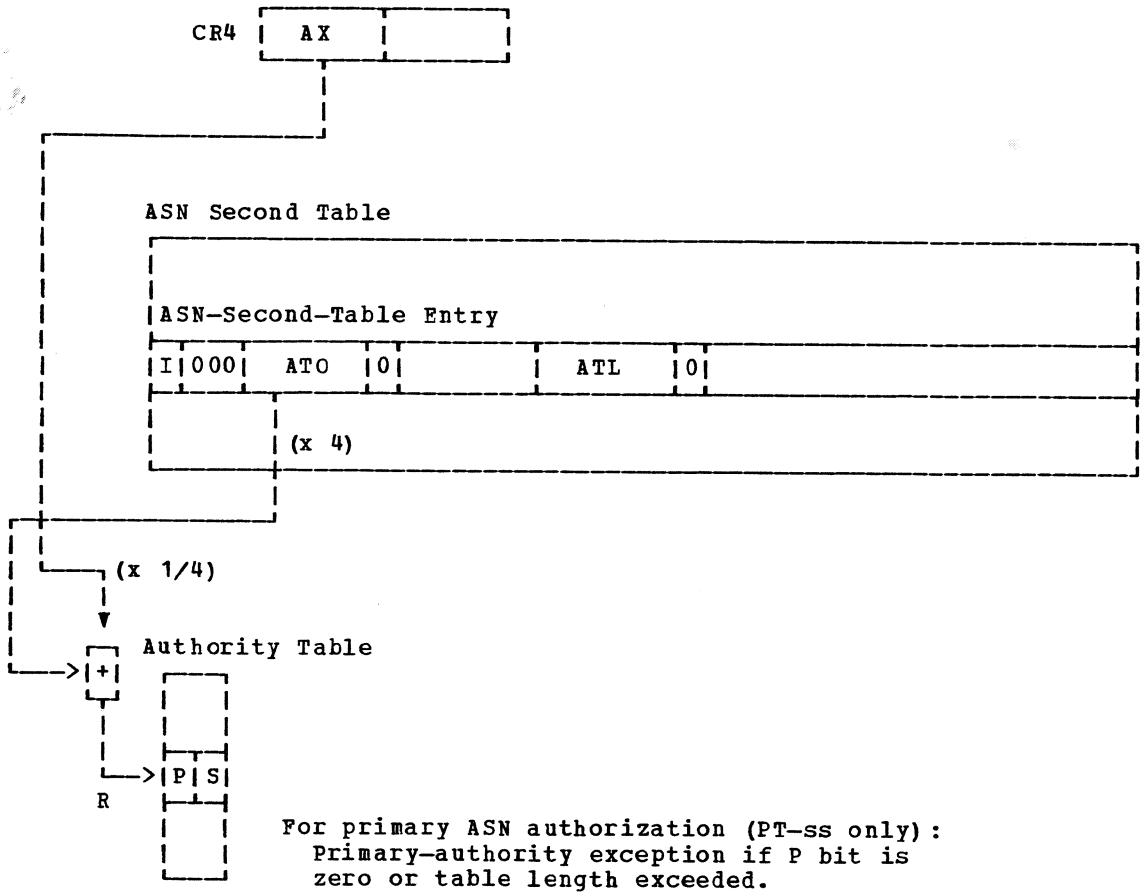
Primary Authority (P): The left bit of an authority-table entry controls whether the program with the authorization index corresponding to the entry is permitted to establish the address space as a primary address space. If the P bit is one, the access is permitted. If the P bit is zero, the access is not permitted.

Secondary Authority (S): The right bit of an authority-table entry controls whether the program with the corresponding authorization index is permitted to establish the address space as a secondary address space. If the S bit is one, the access is permitted. If the S bit is zero, the access is not permitted.

ASN-AUTHORIZATION PRCCESS

This section describes the ASN-authorization process as it is performed during the execution of PROGRAM TRANSFER with space switching and SET SECONDARY ASN with space switching. For these two instructions, the ASN-authorization process is performed by using the authorization index currently in contrcl register 4. Secondary authorization for LOAD ADDRESS SPACE PARAMETERS is the same, except that the value which will become the new contents of control register 4 is used for the authorization index, and a secondary-authority exception does not occur. Instead, such a situation is indicated in the condition code.

The ASN-authorization process is performed by using the authorization index in conjunction with the authority-table origin and length from the AST entry, to select an authority-table entry. The entry is fetched, and either the primary- or secondary-authority bit is examined, depending on whether the primary- or secondary-ASN-authorization process is being performed. The ASN-authorization process is shown in the figure "ASN Authorization."



For primary ASN authorization (PT-ss only):
Primary-authority exception if P bit is zero or table length exceeded.

For secondary ASN authorization (SSAR-ss only):
Secondary-authority exception if S bit is zero or table length exceeded.

For secondary ASN authorization (LASP only):
Set condition code 2 if S bit is zero or table length exceeded.

R: Address is real

ASN Authorization

Authority-Table Lookup

Bit positions 8-31 of the AST entry contain the 24-bit real address of the authority table (ATO), and bit positions 48-59 contain the length of the authority table (ATL).

As part of the authority-table-entry-lookup process, bits 0-11 of the authorization index are compared against the authority-table length (ATL). If the compared portion is greater than ATL, a primary-authority exception or secondary-authority exception is recognized for PT-ss or SSAR-ss, respectively.

The address of a byte in the authority table is obtained by appending 10 zeros on the left to the 14 leftmost bits of the authorization index (AX) obtained from bit positions 0-13 of control register 4 and adding this 24-bit value to the authority-table origin obtained from the AST entry, with two zeros appended on the right. A carry, if any, into bit position 7 is ignored. With extended real addressing, this 24-bit real address is extended on the left with zeros. Thus, the authority table can wrap from $2^{24} - 1$ to zero. If the real address thus generated designates a location which is not available in the configuration, an addressing exception is recognized, and the operation is suppressed. Protection does

not apply to this access.

The byte contains four authority-table entries of two bits each. The rightmost two bits of the authorization index, bits 14 and 15 of control register 4, are used to select one of the four entries. The left or right bit of the entry is then tested, depending on whether the authorization test is for a primary ASN or a secondary ASN. The following table shows the bit which is selected from the byte as a function of bits 14 and 15 of the authorization index and the instruction PT-ss, SSAR-ss, or LOAD ADDRESS SPACE PARAMETERS (LASP).

Authorization-Index Bits		Bit Selected from Authority-Table Byte for Test	
14	15	P Bit (PT-ss)	S Bit (SSAR-ss or LASP)
0	0	0	1
0	1	2	3
1	0	4	5
1	1	6	7

If the selected bit is one, the ASN is authorized, and the appropriate address-space-control parameters from the AST entry are loaded into the appropriate control registers. If the selected bit is zero, the ASN is not authorized, and a primary-authority exception or secondary-authority exception is recognized for PT-ss or SSAR-ss, respectively. For LASP, when the ASN is not authorized, condition code 2 is set.

Recognition of Exceptions During ASN Authorization

The exceptions which can be encountered during the primary- and secondary-ASN-authorization processes and their priorities are described in the definitions of the instructions in which ASN authorization is performed.

Programming Note

The primary- and secondary-authority exceptions cause nullification to permit dynamic modification of the authority table. Thus, when an address space is created or "swapped in," the authority table can first be set to all zeros and the appropriate authority bits set to one only when required.

SEQUENCE OF STORAGE REFERENCES

Conceptually, the CPU processes instructions one at a time, with the execution of one instruction preceding the execution of the following instruction. The execution of the instruction specified by a successful branch follows the execution of the branch. Similarly, an interruption takes place between instructions or, for interruptible instructions, between units of operation of such instructions.

The sequence of events implied by the processing just described is sometimes called the conceptual sequence.

Each operation appears to the program to be performed sequentially, with the current instruction being fetched after the preceding operation is completed and before the execution of the current operation is begun. This appearance is maintained, even though the storage-implementation characteristics and overlap of instruction execution with storage accessing may cause actual processing to be different. The results generated are those that would have been obtained had the operations been performed in the conceptual sequence. Thus, it is possible for an instruction to modify the next succeeding instruction in storage.

In simple models in which operations are not overlapped, the conceptual and actual sequences are essentially the same. However, in more complex machines, overlapped operation, buffering of operands and results, and execution times which are comparable to the propagation delays between units can cause the actual sequence to differ considerably from the conceptual sequence. In these machines, special circuitry is employed to detect dependencies between operations and ensure that the results obtained, as observed by the CPU which generates them, are those that would have been obtained if the operations had been performed in the conceptual sequence. However, channels and other CPUs may, unless otherwise constrained, observe a sequence that differs from the conceptual sequence. Also, in certain situations involving dynamic address translation where different virtual addresses map to the same real address, the effect of its own overlapped operation may be observable by the CPU itself.

It can normally be assumed that the execution of each instruction occurs as an indivisible event. However, in actual operation, the execution of an instruction consists in a series of discrete steps. Depending on the instruction, operands may be fetched and stored in a piecemeal fashion, and some delay may occur between fetching operands and storing results. As a consequence, a channel or another CPU may be able to observe intermediate or partially completed results.

When a program on the CPU interacts with a program on a channel or on another CPU, the programs may have to take into consideration that a single operation may consist in a series of storage references, that a storage reference may in turn consist in a series of accesses, and that the conceptual and actual sequences of these accesses may differ. Storage references associated with instruction execution are of the following types: instruction fetches, DAT-table fetches, storage-key accesses, and storage-operand references.

Programming Note

The sequence of execution may differ from the simple conceptual definition in the following ways.

- As viewed by a program in the CPU, instructions may appear to be prefetched when different effective addresses are used. (See the section "Interlocks for Virtual-Storage References" in this chapter.)
- As viewed by a program in a channel or another CPU, the execution of an instruction may appear to be performed as a sequence of piecemeal steps. This is described for each type of storage reference in one of the following sections.
- As viewed by a program in a channel or another CPU, the storage-operand accesses associated with one instruction are not necessarily performed in the conceptual sequence. (See the section "Relation Between Operand Accesses" in this chapter.)
- As viewed by a program in a channel, in certain unusual situations, the contents of storage may appear to change and then be restored to the original value. (See the section "Storage Change and Restoration for DAT-Associated Access Exceptions" earlier in this chapter.)

INTERLOCKS FOR VIRTUAL-STORAGE REFERENCES

As described in the previous section, CPU operation appears to that CPU to be performed sequentially; the results stored by one instruction appear to the CPU to be completed before the next instruction is fetched. This appearance is maintained in overlapped machines by means of special circuitry to detect accesses to a common location by comparing effective addresses.

For purposes of this definition, the term "effective address" is used to denote the address before translation, if any, regardless of whether the address is virtual, real, or absolute. If two effective addresses have the same value and map to the same absolute location, the effective addresses are said to be the same even though one may be real or in a different address space.

When all accesses to a main-storage location are made by using the same effective address, then the above rule is strictly maintained, as observed by the CPU itself. When different effective addresses are used to access the common location, the above rule does not hold in two cases:

1. For some instructions, the definition specifies the results which must be obtained for overlapping operands. This definition is specified in terms of the sequence of the storage accesses; that is, the results of some or all of the stores of one operand must be placed in storage before some parts or all parts of the other operand are fetched. When the store and the fetch are performed by means of different effective addresses, then the operand may appear to be fetched before the store.
2. When an instruction changes the contents of a main-storage location from which a conceptually subsequent instruction is to be executed, either directly or by means of EXECUTE, and when different effective addresses are used to designate that location for storing the result and fetching the instruction, the instruction may appear to be fetched before the store occurs. This does not occur if an intervening operation causes the prefetched instructions to be discarded. A definition of when prefetched instructions must be discarded is included in the section "Instruction Fetching" in this chapter.

Any change to the storage key appears to be completed before the conceptually following reference to the associated storage block

When no copy of the page-table entry is in the TLB, the fetch of the associated segment-table entry precedes the fetch of the page-table entry.

STORAGE-KEY ACCESSES

References to the storage key are handled as follows:

1. Whenever a reference to storage is made and key-controlled protection applies to the reference, the four access-control bits and the fetch-protection bit associated with the storage location are inspected concurrently with the reference to the storage location.

2. When storing is performed, the change bit is set in the associated storage key concurrently with the store operation.

3. The instructions SET STORAGE KEY and SET STORAGE KEY EXTENDED cause all seven bits to be set concurrently in the storage key. The access to the storage key for SET STORAGE KEY and SET STORAGE KEY EXTENDED follow the sequence rules for storage-operand store references and is a single-access reference.

4. The instructions INSERT STORAGE KEY and INSERT STORAGE KEY EXTENDED provide a consistent image of bits 0-6 of the storage key. Similarly, the instructions INSERT VIRTUAL STORAGE KEY and TEST PROTECTION provide a consistent image of bits 0-4 of the storage key. The access to the storage key for all of these instructions follows the sequence rules for storage-operand fetch references and is a single-access reference.

5. The instructions RESET REFERENCE BIT and RESET REFERENCE BIT EXTENDED modify only the reference bit. All other bits of the storage key remain unchanged. The reference bit and change bit are examined concurrently to set the condition code. The access to the storage key for RESET REFERENCE BIT and RESET REFERENCE BIT EXTENDED follow the sequence rules for storage-operand update references. The reference bit is the only bit which is updated.

The record of references provided by the reference bit is not necessarily accurate, and the handling of the reference bit is not subject to the concurrency rules. However, in the majority of situations, reference recording approximately coincides with the storage reference.

The change bit may be set in cases when no storing has occurred. See the section "Change Recording" in Chapter 3, "Storage."

STORAGE-OPERAND REFERENCES

A storage-operand reference is the fetching or storing of the explicit operand or operands in the storage locations specified by the instruction.

During the execution of an instruction, all or some of the storage operands for that instruction may be fetched, intermediate results may be maintained for subsequent modification, and final results may be temporarily held prior to placing them in storage. Stores caused by channels do not necessarily affect these intermediate results. Storage-operand references are of three types: fetches, stores, and updates.

Storage-Operand Fetch References

When the bytes of a storage operand participate in the instruction execution only as a source, the operand is called a fetch-type operand, and the reference to the location is called a storage-operand fetch reference. A fetch-type operand is identified in individual instruction definitions by indicating that the access exception is for fetch.

All bits within a single byte of a fetch reference are accessed concurrently. When an operand consists of more than one byte, the bytes may be fetched from storage piecemeal, one byte at a time. Unless otherwise specified, the bytes are not necessarily fetched in any particular sequence.

Storage-Operand Store References

When the bytes of a storage operand participate in the instruction execution only as a destination, to the extent of being replaced by the result, the operand is called a store-type operand, and the reference to the location is called a storage-operand store reference. A store-type operand is identified in individual instruction definitions by indicating that the access exception is for store.

prior to all storage-operand references for all instructions that are conceptually later.

An instruction may be prefetched by using a virtual address only when the associated DAT table entries are attached and valid or when usable copies of the entries exist in the TLB. Instructions which are prefetched may be interpreted for execution only for the same virtual address for which the instruction was prefetched.

No limit is established as to the number of instructions which may be prefetched, and multiple copies of the contents of a single storage location may be fetched. As a result, the instruction executed is not necessarily the most recently fetched copy. Storing caused by channels or by other CPUs does not necessarily change the copy of prefetched instructions. However, if a store that is conceptually earlier occurs on the same CPU using the same effective address as that by which the instruction is subsequently fetched, the updated information is obtained.

All copies of prefetched instructions are discarded when:

- A serializing function is performed
- The CPU enters the operating state
- The CPU changes from DAT on to DAT off or from DAT off to DAT on
- A change is made to a translation parameter in control register 0 or 1 when DAT is on
- DAS is installed and the CPU changes from one to another of the following modes: primary-space mode, secondary-space mode, and real mode
- DAS is installed, and a change is made to a translation parameter in control register 7 when DAT is on

Programming Notes

1. As observed by a CPU itself, its own instruction prefetching is not normally apparent; the only exception occurs when multiple virtual addresses in a single address space, or virtual addresses in different address spaces, map to a single real address. This is described in the section "Interlocks for Virtual-Storage References" in this chapter.

2. The following are some effects of instruction prefetching on the execution of a program as viewed by another CPU.

   If a program in a channel or another CPU changes the contents of a storage location and then sets a flag to indicate that the change has been made, a CPU can test and find the flag set but subsequently can branch to the modified location and execute the original contents. Additionally, when a channel or another CPU modifies an instruction, it is possible for a CPU to recognize the changes to some but not all bit positions of the instruction.

   It is possible for a CPU to prefetch an instruction and subsequently, before the instruction is executed, for another CPU to change the storage key. As a result, a CPU may appear to execute instructions from a protected storage location.

DAT-TABLE FETCHES

The fetching of dynamic-address-translation (DAT) table entries may occur as follows:

1. A DAT-table entry may be prefetched into the translation-lookaside buffer (TLB) and used from the TLB without refetching from storage, until the entry is cleared by an INVALIDATE PAGE TABLE ENTRY, PURGE TLE, or SET PREFIX instruction or by CPU reset. DAT-table entries are not necessarily fetched in the sequence conceptually called for; they may be fetched at any time they are attached and valid, including during the execution of conceptually previous instructions.

2. All bytes of a DAT-table entry are fetched concurrently, as viewed by all CPUs in the configuration. However, the reference to the entry may appear to access a single byte at a time, as viewed by channels.

3. A DAT-table entry may be fetched even after some operand references for the instruction have already occurred. The fetch may occur as late as just prior to the actual byte access requiring the DAT-table entry.

4. A DAT-table entry may be fetched for each use of the address, including any trial execution, and for each reference to each byte of each operand.

5. The DAT page-table-entry fetch precedes the reference to the page.

is made, regardless of whether the reference to the storage location is made by a virtual, real, or absolute address. Analogously, any conceptually prior references to the storage block appear completed when the key for that block is changed or inspected.

Programming Note

A single main-storage location can be accessed in several ways by more than one address.

1. The DAT tables may be set up in such a way that multiple addresses in a single address space, or virtual addresses in different address spaces, map to a single real address.

2. The translation of logical, instruction, and virtual addresses may be changed by loading the DAT parameters in the control registers, by changing the address-space-control bit in the PSW, or, for logical and instruction addresses, by turning DAT on or off.

3. Certain instructions use real addresses.

4. Accesses to storage for the purpose of storing and fetching information for interruptions is performed by means of real addresses, and, for the store-status function, by means of absolute addresses, whereas accesses by the program may be by means of virtual addresses.

5. The real-to-absolute mapping may be changed by means of the SET PREFIX instruction or a reset.

6. A main-storage location may be accessed by channels by means of an absolute address and by the CPU by means of a real or a virtual address.

7. A main-storage location may be accessed by another CPU by means of one type of address and by this CPU by means of a different type of address.

8. The CPU updates the interval timer by means of a real address, and the program may access the location by means of a virtual address.

The primary purpose of this section is to describe the effects caused by case 1 above.

For case 2, the effect is not observable, since prefetched instructions are discarded and the effect of delayed stores is not observable to the CPU itself.

For case 3, those instructions which fetch by using real addresses (for example, LOAD REAL ADDRESS), no effect is observable. This is because the only effect across instructions is the prefetching of instructions, and instructions which fetch by using real addresses thus have no special effect. All instructions which store by using a real address or which store across address spaces cause prefetched instructions to be discarded, and no effect is observable.

Cases 4 and 5 are situations which are defined to cause serialization, with the result that prefetched instructions are discarded. In these cases, no effect is observable.

The handling of cases 6 and 7 involves accesses as observed by channels and other CPUs and is covered in the following sections in this chapter.

For case 8, the effect of updating the interval timer is observable only if an instruction is fetched from real location 80 or 82 by using a virtual address which is not 80 or 82, respectively.

INSTRUCTION FETCHING

Instruction fetching consists in fetching the one, two, or three halfwords specified by the instruction address in the current PSW. The immediate field of an instruction is accessed as part of an instruction fetch. If, however, an instruction specifies a storage operand at the location occupied by the instruction itself, the location is accessed both as an instruction and as a storage operand. The fetch of the target instruction of EXECUTE is considered to be an instruction fetch.

The bytes of an instruction may be fetched piecemeal and are not necessarily accessed in a left-to-right direction. The instruction may be fetched multiple times for a single execution; for example, it may be fetched for testing the addressability of operands or for inspection of PER events, and it may be refetched for actual execution.

Instructions are not necessarily fetched in the sequence in which they are conceptually executed and are not necessarily fetched for each time they are executed. In particular, the fetching of an instruction may precede the storage-operand references for an instruction that is conceptually earlier. The instruction fetch occurs

All bits within a single byte of a store reference are accessed concurrently. When an operand consists of more than one byte, the bytes may be placed in storage piecemeal, one byte at a time. Unless otherwise specified, the bytes are not necessarily stored in any particular sequence.

The CPU may delay placing results in storage. There is no defined limit on the length of time that results may remain pending before they are stored.

This delay does not affect the sequence in which results are placed in storage. The results of one instruction are placed in storage after the results of all preceding instructions have been placed in storage and before any results of the succeeding instructions are stored, as observed by channels and other CPUs. The results of any one instruction are stored in the sequence specified for that instruction.

The CPU does not fetch operands or DAT-table entries from a storage location until all information destined for that location by the CPU has been stored. Prefetched instructions may appear to be updated before the information appears in storage.

The stores are necessarily completed only as a result of a serializing operation and before the CPU enters the stopped state.

Storage-Operand Update References

In some instructions, the storage-operand location participates both as a source and as a destination. In these cases, the reference to the location consists first in a fetch and subsequently in a store. The operand is called an update-type operand, and the combination of the two accesses is referred to as an update reference. Instructions such as MOVE ZONES, TRANSLATE, OR (OC, OI), and ADD DECIMAL cause an update to the first-operand location. In most cases, no special interlock is provided between the fetch and store, and accesses by channels and other CPUs are permitted. An update-type operand is identified in the individual instruction definition by indicating that the access exception is for both fetch and store. The fetch and store accesses associated with an update reference do not necessarily occur one immediately after the other, and it is possible for a channel or another CPU to make one or more interleaved accesses to the same location. The interleaved accesses can be either fetches or stores.

The following instructions perform an update which is interlocked against accesses by another CPU to the same location during the execution of the instruction. The instructions TEST AND SET, COMPARE AND SWAP, and COMPARE DOUBLE AND SWAP cause an interlocked update. On models in which the STORE CHARACTERS UNDER MASK instruction with a mask of zero fetches and stores the byte designated by the second-operand address, the fetch and store accesses are an interlocked update.

The fetch and store accesses associated with an interlocked-update reference do not necessarily occur one immediately after the other, but all accesses by another CPU are prevented from occurring between the fetch and the store accesses of an interlocked update. Channel accesses may occur during the interlock period.

Within the limitations of the above requirements, the fetch and store accesses associated with an update follow the same rules as the fetches and stores described in the previous sections.

Programming Notes

1. When two CPUs attempt to update information at a common main-storage location by an instruction that causes fetching and subsequently storing of the updated information, it is possible for both CPUs to fetch the information and subsequently make the store access. The change made by the first CPU to store the result in such a case is lost. Similarly, if one CPU updates the contents of a field but another CPU makes a store operation to that field between the fetch and store parts of the update reference, the effect of the store is lost. If, instead of a store access, a CPU makes an interlocked-update reference to the common storage field between the fetch and store portions of an update due to another CPU, any change in the contents produced by the interlocked update is lost.

2. Only those bytes which are included in the result field of both operations are considered to be part of the common main-storage location. However, all bits within a common byte are considered to be common even if the bits modified by the two operations do not overlap. As an example, if (1) one CPU executes the instruction OR (OC) with a length of 1 and the value 80 hex in the second-operand location and (2) the other CPU executes AND (NC) with a length of 1 and the value FE hex in

the second-operand location, and (3) the first operand of both instructions is the same byte, then one of the updates can be lost.

3. When the store access is part of an update reference by the CPU, the execution of the storing is not necessarily contingent on whether the information to be stored is different from the original contents of the location. In particular, the contents of all designated byte locations are replaced, and, for each byte in the field, the entire contents of the byte are replaced.

   Depending on the model, an access to store information may be performed, for example, in the following cases:

   a. Execution of the OR instruction (OI or OC) with a second operand of all zeros.

   b. Execution of OR (OC) with the first- and second-operand fields coinciding.

   c. For those locations of the first operand of TRANSLATE where the argument and function values are the same.

4. The instructions TEST AND SET, COMPARE AND SWAP, and COMPARE DOUBLE AND SWAP facilitate updating of a common storage field by two CPUs. In order for the change by either CPU not to be lost, both CPUs must use an instruction providing an interlocked update. It is possible, however, for a channel to make an access to the same storage location between the fetch and store portions of an interlocked update.

STORAGE-OPERAND CONSISTENCY

Single-Access References

A fetch reference is said to be a single-access reference if the value is fetched in a single access to each byte of the data field. In the case of overlapping operands, the location may be accessed once for each operand. A store-type reference is said to be a single-access reference if a single store access occurs to each byte location within the data field. An update reference is said to be single-access if both the fetch and store accesses are each single-access.

Except for the accesses associated with multiple-access operands and the stores associated with storage change and restoration for DAT-associated access exceptions, all storage-operand references are single-access references.

Multiple-Access Operands

For some instructions, multiple accesses may be made to all or some of the bytes of a storage operand. The following cases are those storage-operand references which may be multiple-access references:

1. The storage decimal operands of the following instructions: CONVERT TO BINARY, CONVERT TO DECIMAL, MOVE INVERSE, MOVE WITH OFFSET, PACK, TEST BLOCK, and UNPACK.

2. The operands of the decimal instructions.

3. The stores into that portion of the first operand of MOVE LONG which is filled with padding bytes.

4. The stores into a DAS-trace entry.

When a storage-operand store reference to a location is not a single-access reference, the contents placed at a byte location are not necessarily the same for each store access; thus, intermediate results in a single-byte location may be observed by channels and other CPUs.

Programming Notes

1. When multiple fetch accesses are made to a single byte that is being changed by a channel or another CPU, the result is not necessarily limited to that which could be obtained by fetching the bits individually. For example, the execution of MULTIPLY DECIMAL may consist in repetitive additions and subtractions each of which causes the second operand to be fetched from storage.

2. When CPU instructions are used to modify storage locations being accessed by a channel simultaneously, multiple store accesses to a single byte by the CPU may result in intermediate values being observed by the channel. To avoid these intermediate values (especially when

modifying a CCW chain), only instructions making single-access references should be used.

Block-Concurrent References

For some references, the accesses to all bytes within a halfword, word, or doubleword are specified to be concurrent as observed by other CPUs. These accesses do not necessarily appear to a channel to include more than a byte at a time. The halfword, word, or doubleword is referred to in this section as a block. When a fetch-type reference is specified to be concurrent within a block, no store access to the block by another CPU is permitted during the time that bytes contained in the block are being fetched. Channel accesses to the bytes within the block may occur between the fetches. When a store-type reference is specified to be concurrent within a block, no access to the block, either fetch or store, is permitted by another CPU during the time that the bytes within the block are being stored. Channel accesses to the bytes in the block may occur between the stores.

Consistency Specification

The storage-operand references associated with all S-format and RX-format instructions, with the exception of EXECUTE, CONVERT TO DECIMAL, and CONVERT TO BINARY, are block-concurrent, as observed by all CPUs, if the operand is addressed on a boundary which is integral to the size of the operand.

For the instructions COMPARE AND SWAP and COMPARE DOUBLE AND SWAP, all accesses to the storage operand appear to be block-concurrent as observed by all CPUs.

The instructions LOAD MULTIPLE and STORE MULTIPLE, when the operand starts on a word boundary, and the instructions COMPARE LOGICAL (CLC), COMPARE LOGICAL CHARACTERS UNDER MASK, INSERT CHARACTERS UNDER MASK, and STORE CHARACTERS UNDER MASK access their storage operands in a left-to-right direction, and all bytes accessed within each doubleword appear to all CPUs to be accessed concurrently.

The instructions LOAD CONTROL and STORE CONTROL access the storage operand in a left-to-right direction, and all bytes accessed within each word appear to all CPUs to be accessed concurrently.

When destructive overlap does not exist, the operands of MOVE (MVC), MOVE WITH KEY (MVCK), MOVE TO PRIMARY (MVCP), and MOVE TO SECONDARY (MVCS) are accessed as follows:

1. The first operand is accessed in a left-to-right direction, and all bytes accessed within a doubleword appear to all CPUs to be accessed concurrently.

2. The second operand is accessed left to right, and all bytes within a doubleword in the second operand that are moved into a single doubleword in the first operand appear to all CPUs to be fetched concurrently. Thus, if the first and second operands begin on the same byte offset within a doubleword, the second operand appears to be fetched doubleword-concurrent. If the offsets within a doubleword differ by 4, the second operand appears to be fetched word-concurrent.

Destructive overlap is said to exist when the result location is used as a source after the result has been stored, assuming processing to be performed one byte at a time.

The operands for MOVE LONG and COMPARE LOGICAL LONG appear to all CPUs to be accessed doubleword-concurrent when both operands start on doubleword boundaries and are an integral number of doublewords in length, and, for MOVE LONG, execution is in the nonpadding portion and the operands do not overlap.

For EXCLUSIVE OR (XC), the operands are processed in a left-to-right direction, and, when the first and second operands coincide, all bytes accessed within a doubleword appear to all CPUs to be accessed concurrently.

Programming Note

In the case of EXCLUSIVE OR (XC) designating operands which coincide exactly, the bytes within the field may appear to be accessed as many as three times, by two fetches and one store: once as the fetch portion of the first operand update, once as the second-operand fetch, and then once as the store portion of the first-operand update. Each of the three accesses appears to all CPUs to be doubleword-concurrent, but the three accesses do not necessarily appear to occur one immediately after the other. One or both fetch accesses may be omitted since the instruction can be completed without fetching the operands.

RELATION BETWEEN OPERAND ACCESSES

Storage-operand fetches associated with one instruction execution appear to precede all storage-operand references for conceptually subsequent instructions. A storage-operand store specified by one instruction appears to precede all storage-operand stores specified by conceptually subsequent instructions, but it does not necessarily precede storage-operand fetches specified by conceptually subsequent instructions. However, a storage-operand store appears to precede a conceptually subsequent storage-operand fetch from the same main-storage location.

When an instruction has two storage operands both of which cause fetch references, it is unpredictable which operand is fetched first, or how much of one operand is fetched before the other operand is fetched. When the two operands overlap, the common locations may be fetched independently for each operand.

When an instruction has two storage operands the first of which causes a store and the second a fetch reference, it is unpredictable how much of the second operand is fetched before the results are stored. In the case of destructively overlapping operands, the portion of the second operand which is common to the first is not necessarily fetched from storage.

When an instruction has two storage operands the first of which causes an update reference and the second a fetch reference, it is unpredictable which operand is fetched first, or how much of one operand is fetched before the other operand is fetched. Similarly, it is unpredictable how much of the result is processed before it is returned to storage. In the case of destructively overlapping operands, the portion of the second operand which is common to the first is not necessarily fetched from storage.

Programming Note

The independent fetching of a single location for each of two operands may affect the program execution in the following situation.

When the same storage location is designated by two operand addresses of an instruction, and a channel or another CPU causes the contents of the location to change during execution of the instruction, the old and new values of the location may be used simultaneously. For example, comparison of a field to itself may yield a result other than equal, or EXCLUSIVE-ORing of a field to itself may yield a result other than zero.

OTHER STORAGE REFERENCES

The restart, program, SVC, external, I/O, and machine-check PSWs are accessed doubleword-concurrent as observed by other CPUs. These references occur after the conceptually previous unit of operation and before the conceptually subsequent unit of operation. The relationship between the new-PSW fetch, the old-PSW store, and the interruption-code store is unpredictable.

Store accesses for interruption codes not stored within the old PSW are not necessarily single-access stores. The external and SVC interruption-code stores occur between the conceptually previous and conceptually subsequent operations. The program interruption-code store accesses may precede the storage-operand references associated with the instruction which results in the program interruption.

The CSW and I/O-communication-area stores occur within the conceptual limits of the interruption or I/O instruction with which they are associated.

Updating of the interval timer occurs after storage-operand references for the conceptually previous instruction and before storage-operand references for the conceptually subsequent instruction. Interval-timer updates can also occur within an interruptible instruction between units of operation.

SERIALIZATION

The sequence of functions performed by a CPU is normally independent of the functions performed by channels and other CPUs. Similarly, the sequence of functions performed by a channel is normally independent of the functions performed by other channels and CPUs. However, at certain points in its execution, serialization of the CPU occurs. Serialization also occurs at certain points for channels.

CPU SERIALIZATION

All interruptions and the execution of certain instructions cause a serialization of CPU operations. A serialization

operation consists in completing all conceptually previous storage accesses by the CPU, as observed by channels and other CPUs, before the conceptually subsequent storage accesses occur. Serialization affects the sequence of all CPU accesses to storage and to the storage keys, except for those associated with DAT-table-entry fetching.

Serialization is performed by all interruptions and by the execution of the following instructions:

1. The general instructions BRANCH ON CONDITION (BCR) with the M_1 and R_2 field containing all ones and all zeros, respectively, and COMPARE AND SWAP, COMPARE DOUBLE AND SWAP, STORE CLOCK, SUPERVISOR CALL, and TEST AND SET.

2. LOAD PSW, SET STORAGE KEY, AND SET STORAGE KEY EXTENDED.

3. All I/O instructions, CONNECT CHANNEL SET, and DISCONNECT CHANNEL SET.

4. PURGE TLB and SET PREFIX, which also cause the translation-lookaside buffer to be cleared of entries.

5. SIGNAL PROCESSOR, READ DIRECT, and WRITE DIRECT.

6. INVALIDATE PAGE TABLE ENTRY.

7. TEST BLOCK.

8. MOVE TO PRIMARY, MOVE TO SECONDARY, PROGRAM CALL, PROGRAM TRANSFER, SET ADDRESS SPACE CONTROL, and SET SECONDARY ASN.

9. The DAS-tracing function causes serialization to be performed before the trace action and after completion of the trace action.

The sequence of events associated with a serializing operation is as follows:

- All conceptually previous storage accesses by the CPU are completed, as observed by channels and other CPUs. This includes all conceptually previous stores and changes to the storage keys.

- The normal function associated with the serializing operation is performed. In the case of instruction execution, operands are fetched, and the storing of results is completed. The exceptions are LOAD PSW and SET PREFIX, in which the operand may be fetched before previous stores have been completed, and interruptions, in which the interruption code and associated fields may be stored prior to the serialization. The fetching of the serializing instruction occurs before the execution of the instruction and may precede the execution of previous instructions, but may not precede the completion of any previous serializing operation. In the case of an interruption, the old PSW, the interruption code, and other information, if any, are stored, and the new PSW is fetched, but not necessarily in that sequence.

- Finally, instruction fetch and operand accesses for conceptually subsequent operations may begin.

A serializing function affects the sequence of storage accesses that are under the control of the CPU in which the serializing function takes place. It does not affect the sequence of storage accesses under the control of a channel or another CPU.

Programming Notes

1. The following are some effects of a serializing operation:

   a. When an instruction changes the contents of a storage location that is used as a source of a following instruction and when different addresses are used to designate the same absolute location for storing the result and fetching the instruction, a serializing operation following the change ensures that the modified instruction is executed.

   b. When a serializing operation takes place, the channel and any other CPUs observe instruction and operand fetching and result storing to take place in the sequence established by the serializing operation.

2. Storing into a location from which a serializing instruction is fetched does not necessarily affect the execution of the serializing instruction unless a serializing function has been performed after the storing and before the execution of the serializing instruction.

CHANNEL-PROGRAM SERIALIZATION

Serialization of a channel program occurs as follows:

1. For a channel program, all storage accesses and storage-key accesses by the channel follow initiation of the execution of START I/O or START I/O FAST RELEASE, as observed by programs in CPUs or another channel. This includes all accesses for the CAW, CCWs, IDAWs, and data.

2. For a channel program, all storage accesses and storage-key accesses are completed, as observed by programs in CPUs or another channel, before the interruption condition indicating channel end is presented to the CPU.

3. For a channel program, if a CCW in the chain contains a PCI bit which is one, all storage accesses and storage-key accesses due to CCWs preceding it in the chain are completed, as observed by programs in CPUs or another channel, before the PCI condition is presented to the CPU.

The serialization of a channel program does not affect the sequence of storage accesses or storage-key accesses caused by programs in CPUs or another channel. It also does not affect the sequence of storage accesses or storage-key accesses caused by other channel programs on the same channel.

CHAPTER 6. INTERRUPTIONS

Source Identification	Interruption Code	PSW-Mask Bits EC	PSW-Mask Bits BC	Mask Bits in Ctrl Registers Reg, Bit	ILC Set	Execution of Instruction Identified by Old PSW
MACHINE CHECK (old PSW 48, new PSW 112)	Locations 232-239[1]					
Exigent condition		13	13		x	terminated or nullified[2]
Repressible cond		13	13	14, 4-7	x	unaffected[2]
SUPERVISOR CALL (old PSW 32, new PSW 96)	Locations 138-139 in EC mode and 34-35 in BC mode					
Instruction bits	00000000 ssssssss				1,2	completed
PROGRAM (old PSW 40, new PSW 104)	Locations 142-143 in EC mode and 42-43 in BC mode					
Operation	00000000 p0000001				1,2,3	suppressed
Privileged oper	00000000 p0000010				1,2	suppressed
Execute	00000000 p0000011				2	suppressed
Protection	00000000 p0000100				0,1,2,3	suppressed or terminated
Addressing	00000000 p0000101				0,1,2,3	suppressed or terminated
Specification	00000000 p0000110				0,1,2,3	suppressed or completed
Data	00000000 p0000111				2,3	suppressed or terminated
Fixed-pt overflow	00000000 p0001000	20	36		1,2	completed
Fixed-point divide	00000000 p0001001				1,2	suppressed or completed
Decimal overflow	00000000 p0001010	21	37		2,3	completed
Decimal divide	00000000 p0001011				2,3	suppressed
Exponent overflow	00000000 p0001100				1,2	completed
Exponent underflow	00000000 p0001101	22	38		1,2	completed
Significance	00000000 p0001110	23	39		1,2	completed
Floating-pt divide	00000000 p0001111				1,2	suppressed
Segment transl	00000000 p0010000				1,2,3	nullified
Page translation	00000000 p0010001				1,2,3	nullified
Translation spec	00000000 p0010010				1,2,3	suppressed
Special operation	00000000 p0010011			0, 1	2	suppressed
ASN-transl spec	00000000 p0010111				2	suppressed
Space-switch event	00000000 p0011100			1, 31	2	completed
PC-transl spec	00000000 p0011111				2	suppressed
AFX translation	00000000 p0100000				2	nullified
ASX translation	00000000 p0100001				2	nullified
LX translation	00000000 p0100010				2	nullified
EX translation	00000000 p0100011				2	nullified
Primary authority	00000000 p0100100				2	nullified
Secondary auth	00000000 p0100101				2	nullified
Monitor event	00000000 p1000000			8, 16+	2	completed
PER event	00000000 1nnnnnnn[3]	1	*	9, 0-3@	0,1,2,3	completed[4]

Interruption Action (Part 1 of 2)

The interruption facility permits the CPU to change its state as a result of conditions external to the configuration, within the configuration, or within the CPU itself. To permit fast response to conditions of high priority and immediate recognition of the type of condition, interruption conditions are grouped into six classes: external, input/output, machine check, program, restart, and supervisor call.

INTERRUPTION ACTION

An interruption consists in storing the current PSW as an old PSW, storing information identifying the cause of the interruption, and fetching a new PSW. Processing resumes as specified by the new PSW.

The old PSW stored on an interruption normally contains the address of the instruction that would have been executed next had the interruption not occurred, thus permitting resumption of the interrupted program. For program and supervisor-call interruptions, the information stored also contains a code that identifies the length of the last-executed instruction, thus permitting the program to respond to the cause of the interruption. In the case of some program conditions for which the normal response is reexecution of the instruction causing the interruption, the instruction address directly identifies the instruction last executed.

Except for restart, an interruption can occur only when the CPU is in the operating state. The restart interruption can occur with the CPU in either the stopped or operating state.

The details of source identification, location determination, and instruction execution are explained in later sections and are summarized in the figure "Interruption Action."

Source Identification	Interruption Code	PSW-Mask Bits EC	PSW-Mask Bits BC	Mask Bits in Ctrl Registers Reg, Bit	ILC Set	Execution of Instruction Identified by Old PSW
EXTERNAL (old PSW 24, new PSW 88)	Locations 134–135 in EC mode and 26–27 in BC mode					
Interval timer	00000000 1eeeeeee	7	7	0, 24	x	unaffected
Interrupt key	00000000 e1eeeeee	7	7	0, 25	x	unaffected
External signal 2	00000000 ee1eeeee	7	7	0, 26	x	unaffected
External signal 3	00000000 eee1eeee	7	7	0, 26	x	unaffected
External signal 4	00000000 eeee1eee	7	7	0, 26	x	unaffected
External signal 5	00000000 eeeee1ee	7	7	0, 26	x	unaffected
External signal 6	00000000 eeeeee1e	7	7	0, 26	x	unaffected
External signal 7	00000000 eeeeeee1	7	7	0, 26	x	unaffected
Malfunction alert	00010010 00000000	7	7	0, 16	x	unaffected
Emergency signal	00010010 00000001	7	7	0, 17	x	unaffected
External call	00010010 00000010	7	7	0, 18	x	unaffected
TOD-clock sync chk	00010000 00000011	7	7	0, 19	x	unaffected
Clock comparator	00010000 00000100	7	7	0, 20	x	unaffected
CPU timer	00010000 00000101	7	7	0, 21	x	unaffected
Service signal	00100100 00000001	7	7	0, 22	x	unaffected
INPUT/OUTPUT (old PSW 56, new PSW 120)	Locations 186–187 in EC mode and 58–59 in BC mode					
Channel 0	00000000 dddddddd	6	0	2, 0[5]	x	unaffected
Channel 1	00000001 dddddddd	6	1	2, 1[5]	x	unaffected
Channel 2	00000010 dddddddd	6	2	2, 2[5]	x	unaffected
Channel 3	00000011 dddddddd	6	3	2, 3[5]	x	unaffected
Channel 4	00000100 dddddddd	6	4	2, 4[5]	x	unaffected
Channel 5	00000101 dddddddd	6	5	2, 5[5]	x	unaffected
Channel 6 & up	cccccccc dddddddd	6	6	2, 6+	x	unaffected
RESTART (old PSW 8, new PSW 0)	Locations 2–3 in BC mode					
Restart key	00000000 00000000[6]				x	unaffected

Explanation:

Locations for the old PSWs, new PSWs, and interruption codes are real locations.

[1] A model-independent machine-check interruption code of 64 bits is stored at real locations 232–239. In the BC mode, the contents of real locations 50–51 are unpredictable.

[2] The effect of the machine-check condition is identified by the validity bits in the machine-check interruption code. The unit of operation is nullified or unaffected only if all the associated validity bits are ones.

[3] When the interruption code indicates a PER event, an ILC of 0 may be stored only when bits 8–15 of the interruption code are 10000110 (PER, specification).

[4] The unit of operation is completed, unless a program exception concurrently indicated causes the unit of operation to be nullified, suppressed, or terminated.

[5] For channels 0–5, channel masks in control register 2 have no effect in the BC mode.

[6] Bits 16–31 in the old PSW in the BC mode are set to zeros. No interruption code is provided in the EC mode.

\+ Plus the following bits in the control register. One mask bit is provided for each channel; the bit position matches the channel number.

* In the BC mode, PER is disabled.

@ Additional masks in control register 9, bit positions 16–31, provide detailed control over the source of general-register-alteration PER events which are masked by control register 9, bit 3.

c Channel-address bits.

d Device-address bits.

e If one, the bit indicates another concurrent external-interruption condition.

n A possible nonzero code, indicating another concurrent program-interruption condition.

p If one, the bit indicates a concurrent PER-event interruption condition.

s Bits of the I field of SUPERVISOR CALL.

x Unpredictable in the BC mode; not stored in the EC mode.

Interruption Action (Part 2 of 2)

SOURCE IDENTIFICATION

The six classes of interruptions (external, I/O, machine check, program, restart, and supervisor call) are distinguished by the storage locations at which the old PSW is stored and from which the new PSW is fetched. For most classes, the causes are further identified by an interruption code and, for some classes, by additional information placed in permanently assigned real storage locations during the interruption. (See also the section "Assigned Storage Locations" in Chapter 3, "Storage.") For external, I/O, program, and supervisor-call interruptions, the interruption code consists of 16 bits.

For external interruptions in the EC mode, the interruption code is stored at real locations 134-135. In the BC mode, the interruption code is placed in the old PSW. A parameter may be stored at real locations 128-131, or a CPU address may be stored at real locations 132-133.

For I/O interruptions in the EC mode, the interruption code, which contains the I/O address, is stored at real locations 186-187. In the BC mode, the interruption code is placed in the old PSW. Additional information is provided by the contents of the channel-status word (CSW) stored at real location 64. Further information may be provided by the limited channel logout stored at real locations 176-197 and by a full channel logout stored in the fixed-logout area (real locations 256-351) or in the I/O-extended-logout area.

For machine-check interruptions, the interruption code consists of 64 bits and is stored at real locations 232-239. Additional information for identifying the cause of the interruption and for recovering the state of the machine may be provided by the contents of the machine-check failing-storage address, the external-damage code, the region code, and the contents of the fixed-logout, extended-logout, and machine-check-save areas. (See Chapter 11, "Machine-Check Handling.")

For program interruptions in the EC mode, the interruption code is stored at real locations 142-143, and the instruction-length code is stored in bit positions 5 and 6 of real location 141. In the BC mode, the interruption code and instruction-length code are placed in the old PSW. Further information may be provided in the form of the translation-exception identification, monitor-class number, monitor code, PER code, and PER address, which are stored at real locations 144-159.

For restart interruptions in the EC mode, no interruption code is stored. In the BC mode, an interruption code of zero is placed in the old PSW.

For supervisor-call interruptions in the EC mode, the interruption code is stored at real locations 138-139, and the instruction-length code is stored in bit positions 5 and 6 of real location 137. In the BC mode, the interruption code and instruction-length code are placed in the old PSW.

ENABLING AND DISABLING

By means of mask bits in the current PSW and in control registers, the CPU may be enabled or disabled for all external, I/O, and machine-check interruptions and for some program interruptions. When a mask bit is one, the CPU is enabled for the corresponding class of interruptions, and these interruptions can occur.

When a mask bit is zero, the CPU is disabled for the corresponding interruptions. The conditions that cause I/O interruptions remain pending. External-interruption conditions either remain pending or persist until the cause is removed. Machine-check-interruption conditions, depending on the type, are ignored, remain pending, or cause the CPU to enter the check-stop state. The disallowed program-interruption conditions are ignored, except that some causes are indicated also by the setting of the condition code. The setting of the significance and exponent-underflow program-mask bits affects the manner in which floating-point operations are completed when the corresponding condition occurs.

The CPU is always enabled for program interruptions for which mask bits are not provided, as well as the supervisor-call and restart interruptions.

The mask bits may allow or disallow all interruptions within the class, or they may selectively allow or disallow interruptions for particular causes. This control may be provided by mask bits in the PSW that are assigned to particular causes, such as the bits assigned to the four maskable program-interruption conditions. Alternatively, there may be a hierarchy of masks, where a mask bit in the PSW controls all interruptions within a type, and mask bits in a control register provide more detailed control over the sources.

When the mask bit is one, the CPU is enabled for the corresponding interruptions. When the mask bit is zero, these

interruptions are disallowed. Interruptions that are controlled by a hierarchy of masks are allowed only when all controlling mask bits are ones.

Programming Notes

1. Mask bits in the PSW provide a means of disallowing all maskable interruptions; thus, subsequent interruptions can be disallowed by the new PSW introduced by an interruption. Furthermore, the mask bits can be used to establish a hierarchy of interruption priorities, where a condition in one class can interrupt the program handling a condition in another class but not vice versa. To prevent an interruption-handling routine from being interrupted before the necessary housekeeping steps are performed, the new PSW must disable the CPU for further interruptions within the same class or within a class of lower priority.

2. Because the mask bits in control registers are not changed as part of the interruption procedure, these masks cannot be used to prevent an interruption immediately after a previous interruption in the same class. The mask bits in control registers provide a means for selectively enabling the CPU for some sources and disabling it for others within the same class.

HANDLING OF FLOATING INTERRUPTION CONDITIONS

An interruption condition which can be presented to any CPU in the configuration is called a floating interruption condition. The condition is presented to the first CPU in the configuration which is enabled for the corresponding interruption and which can accept the interruption, and then the condition is cleared and not presented to any other CPU in the configuration. A CPU cannot accept the interruption when it is in the check-stop state, has an invalid prefix, is in a string of program interruptions due to a specification exception of the type which is recognized early,, is executing a READ DIRECT instruction, or is in the stopped state. However, a CPU with the rate control set to instruction step can accept the interruption when the start key is activated.

Service signal and certain machine-check conditions are floating interruption conditions.

INSTRUCTION-LENGTH CODE

The instruction-length code (ILC) occupies two bit positions and provides the length of the last instruction executed. It permits identifying the instruction causing the interruption when the instruction address in the old PSW designates the next sequential instruction. The ILC is provided also by the BRANCH AND LINK instructions.

When the old PSW specifies the EC mode, the ILC for program and supervisor-call interruptions is stored in bit positions 5 and 6 of the bytes at real locations 141 and 137, respectively. For external, I/O, machine-check, and restart interruptions, the ILC is not stored since it cannot be related to the length of the last-executed instruction.

When the old PSW specifies the BC mode, the ILC is stored in bit positions 32 and 33 of that PSW. The ILC is meaningful, however, only after a supervisor-call or program interruption. For machine-check, external, I/O, and restart interruptions, the ILC does not indicate the length of the last-executed instruction and is unpredictable. Similarly, the ILC is unpredictable in the PSW stored during execution of the store-status function and when the PSW is displayed.

For supervisor-call and program interruptions, a nonzero ILC identifies in halfwords the length of the instruction that was last executed. Whenever an instruction is executed by means of EXECUTE, instruction-length code 2 is set to indicate the length of EXECUTE and not that of the target instruction.

The value of a nonzero instruction-length code is related to the leftmost two bits of the instruction. The value does not depend on whether the operation code is assigned or on whether the instruction is installed. The following table summarizes the meaning of the instruction-length code:

ILC		Instr Bits	Instruction Length
Decimal	Binary	0–1	
0	00		Not available
1	01	00	One halfword
2	10	01	Two halfwords
2	10	10	Two halfwords
3	11	11	Three halfwords

Zero ILC

Instruction-length code 0, after a program interruption, indicates that the instruction address stored in the old PSW does not identify the instruction causing the interruption.

An ILC of 0 occurs when a specification exception due to a PSW-format error is recognized as part of early exception recognition and the PSW has been introduced by LOAD PSW or an interruption. (See the section "Exceptions Associated with the PSW" later in this chapter.) In the case of LOAD PSW, the instruction address of the LOAD PSW or EXECUTE has been replaced by the instruction address of the new PSW. When the invalid PSW is introduced by an interruption, the PSW-format error cannot be attributed to an instruction.

On some models without the translation facility, an ILC of zero occurs also when an addressing exception or a protection exception is recognized for a store-type reference. In these cases, the interruption due to the exception is delayed, the length of time or number of instructions of the delay being unpredictable. Neither the instruction address of the instruction causing the exception nor the length of the last-executed instruction is made available to the program. This type of interruption is sometimes referred to as an imprecise program interruption.

In the case of LOAD PSW and the supervisor-call interruption, a PER event may be indicated concurrently with a specification exception having an ILC of 0.

ILC on Instruction-Fetching Exceptions

When a program interruption occurs because of an exception that prohibits access to the instruction, the instruction-length code cannot be set on the basis of the first two bits of the instruction. As far as the significance of the ILC for this case is concerned, the following two situations are distinguished:

1. When an odd instruction address causes a specification exception to be recognized or when an addressing, protection, or translation-specification exception is encountered on fetching an instruction, the ILC is set to 1, 2, or 3, indicating the multiple of 2 by which the instruction address has been incremented. It is unpredictable whether the instruction address is incremented by 2, 4, or 6. By reducing the instruction address in the old PSW by the number of halfword locations indicated in the ILC, the instruction address originally appearing in the PSW may be obtained.

2. When a segment-translation or page-translation exception is recognized while fetching an instruction, including the target instruction of EXECUTE, the ILC is arbitrarily set to 1, 2, or 3. In this case, the operation is nullified, and the instruction address is not incremented.

The ILC is not necessarily related to the first two bits of the instruction when the first halfword of an instruction can be fetched but an access exception is recognized on fetching the second or third halfword. The ILC may be arbitrarily set to 1, 2, or 3 in these cases. The instruction address is or is not updated, as described in situations 1 and 2 above.

When any exceptions other than segment translation or page translation are encountered on fetching the target instruction of EXECUTE, the ILC is 2.

Programming Notes

1. A nonzero instruction-length code for a program interruption indicates the number of halfword locations by which the instruction address in the program old PSW must be reduced to obtain the instruction address of the last instruction executed, unless one of the following situations exists:

   a. The interruption is caused by an exception resulting in nullification.

   b. An interruption for a PER alone event occurs before the execution of an interruptible instruction is completed.

   c. The interruption is caused by a PER event due to LOAD PSW or a branch or linkage instruction, including SUPERVISOR CALL (but not including MONITOR CALL).

   d. The interruption is caused by an access exception encountered in fetching an instruction, and the instruction address has been introduced into the PSW by a means other than sequential operation (by a branch instruction, LOAD PSW, or an interruption).

e. The interruption is caused by a specification exception because of an odd instruction address.

f. The interruption is caused by an early specification exception or by an access exception encountered in fetching an instruction, and changes have been made to a parameter that controls the relation between instruction addresses and real addresses. The relation between instruction addresses and real addresses can be changed without introducing an entire new PSW by switching from real mode, primary-virtual mode, or secondary-virtual mode to a different mode, or by changing one or more of the translation parameters in control registers 0, 1, and 7. The early specification exception can be caused by executing STORE THEN OR SYSTEM MASK or SET SYSTEM MASK, which switches to or from real mode while introducing invalid values in bit positions 0-7 of an EC-mode PSW.

For situations a and b above, the instruction address in the PSW is not incremented, and the instruction designated by the instruction address is the same as the last one executed. These situations are the only ones in which the instruction address in the old PSW identifies the instruction causing the exception.

For situations c, d, and e, the instruction address has been replaced as part of the operation, and the address of the last instruction executed cannot be calculated using the one appearing in the program old PSW.

For situation f, the instruction address in the PSW has not been replaced, but the corresponding real address after the change may be different.

2. When a PER event is indicated, bit 8 in the interruption code is one, the PER address in the word at real location 152 identifies the instruction causing the interruption (or the EXECUTE instruction, as appropriate); and the instruction-length code (ILC) is redundant. Similarly, the ILC is redundant when the operation is nullified, since in this case the instruction address in the PSW is not incremented. If the ILC value is required in this case, it can be derived from the operation code of the instruction identified by the old PSW.

EXCEPTIONS ASSOCIATED WITH THE PSW

Exceptions associated with erroneous information in the current PSW may be recognized when the information is introduced into the PSW or may be recognized as part of the execution of the next instruction. Errors in the PSW which are specification-exception conditions are called PSW-format errors.

Early Exception Recognition

For the following error conditions, a program interruption for a specification exception occurs immediately after the PSW becomes active:

- The EC mode is specified (PSW bit 12 is one) in a CPU that does not have the translation facility installed.
- Bit position 16 of an EC-mode PSW is one and DAS is not installed.
- A one is introduced into an unassigned bit position of an EC-mode PSW (that is, any of bit positions 0, 2-4, 17, or 24-39)

The interruption occurs regardless of whether the wait state is specified. If the invalid PSW causes the CPU to become enabled for a pending I/O, external, or machine-check interruption, the program interruption occurs instead, and the pending interruption is subject to the mask bits of the new PSW introduced by the program interruption. If the EC mode is not present, bits 0-15 and 34-63 of the invalid PSW are stored unchanged at the corresponding bit positions of the program old PSW, and the interruption code and instruction-length code are stored at bit positions 16-33 of the program old PSW.

When the execution of LOAD PSW or an interruption introduces a PSW with one of the above error conditions, the instruction-length code is set to 0, and the newly introduced PSW, except for the interruption code and the instruction-length code in the BC mode, is stored unmodified as the old PSW. When one of the above error conditions is introduced by execution of SET SYSTEM MASK or STORE THEN OR SYSTEM MASK, the instruction-length code is set to 2, and the instruction address is incremented by 4. The PSW containing the invalid value introduced into the system-mask field is stored as the old PSW.

When a PSW with one of the above error conditions is introduced during initial program loading, the loading sequence is

not completed, and the load indicator remains on.

Late Exception Recognition

For the following conditions, the exception is recognized as part of the execution of the next instruction:

- A specification exception is recognized due to an odd instruction address in the PSW (PSW bit 63 is one).
- An access exception (addressing, page-translation, protection, segment-translation, or translation-specification) is associated with the location designated by the instruction address or with the location of the second or third halfword of the instruction starting at the designated instruction address.

The instruction-length code and instruction address stored in the program old PSW under these conditions are discussed in the section "ILC on Instruction-Fetching Exceptions" in this chapter.

If the invalid PSW causes the CPU to be enabled for a pending I/O, external, or machine-check interruption, the corresponding interruption occurs, and the PSW invalidity is not recognized. Similarly, the specification or access exception is not recognized in a PSW specifying the wait state.

Programming Notes

1. The execution of LOAD PSW, SET SYSTEM MASK, STORE THEN AND SYSTEM MASK, and STORE THEN OR SYSTEM MASK is suppressed on an addressing or protection exception, and hence the program old PSW provides information concerning the program causing the exception.
2. When the first halfword of an instruction can be fetched but an access exception is recognized on fetching the second or third halfword, the ILC is not necessarily related to the operation code.
3. If the new PSW introduced by an interruption contains a PSW-format error, a string of interruptions may occur (See the section "Priority of Interruptions" in this chapter.)

EXTERNAL INTERRUPTION

The external interruption provides a means by which the CPU responds to various signals originating either from within or from without the configuration.

An external interruption causes the old PSW to be stored at real location 24 and a new PSW to be fetched from real location 88.

The source of the interruption is identified in the interruption code. When the old PSW specifies the EC mode, the interruption code is stored at real locations 134-135. When the old PSW specifies the BC mode, the interruption code is placed in bit positions 16-31 of the old PSW, and the instruction-length code is unpredictable.

Additionally, for the malfunction-alert, emergency-signal, and external-call conditions, a 16-bit CPU address is associated with the source of the interruption and is stored at real locations 132-133 in both the EC and BC modes. When the CPU address is stored, bit 6 of the interruption code is set to one. For all other conditions, no CPU address is stored, and bit 6 of the interruption code is set to zero. When bit 6 is zero and the old PSW specifies the EC mode, zeros are stored at real locations 132-133. When bit 6 is zero and the old PSW specifies the BC mode, the contents of real locations 132-133 remain unchanged.

For the service-signal interruption, a 32-bit parameter is associated with the interruption and is stored at real locations 128-131 in both EC and BC modes. Bit 2 of the external-interruption code indicates that a parameter has been stored. When bit 2 is zero, the contents of real locations 128-131 remain unchanged.

External-interruption conditions are of two types: those for which an interruption request condition is held pending, and those for which the condition directly requests the interruption. Clock comparator, CPU timer, and TOD-clock sync check are conditions which directly request external interruptions. If a condition which directly requests an external interruption is removed before the request is honored, the request does not remain pending, and no interruption occurs. Conversely, the request is not cleared by the interruption, and if the condition persists, more than one interruption may result from a single occurrence of the condition.

When several interruption requests for a single source are generated before the interruption occurs, and the interruption

condition is of the type which is held pending, only one request for that source is preserved and remains pending.

An external interruption for a particular source can occur only when the CPU is enabled for interruption by that source. The external interruption occurs at the completion of a unit of operation. Whether the CPU is enabled for external interruption is controlled by the external mask, PSW bit 7, and external subclass-mask bits in control register 0. Each source for an external interruption has a subclass-mask bit assigned to it, and the source can cause an interruption only when the external-mask bit is one and the corresponding subclass-mask bit is one. The use of the subclass-mask bits does not depend on whether the CPU is in the EC or BC mode.

When the CPU becomes enabled for a pending external-interruption condition, the interruption occurs at the completion of the instruction execution or interruption that causes the enabling.

More than one source may present a request for an external interruption at the same time. When the CPU becomes enabled for more than one concurrently pending request, the interruption occurs for the pending condition or conditions having the highest priority.

The priorities for external-interruption requests in descending order are as follows:

Interval timer, interrupt key, external signals 2-7
Malfunction alert
Emergency signal
External call
TOD-clock sync check
Clock comparator
CPU timer
Service signal

The interval timer, interrupt key, and the external signals 2-7 are of equal priority; if more than one of these conditions is pending and allowed, the conditions are indicated concurrently. All other requests are honored one at a time. When more than one emergency-signal request exists at a time or when more than one malfunction-alert request exists at a time, the request associated with the smallest CPU address is honored first.

CLOCK COMPARATOR

An interruption request for the clock comparator exists whenever either of the following conditions is met:

1. The TOD clock is in the set or not-set state, and the value of the clock comparator is less than the value in the compared portion of the TOD clock, both compare values being considered unsigned binary integers.

2. The clock comparator is installed, and the TOD clock is in the error or not-operational state.

If the condition responsible for the request is removed before the request is honored, the request does not remain pending, and no interruption occurs. Conversely, the request is not cleared by the interruption, and, if the condition persists, more than one interruption may result from a single occurrence of the condition.

When the TOD clock accessed by a CPU is set or changes state, interruption conditions, if any, that are due to the clock comparator may or may not be recognized for up to 1.048576 seconds after the change.

The clock-comparator condition is indicated by an external-interruption code of 1004 (hex).

The subclass-mask bit is in bit position 20 of control register 0. This bit is initialized to zero.

CPU TIMER

An interruption request for the CPU timer exists whenever the CPU-timer value is negative (bit 0 of the CPU timer is one). If the value is made positive before the request is honored, the request does not remain pending, and no interruption occurs. Conversely, the request is not cleared by the interruption, and, if the condition persists, more than one interruption may occur from a single occurrence of the condition.

When the TOD clock accessed by a CPU is set or changes state, interruption conditions, if any, that are due to the CPU timer may or may not be recognized for up to 1.048576 seconds after the change.

The CPU-timer condition is indicated by an external-interruption code of 1005 (hex).

The subclass-mask bit is in bit position 21 of control register 0. This bit is initialized to zero.