Published: 01/31/67

Identification

Interrupt Interceptor
L. J. Lambert

Purpose

The Interrupt Interceptor is the interface between the hardware interrupt cells of the system controllers and the interrupt handlers. Its primary functions are to save the processor state, mask the processor for further interrupts of equal or lower priority and call the interrupt handler for this interrupt. Upon return from the interrupt handler, it restores the processor state to continue the interrupted procedure. The Interrupt Interceptor module must execute privileged instructions and inhibit interrupts during certain crucial operations. It therefore is a machine language master mode procedure that is entered only as the result of an interrupt.

Interrupts

Interrupts within Multics are of two types. System interrupts are those which arise from I/O devices and system clocks and are not necessarily directed at the process currently running on the interrupted processor. Process interrupts (time out, pre-empt, quit) are directed to the process running on the processor at the time of interrupt. Process interrupts are triggered by a processor operating in the Traffic Controller module. Interrupt priority assignments are detailed in BC.1.04.

Stack Usage

For every interrupt that occurs the Interrupt Interceptor requires a storage area for the processor state, temporary storage for itself and for making calls to interrupt handlers. This storage area is called an interrupt frame (BK.1.03). System interrupts cause interrupt frames to be established in the Processor Stack. Process interrupts establish interrupt frames in the Process Concealed Stack. Stack usage is the only distinction made by the Interrupt Interceptor between process and system interrupts.

Interrupt Interceptor Actions

The following actions are taken by the Interrupt Interceptor when a system or process interrupt is recognized.

1. The processor state is stored in the current interrupt frame in the appropriate stack. Pointers to the current interrupt frame are maintained at the base of the Processor Stack (BK.1.03) and Process Concealed Stack. The processor state is stored in three steps:

   a. The processor control unit is stored (store control unit instruction) using the scu pointer.

   b. The arithmetic registers are stored (store registers instruction) using the sreg pointer.

   c. The address base registers are stored (store bases instructions) using the stb pointer.

   d. The ring number in which the processor was executing at the time of interrupt is stored as part of the scu data.

2. The base registers are loaded to establish the standard pairings for the Interrupt Interceptor and provide linkage.

3. The descriptor base register is loaded with the value of the ring 0 descriptor segment.

4. A new interrupt frame is allocated in preparation for the next interrupt. Since a processor may be interrupted in the course of handling an earlier interrupt, care is taken to prevent a possible overlapping of the new interrupt frame and the interim stack in use at the instant of the interrupt. A new interrupt frame is allocated as follows:

   a. The value of the stack pointer, sp, is obtained. If the stack is not "empty", the sp-sb base register pair stored in Step 1 (assumes sp-sb pair is locked) points to the base location of the stack frame in use at the instant of the interrupt and sp|18 points to the base location of the next (intended stack frame (i.e. next sp). In this case, a constant, k, is added to the value of the next sp to obtain the base location of the new interrupt frame. (Currently, k is equal to 32.) If the stack is empty at the instant an interrupt occurs, then the new interrupt frame is allocated immediately following the current interrupt frame. In this case, the base location of the new interrupt frame is obtained by adding 24 (i.e. the length of an interrupt frame) to the stb pointer at the base of the stack.

b. A back pointer to the previous interrupt frame is fabricated and stored into locations 22-23 of the new interrupt frame.

c. The stb, sreg, and scu pointers stored at the base of the stack are adjusted to point to the appropriate areas within the new interrupt frame.

5. A new interim stack is created immediately following the new interrupt frame. The sp-sb address base register pair is set to point to the base location of the new interim stack.

6. The value of the next sp is determined and stored at sp|18 in the first stack frame of the new interim stack. The amount of temporary storage required by the Interrupt Interceptor is used in determining the value to be stored at sp|18.

7. The system controller interrupt mask register is stored in the temporary storage of the first stack frame of the new interim stack. This step is repeated for all system controllers for which this processor is designated control processor.

8. The system clock is read to establish the time of interrupt recognition. This value is stored in the temporary storage of the Interrupt Interceptor.

9. A mask that will inhibit recognition of interrupts of equal or lower priority is set in the system controller interrupt mask register. This step is repeated for all system controllers for which this processor is designated control processor.

10. A standard CALL is issued to the interrupt handler module determined from the Interrupt Decode Table (discussed below). This module may CALL other modules. Until a RETURN is made to the Interrupt Interceptor, all stack management is performed by the standard CALL, SAVE, and RETURN sequence. When control returns to the Interrupt Interceptor, the sequence of steps presented here continues.

11. The scu, sreg, and stb pointers stored at the base of the stack are restored from locations 22-23 of the current interrupt frame to point to the base location of the previous interrupt frame.

12. The system controller interrupt mask register is restored to its state at the instant the interrupt occurred. This step is repeated for all system controllers for which this processor is designated control processor.

13. The descriptor base register is loaded with the value for the ring in which the processor was executing at time of interrupt. This information was stored with the scu data.

14. The processor is restored to its state at the instant the interrupt occurred. This is done as follows:

    a. The arithmetic registers are restored (load registers instruction) using the sreg pointer.

    b. The address base registers are restored (load bases instruction) using the stb pointer.

    c. The processor control unit is restored (restore control unit instruction) using the scu pointer. When the processor control unit is restored, control returns to the point at which the interrupt occurred.

All of the above steps are executed in master mode with interrupts inhibited. The CALL in step 9 is also performed in inhibited mode on the chance that some interrupt handler may require continuous inhibition. (It should be noted that the standard CALL, SAVE, sequence for master mode procedures does not allow inhibition through all of the SAVE).

Decoding of interrupts and processor masking, two of the Interrupt Interceptor functions, are discussed in greater detail below. Figure 1 will prove helpful in following the description.

Interrupt Decoding and Processor Masking

Interrupts signaled by a system controller force the execution of an execute double instruction (xed) by the processor designated control processor for the system controller sending the interrupt signal. The operand (called an interrupt word pair) of the forced execute double instruction is determined uniquely as a combination of the processor base switches, (18 toggle switches on the processor maintenance panel), the processor port through which the interrupt

signal was received (0-7) and the interrupt cell number (0-31) specified by the system controller. Each interrupt word pair contains instructions of the form

```
scu     abs_1,*
tra     abs_2,*
```

abs_1 - is the absolute address of a pointer (its pair) which points to the area in the current interrupt frame used to stored the processor control unit.

abs_2 - is the absolute address of a pointer (its pair) which points to a unique entry of the Interrupt Interceptor.

These word pairs and pointer, as well as the Interrupt Decode Table described below, are constructed at initialization time by the Interrupt Interceptor Initializer (BL.3.04).

The number of interrupt word pairs is equal to the number of unique interrupt signals that system controllers send to control processors. Interrupt word pairs which should never be set are all directed to a trouble entry of the Interrupt Interceptor (entry_XXX).

Interrupt Decode Table

This table contains information needed to identify interrupts, perform the masking function and notify the responsible handler. For each possible cause of an interrupt an entry provides the following information.

1. Interrupt handler pointer - This is a pointer (its pair) to an appropriate interrupt handler entry point for this interrupt.

2. Active device number - This is a number (precision 18 bits) used to identify similar active devices (e.g. GIOC1 = 1, GIOC2 = 2). See the individual interrupt handlers for actual specification.

3. Interrupt number - This is a number (precision 18 bits) used to define uniquely the interrupts from an active device. See the individual interrupt handlers for assignments.

4. Mask table pointer generator - This is an effective address to pair (eap) instruction used to generate a pointer to the mask table entry for this interrupt. The mask table is located in the System Communication Segment. Its format is described below.

Mask Table

The mask table because of its sensitivity to hardware configuration changes is located in the System Communication Segment. It contains the following information on a per interrupt basis.

1. The number of masks to be loaded when this interrupt occurs. This is also equal to the number of system controllers for which the interrupted processor is control processor.

2. A pointer (its pair) which points to a "special segment" (see BK.1.04). The effective address generated by referencing this segment specifies the system controller to be masked.

3. A 72 bit mask to be set in the system controller specified by 2 above.

Items 2 and 3 repeated once for each system controller for which the interrupted processor is control processor.

Interrupt Handler Call

The Interrupt Interceptor after decoding the interrupt and masking the processor calls the interrupt handler as follows.

call int_handler (active_device, int_number, time_ptr)

int_handler - interrupt handler entry point

active_device - described previously

int_number - described previously

time_ptr - a pointer to the system clock reading taken when this interrupt was recognized.

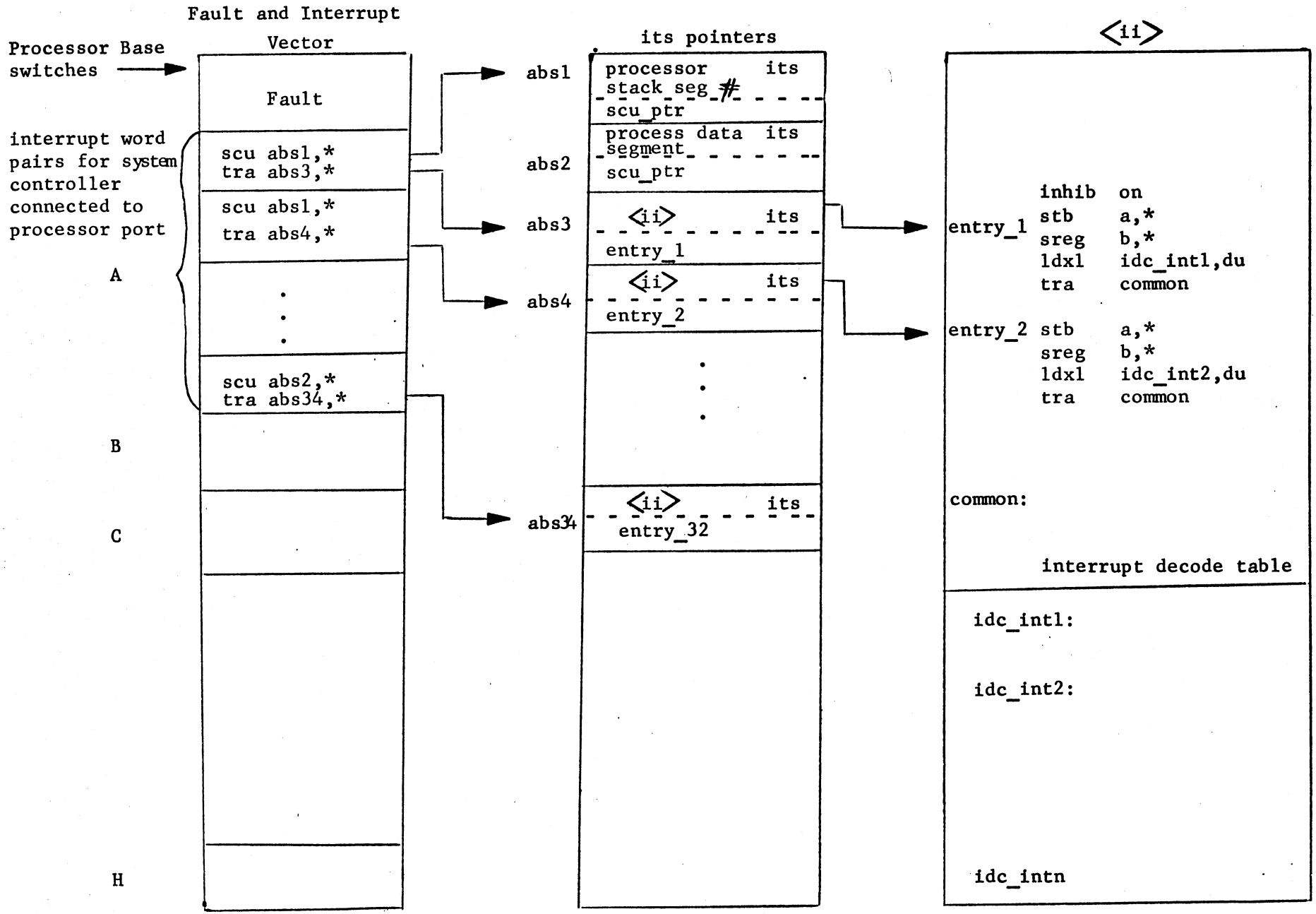
Fault and Interrupt Vector
its pointers
<ii>
Processor Base switches
Fault
interrupt word pairs for system controller connected to processor port
scu abs1,*
tra abs3,*
scu abs1,*
tra abs4,*
scu abs2,*
tra abs34,*
A
B
C
H
abs1
abs2
abs3
abs4
abs34
processor stack_seg #
its
scu_ptr
process data segment
its
scu_ptr
<ii>
its
entry_1
<ii>
its
entry_2
<ii>
its
entry_32
entry_1
inhib on
stb a,*
sreg b,*
ldxl idc_intl,du
tra common
entry_2
stb a,*
sreg b,*
ldxl idc_int2,du
tra common
common:
interrupt decode table
idc_intl:
idc_int2:
idc_intn

Figure 1