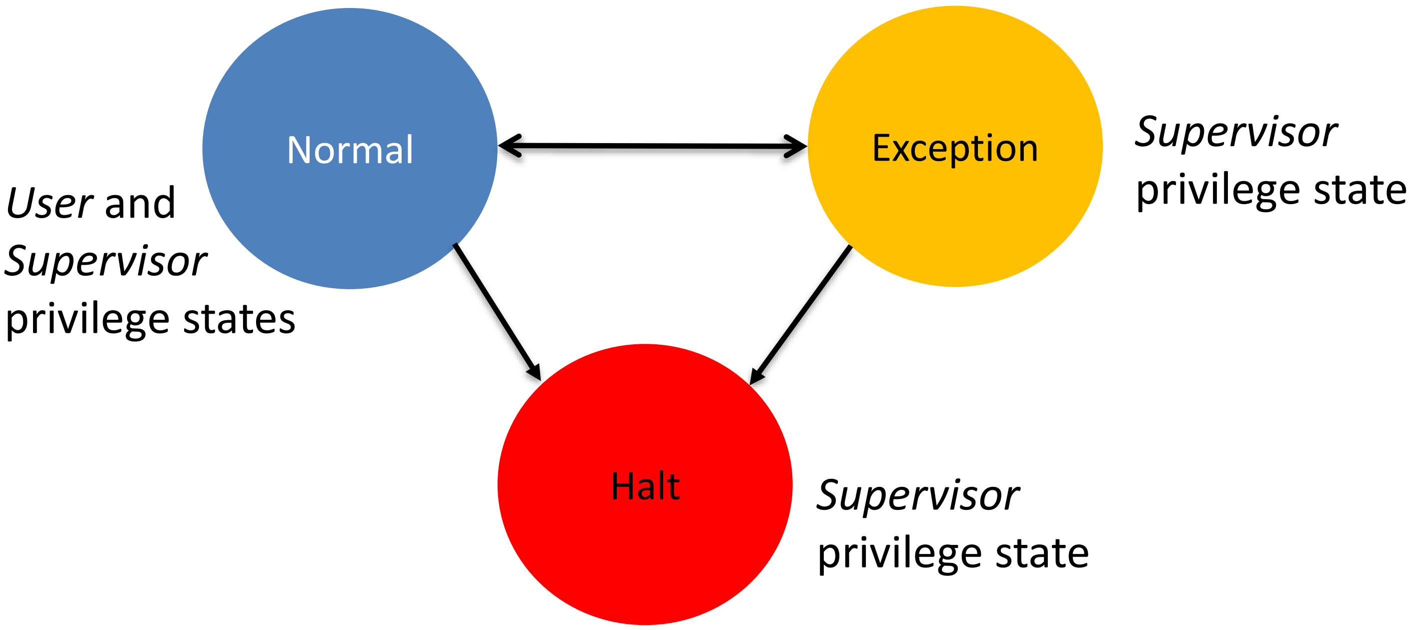
Execution/Privilege States

- The 68000 always in one of three processing states:
  - Normal
  - Halt
  - Exception



Normal

Exception

Halt

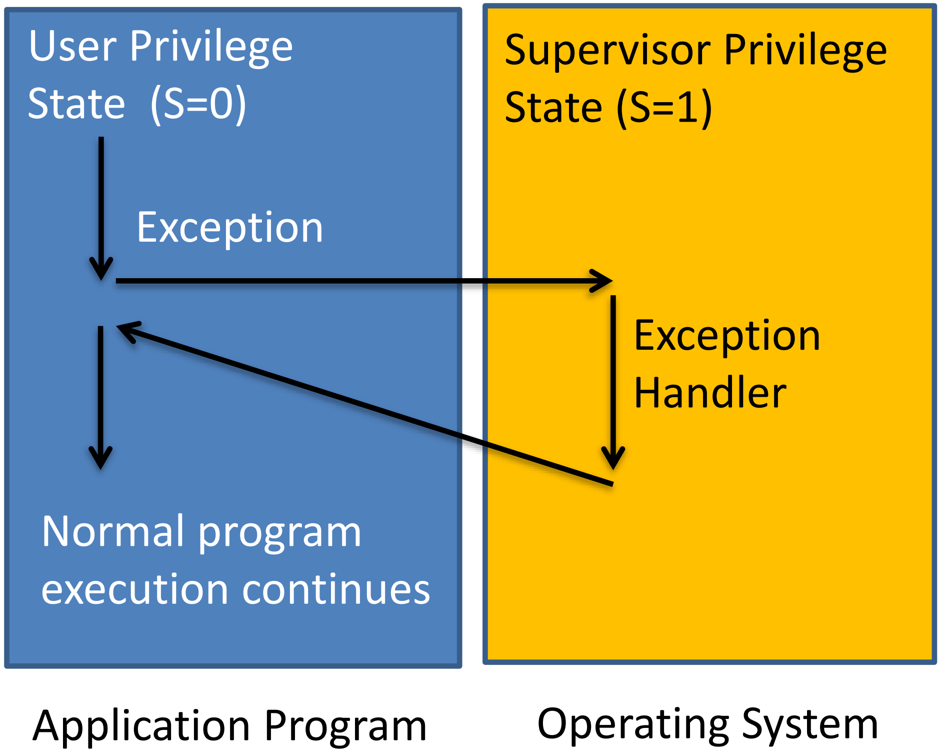
User and *Supervisor* privilege states

Supervisor privilege state

Supervisor privilege state

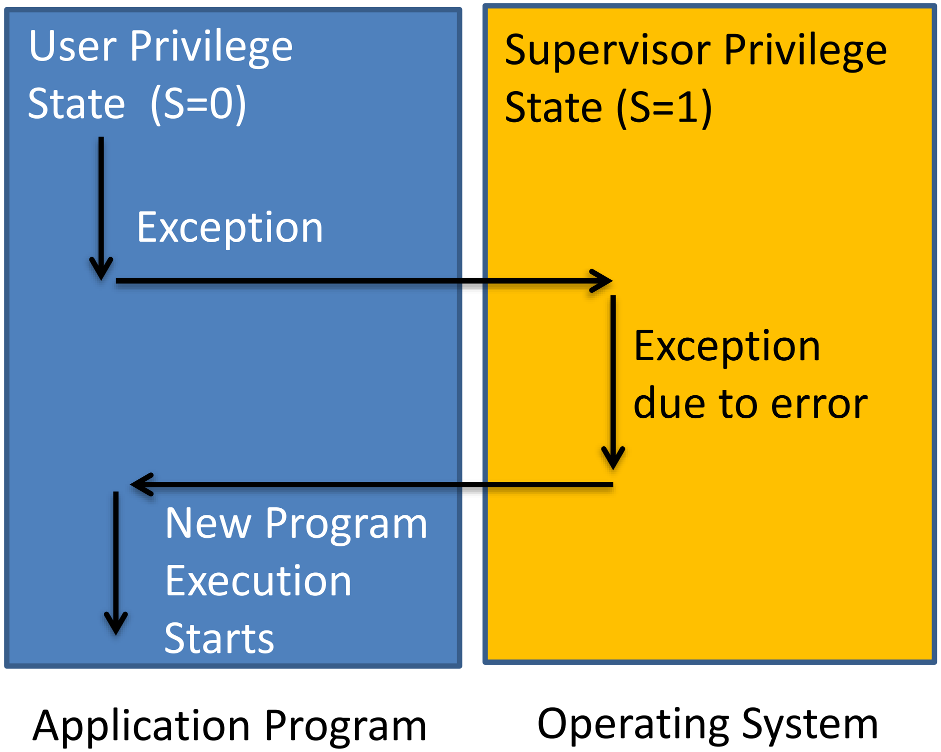
Exceptions in General

- Generated either in hardware or software, an *exception* causes the currently executing code to be effectively paused while the event is serviced by the operating system


User Privilege State (S=0)
Exception
Normal program execution continues
Supervisor Privilege State (S=1)
Exception Handler
Application Program
Operating System

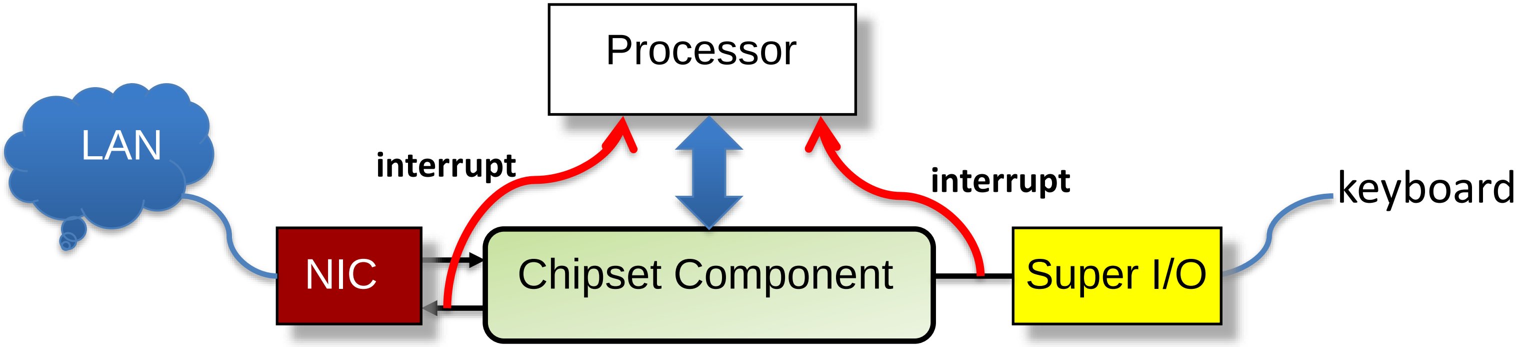
Exceptions in General

- Generated either in hardware or software, an *exception* causes the currently executing code to be effectively paused while the event is serviced by the operating system


User Privilege State (S=0)
Exception
New Program Execution Starts
Supervisor Privilege State (S=1)
Exception due to error
Application Program
Operating System

Hardware Exceptions

- Hardware generated exceptions are
  - called “interrupts”
  - asynchronous events
  - serviced only if external interrupt recognition is enabled (interrupt mask)



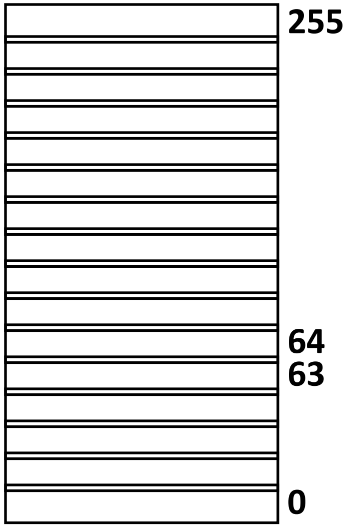
Software Exceptions

- Software generated exceptions are
  - called “software interrupts” or just “exceptions”
  - synchronous events
    - e.g., TRAP #N
  - serviced regardless of the state of the interrupt mask

- Processor generated exceptions occur when the processor attempts to execute an instruction that requires additional attention
  - e.g., trying to divide by zero
  - e.g., trying to access memory that is “not present” in RAM
  - e.g., page fault (virtual memory)

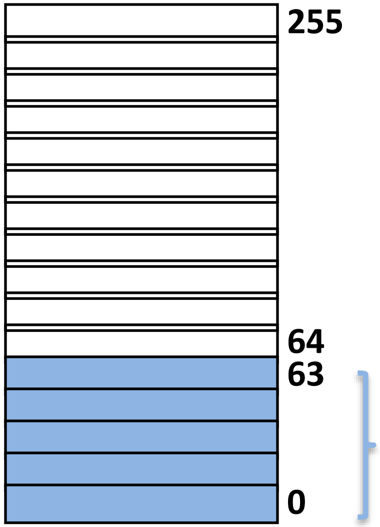
Vector Number

- Each exception is represented (identified) by a Vector Number
  - A vector number is an 8-bit (unsigned) value that identifies the exception and is defined by the ISA
- The vector number is used to locate the service routine to handle the specific event
  - Interrupt Service Routine (ISR) in the case of a hardware exception
  - Exception Handler (EH) in case of a software exception



Exceptions Cont’d

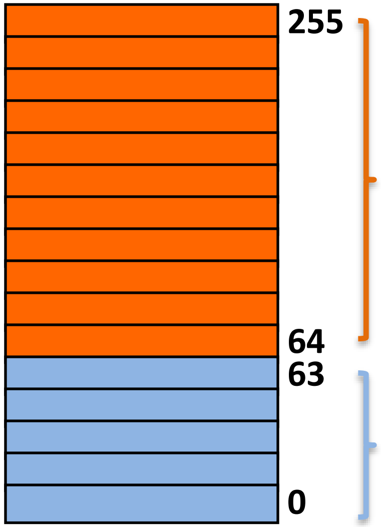
- Each exception is represented (identified) by a Vector Number
  - A vector number is an 8-bit (unsigned) value that identifies the exception and is defined by the ISA
- The vector number is used to locate the service routine to handle the specific event
  - Interrupt Service Routine (ISR) in the case of a hardware exception
  - Exception Handler (EH) in case of a software exception



Vector Numbers 0-63 are reserved for (software) exceptions

Exceptions Cont'd

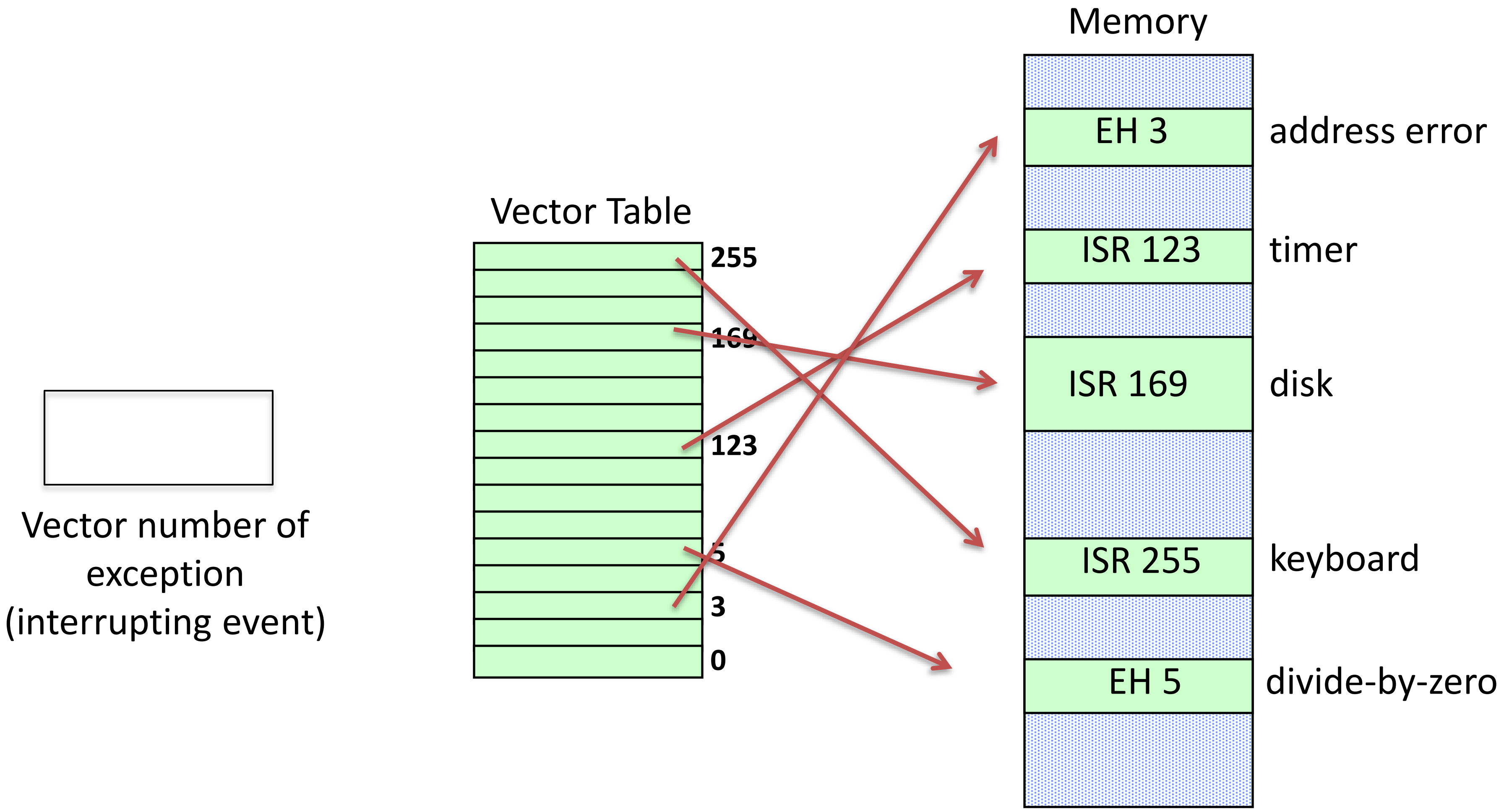
- Each exception is represented (identified) by a Vector Number
  - A vector number is an 8-bit (unsigned) value that identifies the exception and is defined by the ISA
- The vector number is used to locate the service routine to handle the specific event
  - Interrupt Service Routine (ISR) in the case of a hardware exception
  - Exception Handler (EH) in case of a software exception



Vector Numbers 64 - 255 are reserved for hardware (exceptions) interrupts

Vector Numbers 0-63 are reserved for (software) exceptions

Finding the Exception Handler or ISR to Run



Vector number of exception (interrupting event)

Vector Table

255

169

123

5

3

0

Memory

EH 3 address error

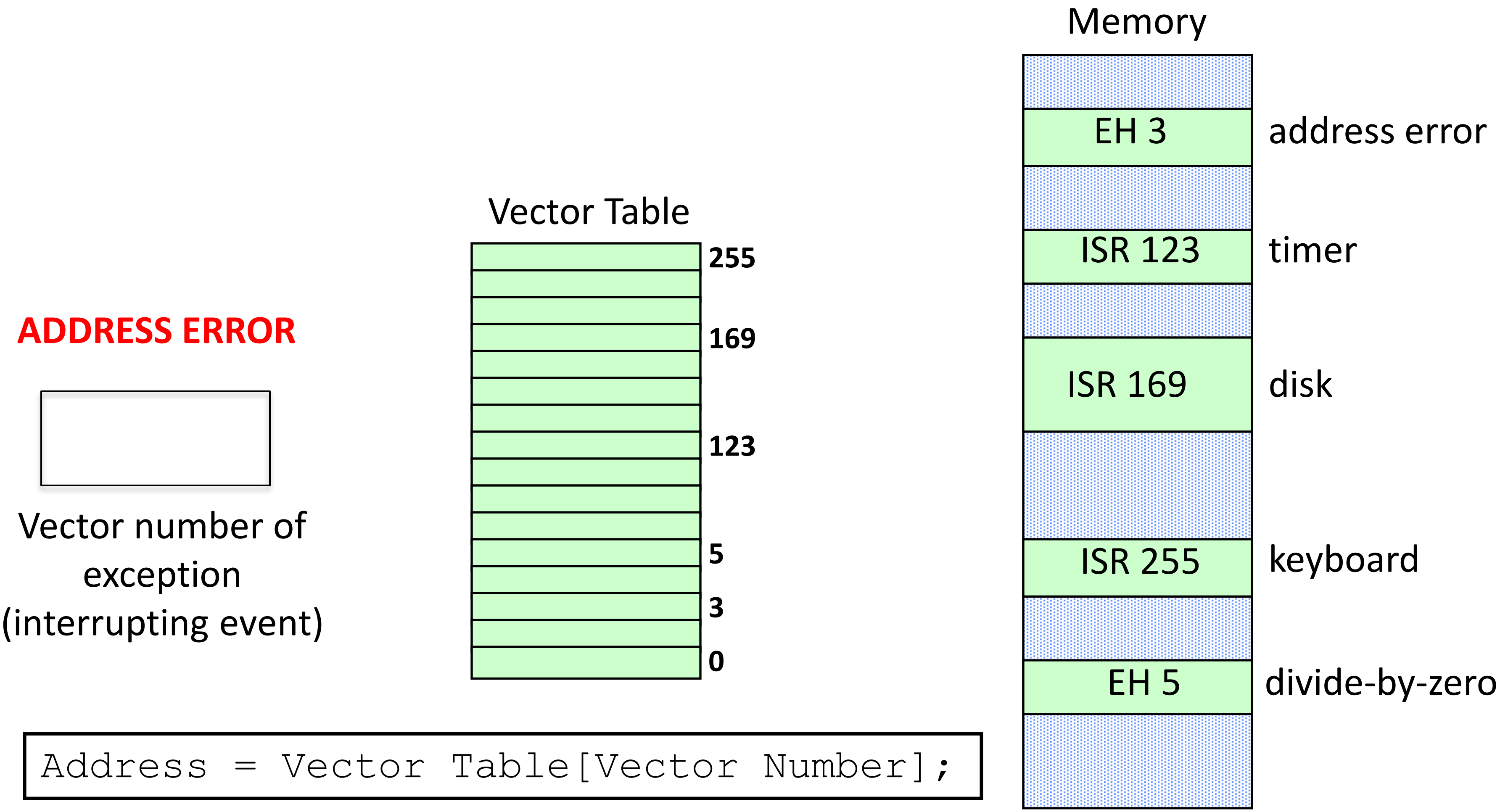
ISR 123 timer

ISR 169 disk

ISR 255 keyboard

EH 5 divide-by-zero

Finding the Exception Handler or ISR to Run

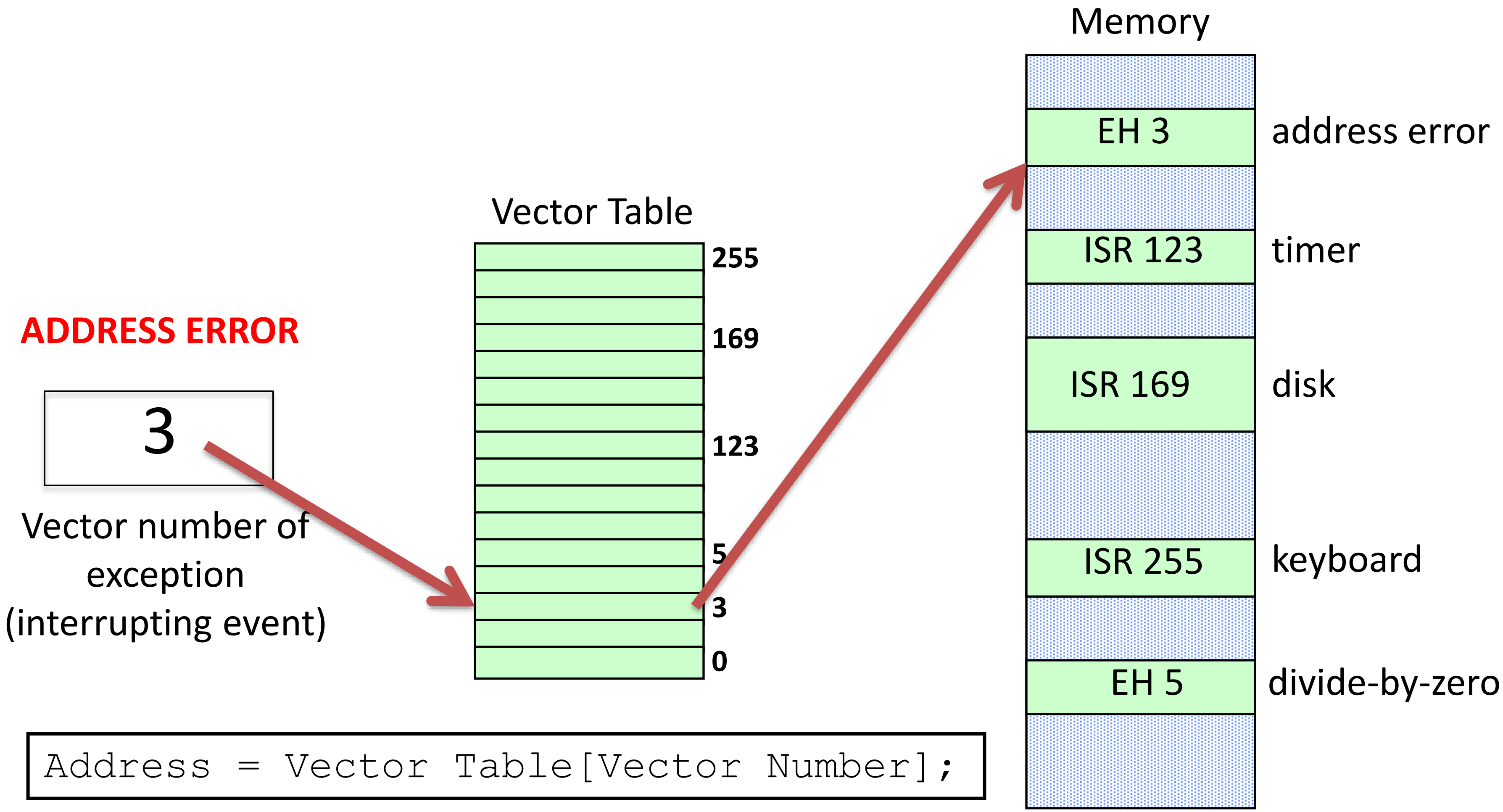


ADDRESS ERROR

Vector number of exception (interrupting event)

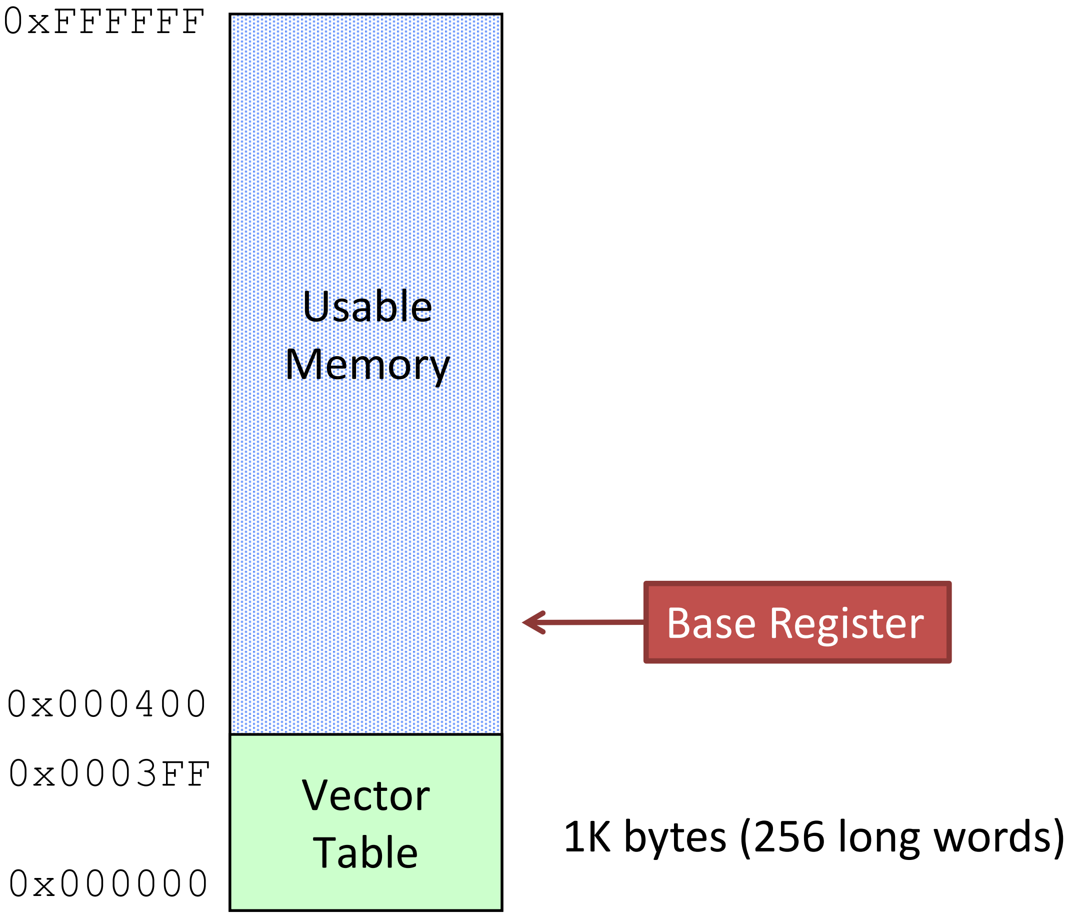
```
Address = Vector Table[Vector Number];
```

Finding the Exception Handler or ISR to Run

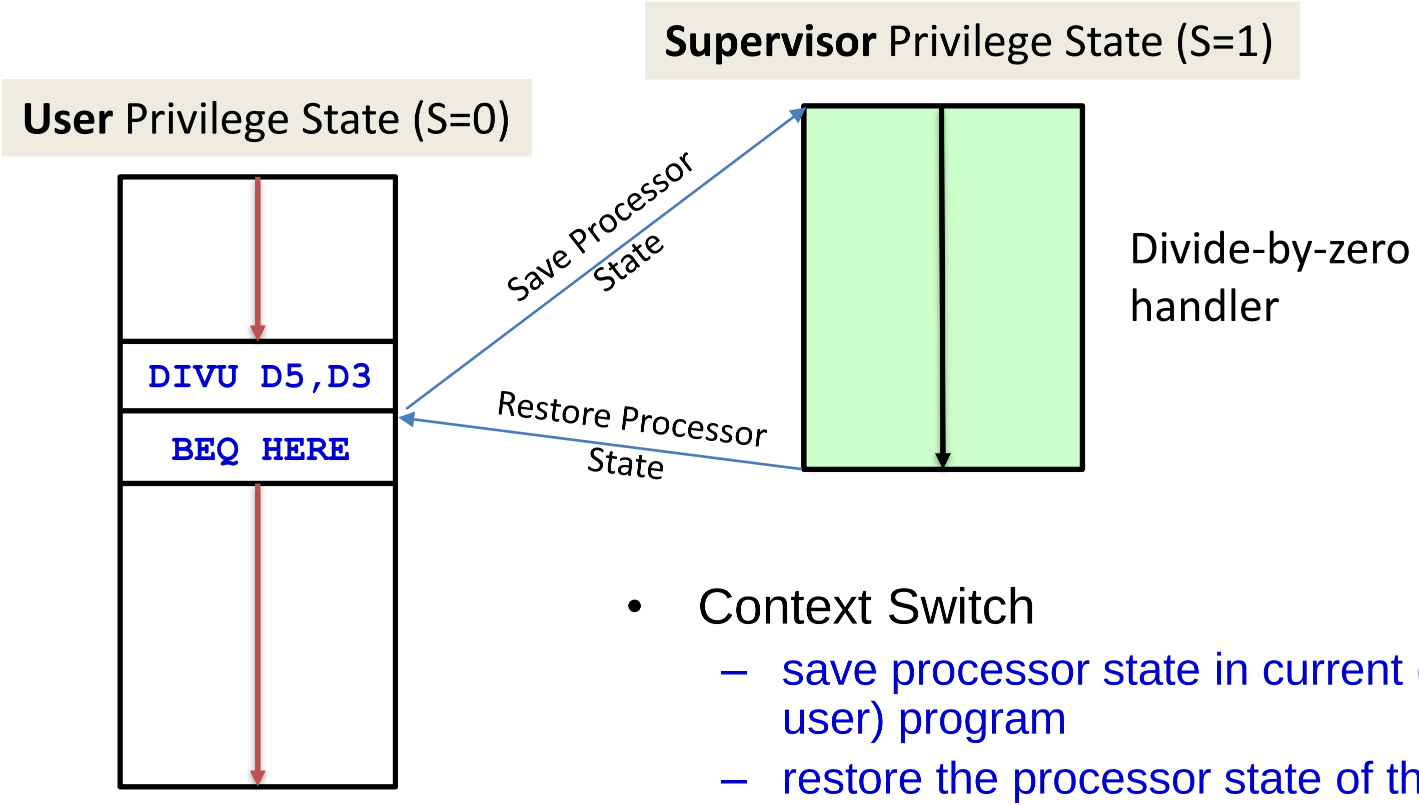


```
Address = Vector Table[Vector Number];
```

Location of Vector Table

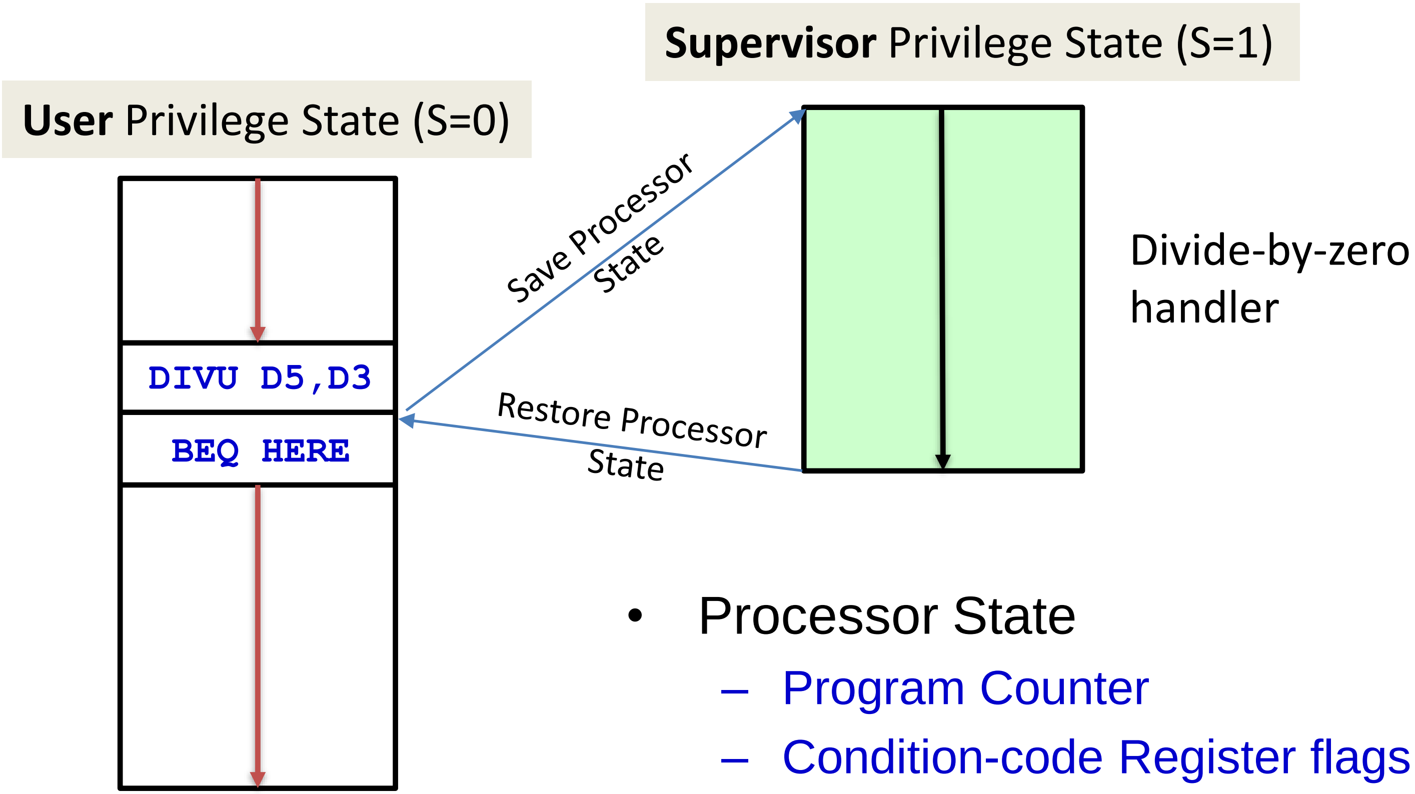


Context Switch



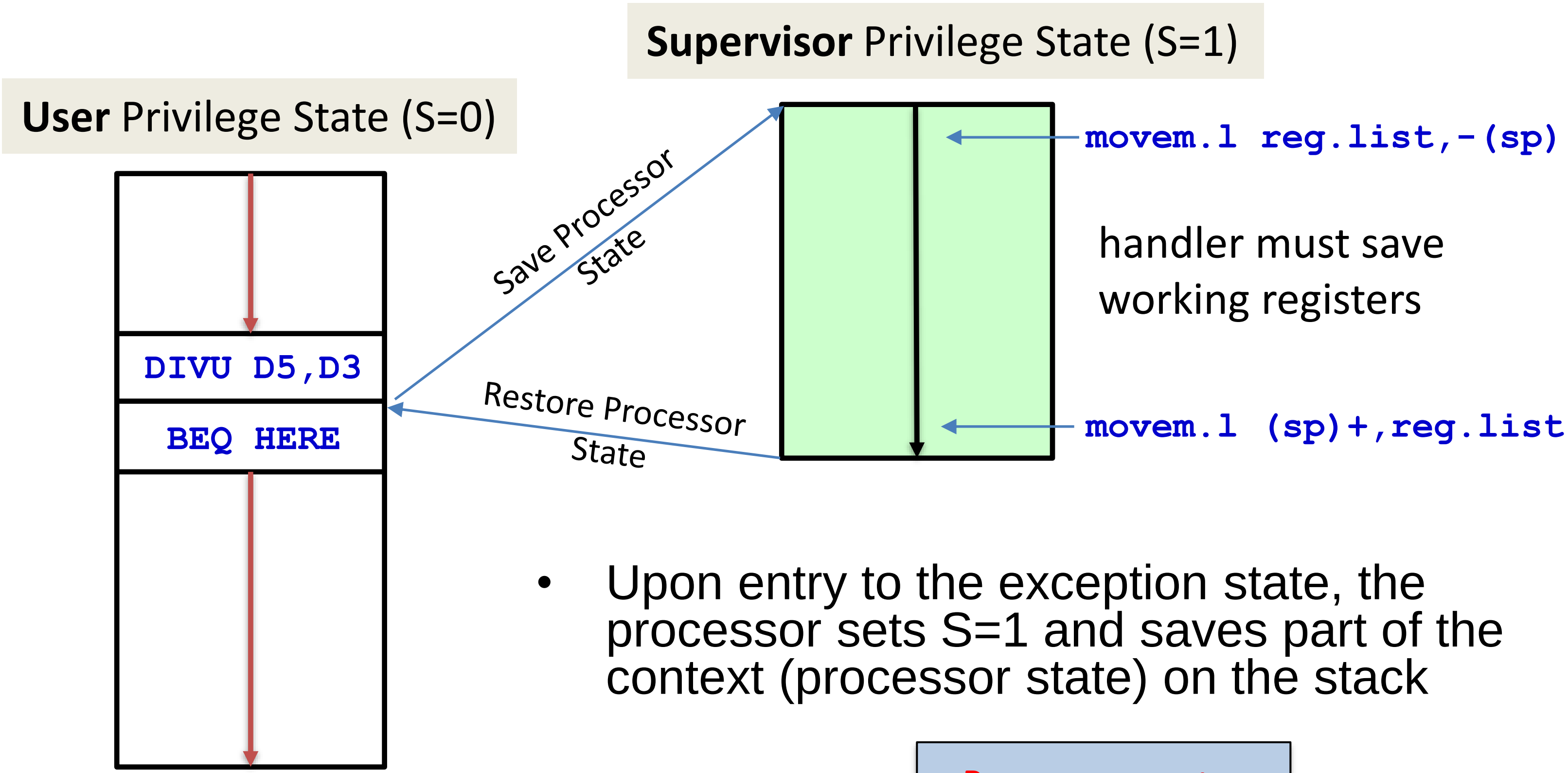
- Context Switch
  - save processor state in current (e.g., user) program
  - restore the processor state of the preempted (user) program
  - pass control to the restored (user) program

Processor State



- Processor State
  - Program Counter
  - Condition-code Register flags
  - Privilege state
  - Working registers

Saving Processor State



Supervisor Privilege State (S=1)

User Privilege State (S=0)

`DIVU D5,D3`

`BEQ HERE`

Save Processor State

Restore Processor State

`movem.l reg.list,-(sp)`

handler must save working registers

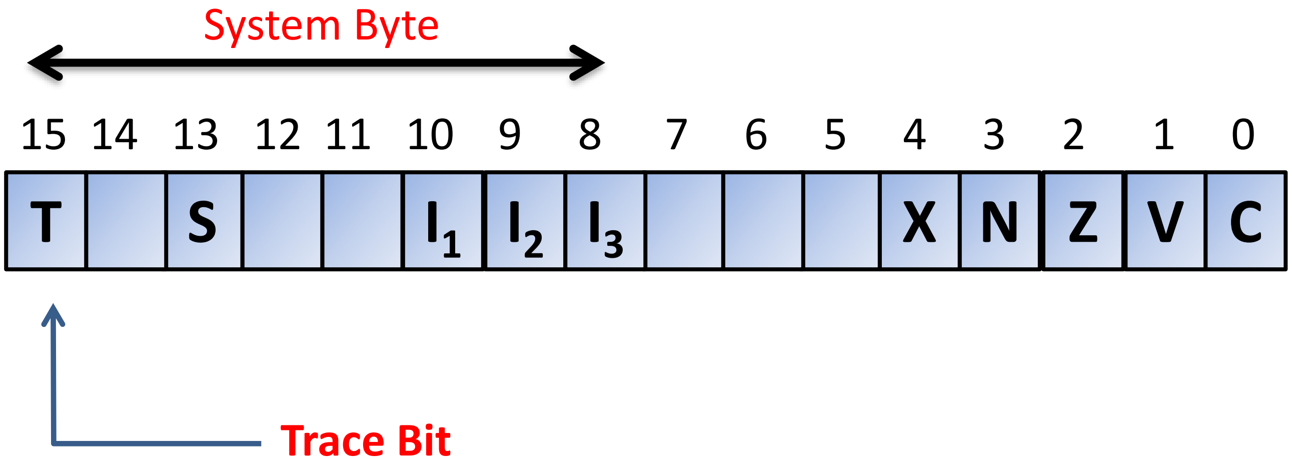
`movem.l (sp)+,reg.list`

- Upon entry to the exception state, the processor sets S=1 and saves part of the context (processor state) on the stack

Program counter

Status register

The Status Register and System Byte



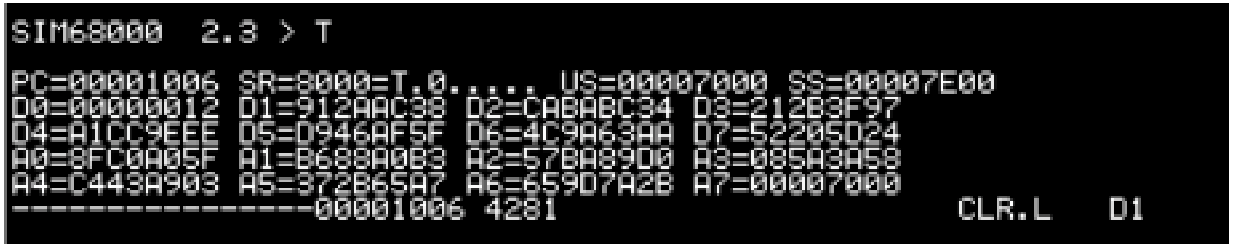
System Byte

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
T		S			I_1	I_2	I_3				X	N	Z	V	C

Trace Bit

Sets a post-instruction debug routine into action

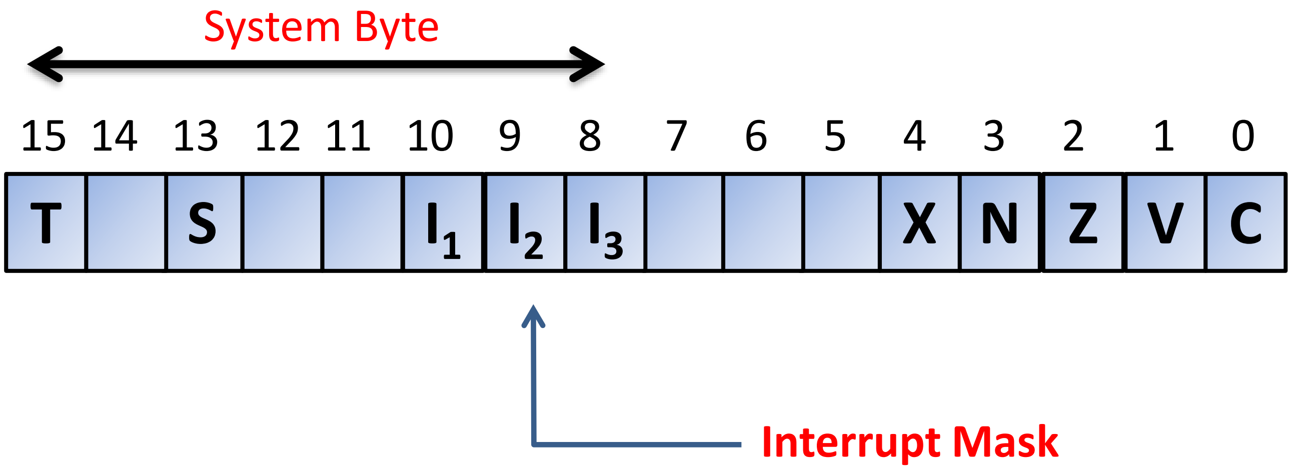
- T=0 (Trace off)
- T=1 (Trace on)



```
SIM68000  2.3 > T

PC=00001006 SR=8000=T.0..... US=00007000 SS=00007E00
D0=00000012 D1=912AAC38 D2=CABABC34 D3=212B3F97
D4=A1CC9EEE D5=D946AF5F D6=4C9A63AA D7=52205D24
A0=8FC0A05F A1=B688A0B3 A2=57BA89D0 A3=085A3A58
A4=C443A903 A5=372B65A7 A6=659D7A2B A7=00007000
----------------00001006 4281                     CLR.L    D1
```

The Status Register and System Byte



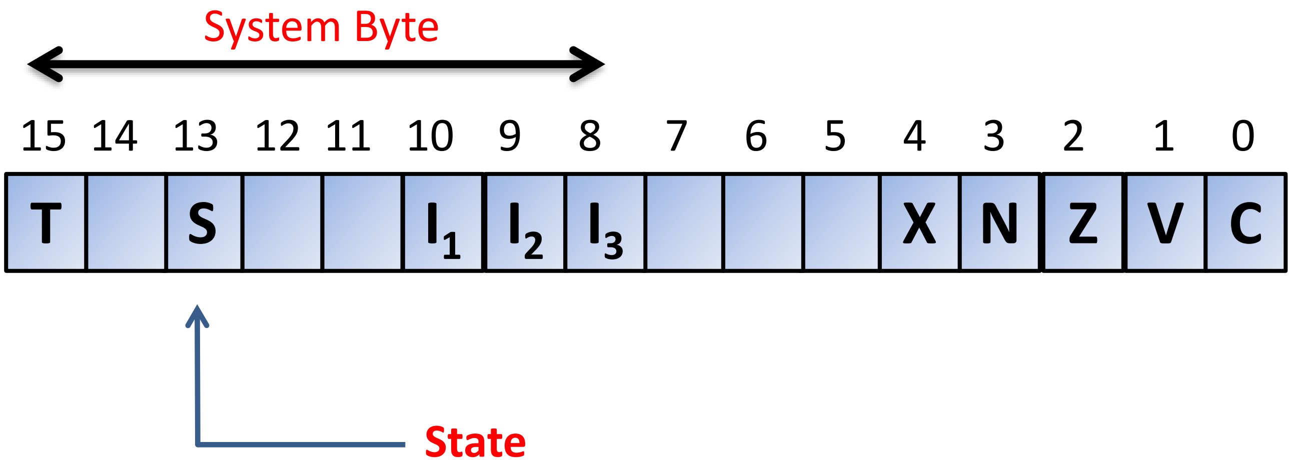
System Byte

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
T		S			I_1	I_2	I_3				X	N	Z	V	C

Interrupt Mask

Sets the lowest priority interrupt that the processor will respond do

The Status Register



System Byte

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
T		S			I_1	I_2	I_3				X	N	Z	V	C

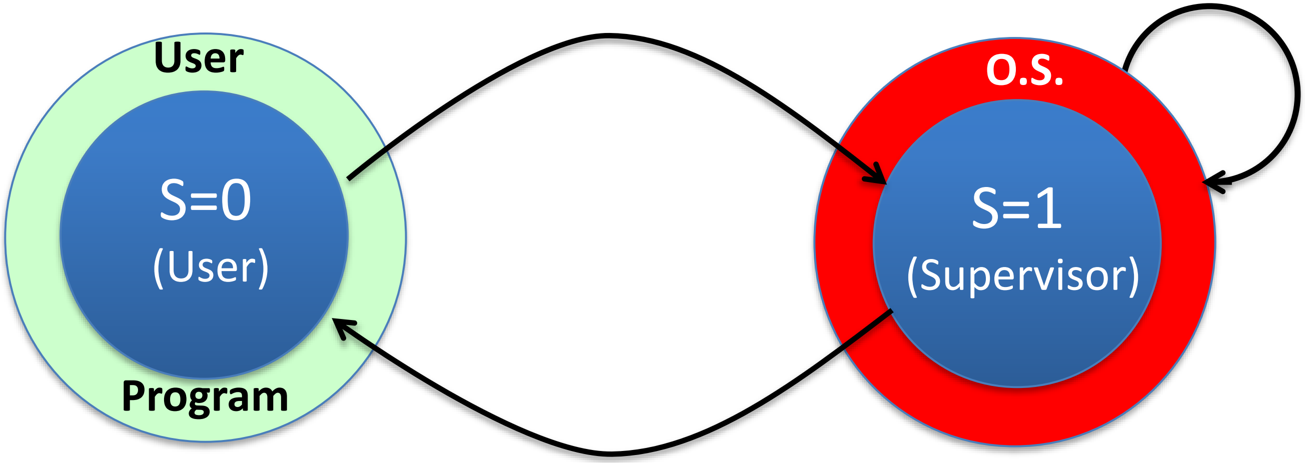
State

Allows a privileged mode of execution which is essential for multi-user environments

- S=0 (User)
  - Application or user program
- S=1 (Supervisor)
  - Operating System or monitor

Switching Between Supervisor and User States

Transitions may occur only through interrupt/exception processing



Any exception

Transition may occur via three privileged instructions:

- `ANDI #data,SR`
- `EORI #data,SR`
- `RTE`

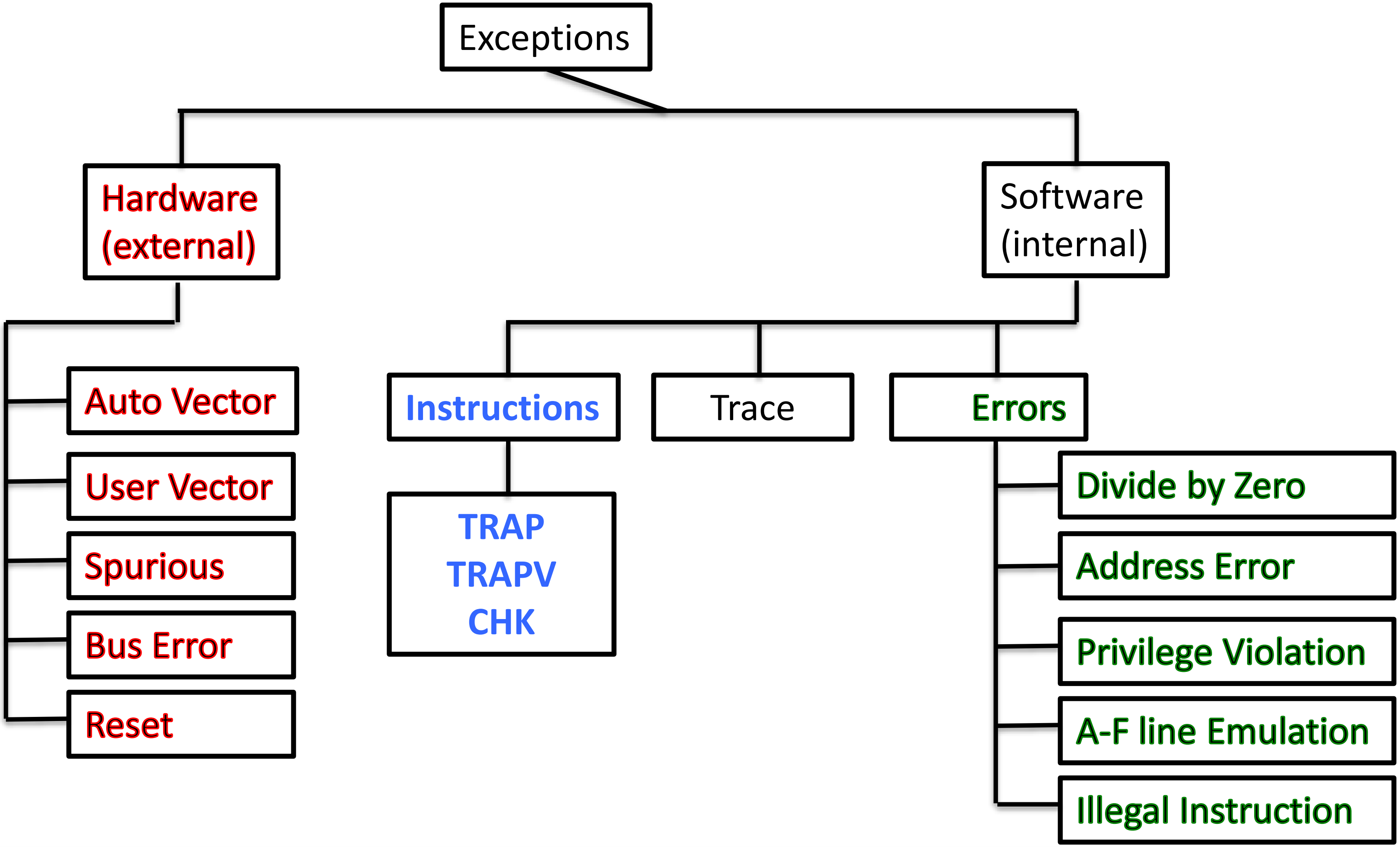
Difference Between Supervisor and User States

ISA Component	User (S=0)	Supervisor (S=1)
Active Stack Pointer	USP/A7	SSP/A7
Other Stacks	A0-A6	A0-A6, USP
Status Register Access	Read: Entire SR Write: CCR bits only	Read: Entire SR Write: Entire SR
Available Instructions	All Except: – `ANDI #data,SR` – `EORI #data,SR` – `ORI  #data,SR` – `MOVE <ea>,SR` – `MOVE USP,An` – `MOVE An,USP` – `RTE` – `RESET` – `STOP`	All

Problem

- Show how the `EORI` instruction can be used to change the state of the processor from supervisor to user, without affecting any other bits in the status register.

68000 Exception Taxonomy



Exceptions

Hardware (external)

Auto Vector

User Vector

Spurious

Bus Error

Reset

Software (internal)

Instructions

TRAP
TRAPV
CHK

Trace

Errors

Divide by Zero

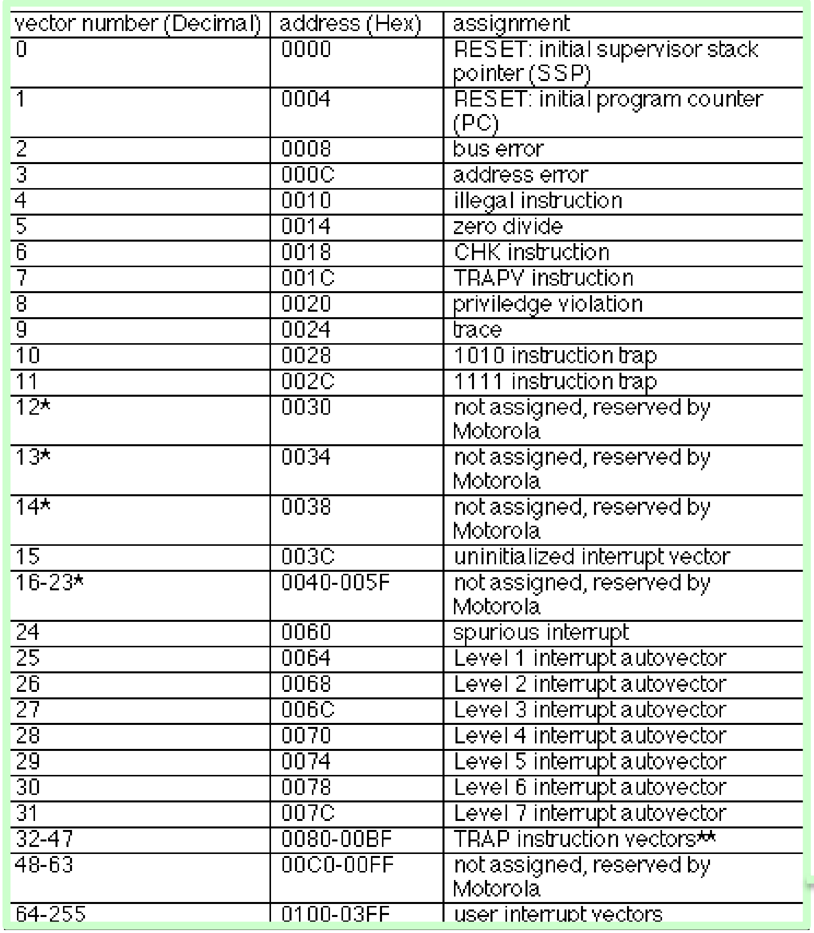
Address Error

Privilege Violation

A-F line Emulation

Illegal Instruction

68000 Vector Table, Vector Numbers and Addresses



vector number (Decimal)	address (Hex)	assignment
0	0000	RESET: initial supervisor stack pointer (SSP)
1	0004	RESET: initial program counter (PC)
2	0008	bus error
3	000C	address error
4	0010	illegal instruction
5	0014	zero divide
6	0018	CHK instruction
7	001C	TRAPV instruction
8	0020	priviledge violation
9	0024	trace
10	0028	1010 instruction trap
11	002C	1111 instruction trap
12*	0030	not assigned, reserved by Motorola
13*	0034	not assigned, reserved by Motorola
14*	0038	not assigned, reserved by Motorola
15	003C	uninitialized interrupt vector
16-23*	0040-005F	not assigned, reserved by Motorola
24	0060	spurious interrupt
25	0064	Level 1 interrupt autovector
26	0068	Level 2 interrupt autovector
27	006C	Level 3 interrupt autovector
28	0070	Level 4 interrupt autovector
29	0074	Level 5 interrupt autovector
30	0078	Level 6 interrupt autovector
31	007C	Level 7 interrupt autovector
32-47	0080-00BF	TRAP instruction vectors**
48-63	00C0-00FF	not assigned, reserved by Motorola
64-255	0100-03FF	user interrupt vectors

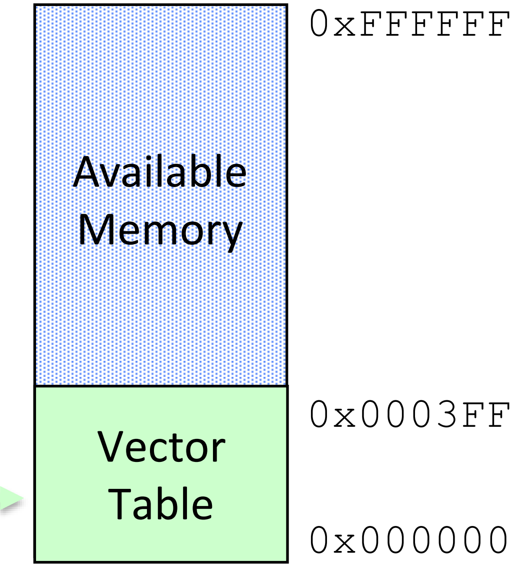
NOTES:
* No peripheral devices should be assigned these numbers
** TRAP #N uses vector number 32+N

Vector Number

- Unique number associated with each exception

Vector address

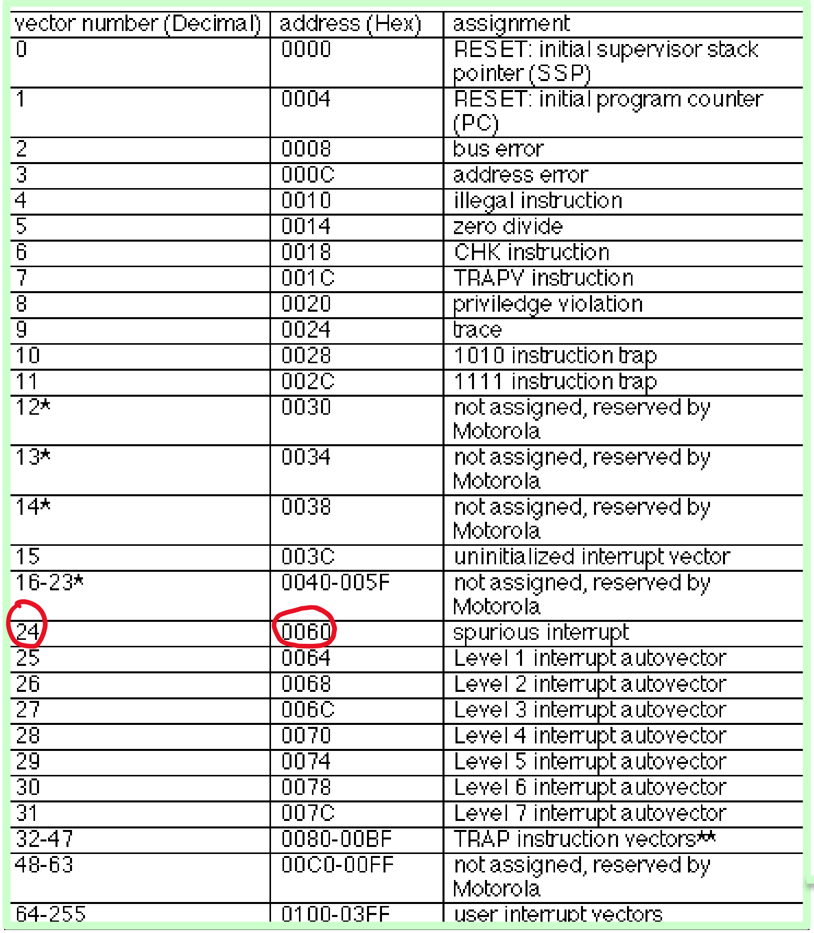
- Address of the EH/ISR



Problem

- What is the vector address of the exception assigned vector number 24? What is the exception?

68000 Vector Table, Vector Numbers and Addresses



vector number (Decimal)	address (Hex)	assignment
0	0000	RESET: initial supervisor stack pointer (SSP)
1	0004	RESET: initial program counter (PC)
2	0008	bus error
3	000C	address error
4	0010	illegal instruction
5	0014	zero divide
6	0018	CHK instruction
7	001C	TRAPV instruction
8	0020	priviledge violation
9	0024	trace
10	0028	1010 instruction trap
11	002C	1111 instruction trap
12*	0030	not assigned, reserved by Motorola
13*	0034	not assigned, reserved by Motorola
14*	0038	not assigned, reserved by Motorola
15	003C	uninitialized interrupt vector
16-23*	0040-005F	not assigned, reserved by Motorola
24	0060	spurious interrupt
25	0064	Level 1 interrupt autovector
26	0068	Level 2 interrupt autovector
27	006C	Level 3 interrupt autovector
28	0070	Level 4 interrupt autovector
29	0074	Level 5 interrupt autovector
30	0078	Level 6 interrupt autovector
31	007C	Level 7 interrupt autovector
32-47	0080-00BF	TRAP instruction vectors**
48-63	00C0-00FF	not assigned, reserved by Motorola
64-255	0100-03FF	user interrupt vectors

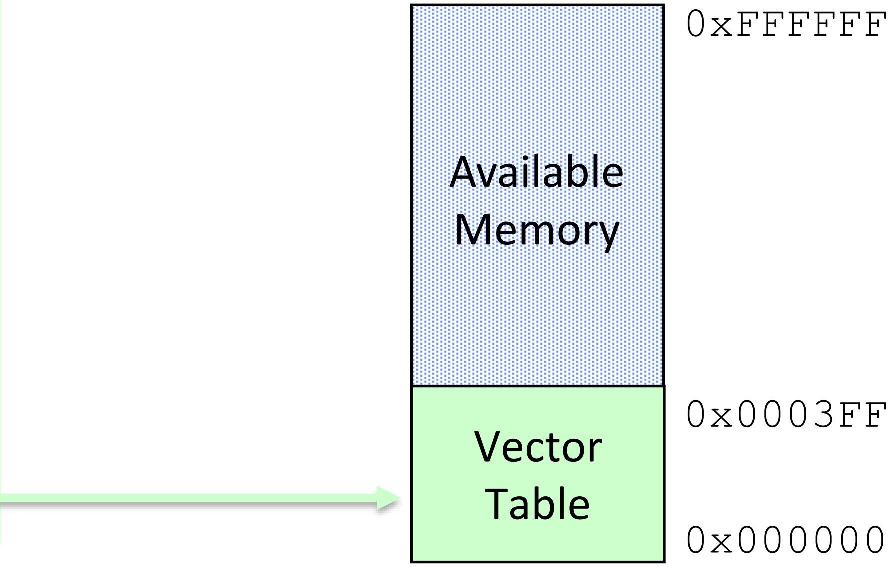
NOTES:
* No peripheral devices should be assigned these numbers
** TRAP #N uses vector number 32+N

Vector Number

- Unique number associated with each exception

Vector address

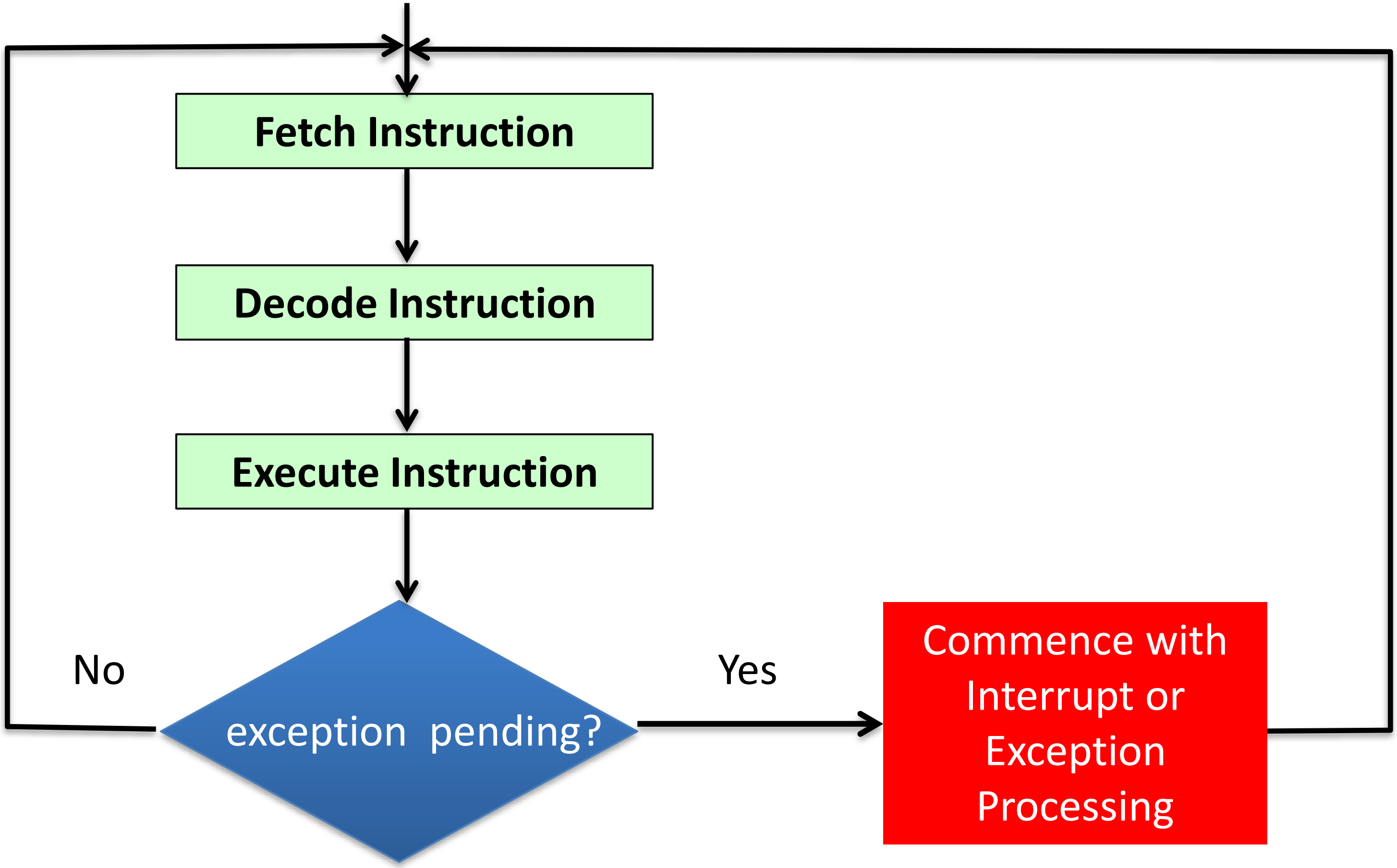
- Address of the EH/ISR



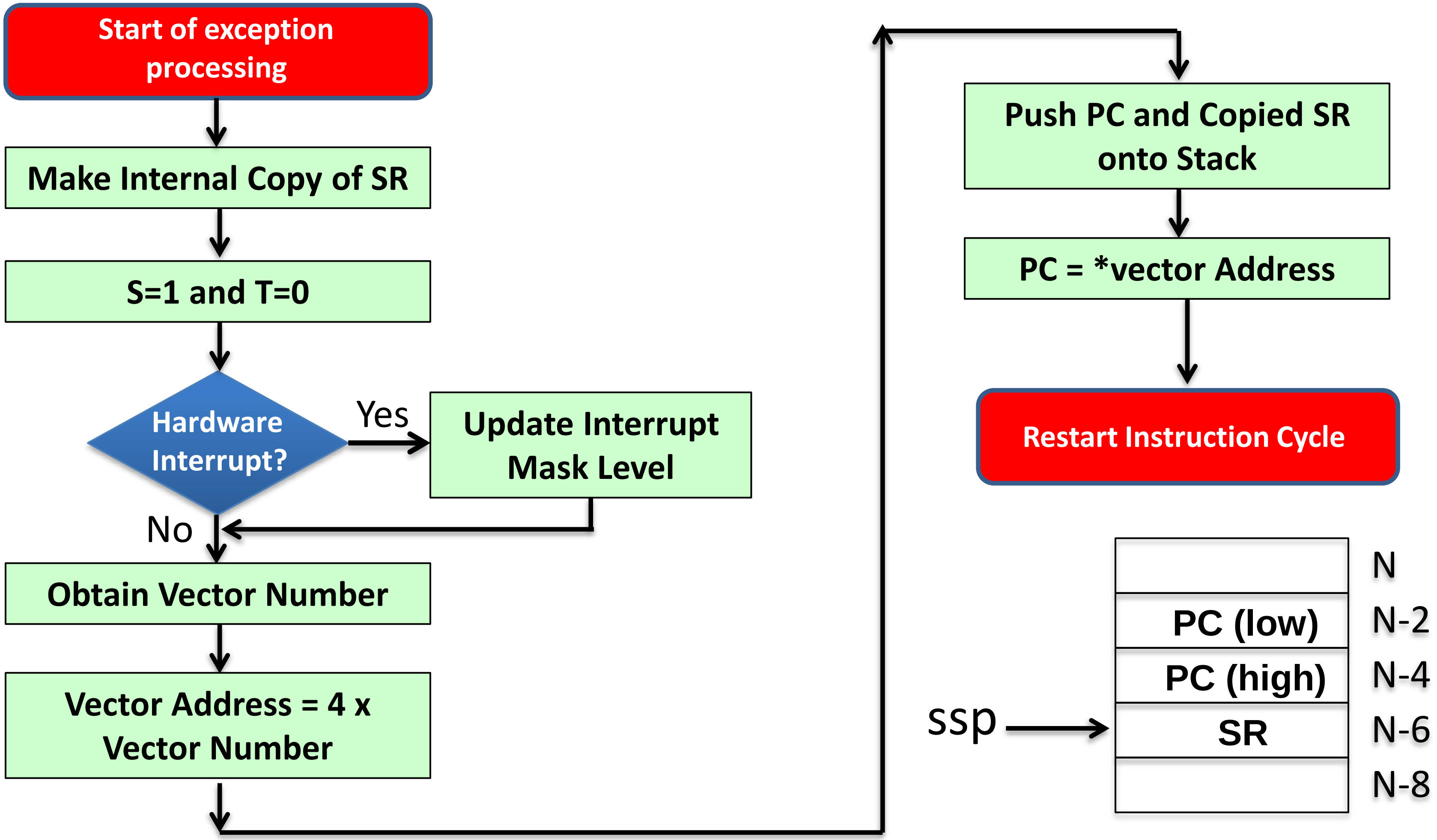
Problem

- A systems' programmer writes code for a divide-by-zero exception handler and places the routine at memory address $FFA238. Write an instruction to update the vector table accordingly.

When are exceptions Processed?



How are Exceptions Processed?



Start of exception processing

Make Internal Copy of SR

S=1 and T=0

Hardware Interrupt?

Yes

Update Interrupt Mask Level

No

Obtain Vector Number

Vector Address = 4 x Vector Number

Push PC and Copied SR onto Stack

PC = *vector Address

Restart Instruction Cycle

		N
	PC (low)	N-2
	PC (high)	N-4
ssp →	SR	N-6
		N-8

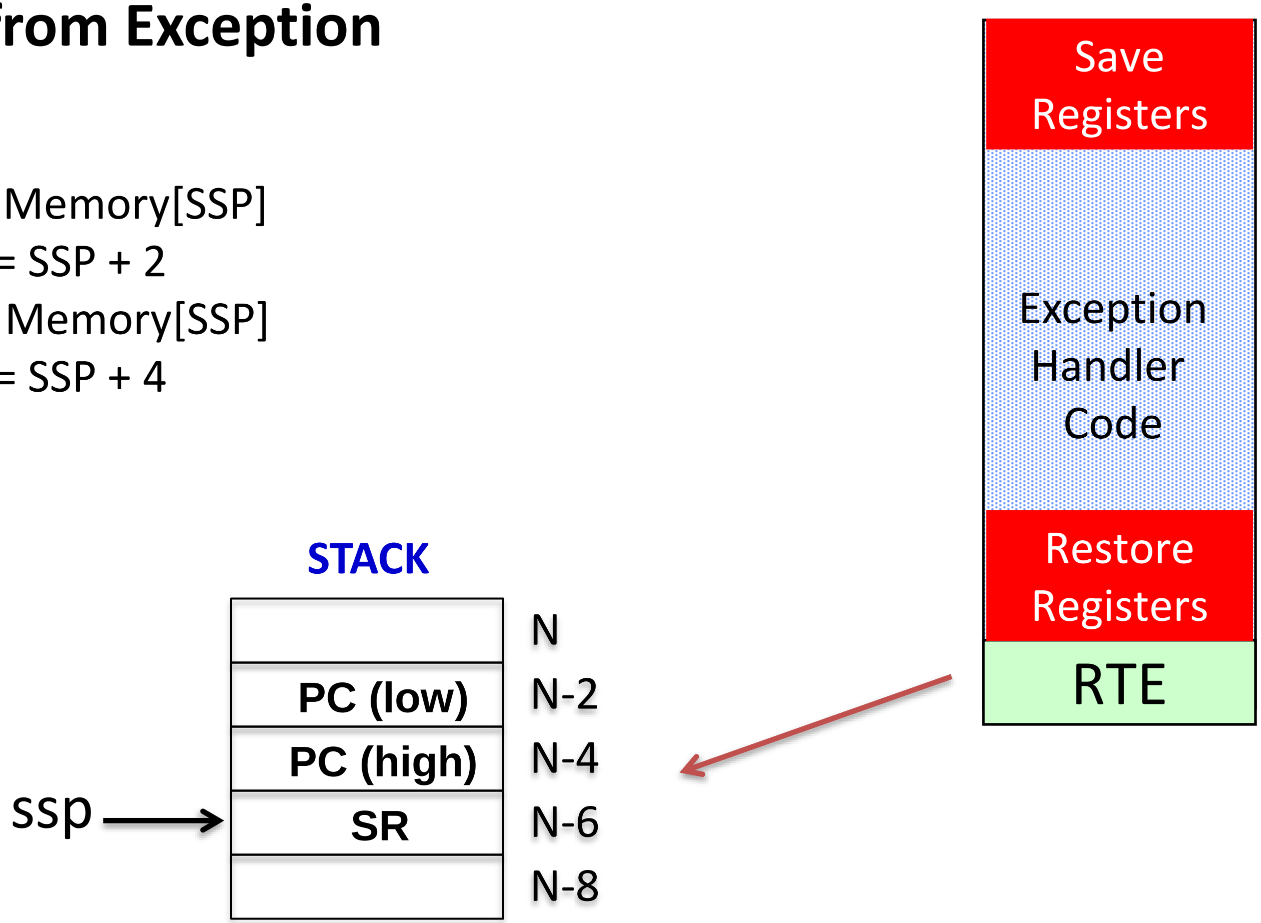
How to Return to Pending Program?

RTE Return from Exception

Syntax: `RTE`

Operation:

```
SR = Memory[SSP]
SSP = SSP + 2
PC = Memory[SSP]
SSP = SSP + 4
```



Save Registers

Exception Handler Code

Restore Registers

RTE

STACK

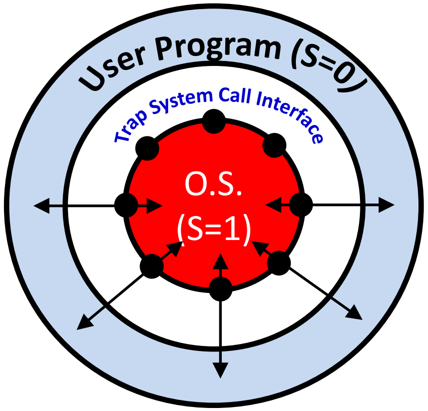
	N
PC (low)	N-2
PC (high)	N-4
SR	N-6
	N-8

Case Study: TRAP Instruction

TRAP - used to perform operating system calls

Syntax: `TRAP #N (0-15)`

Operation:
SSP = SSP - 4
Memory[SSP] = PC
SSP = SSP -2
Memory[ssp] = SR
PC = Vector Table [N + 32]



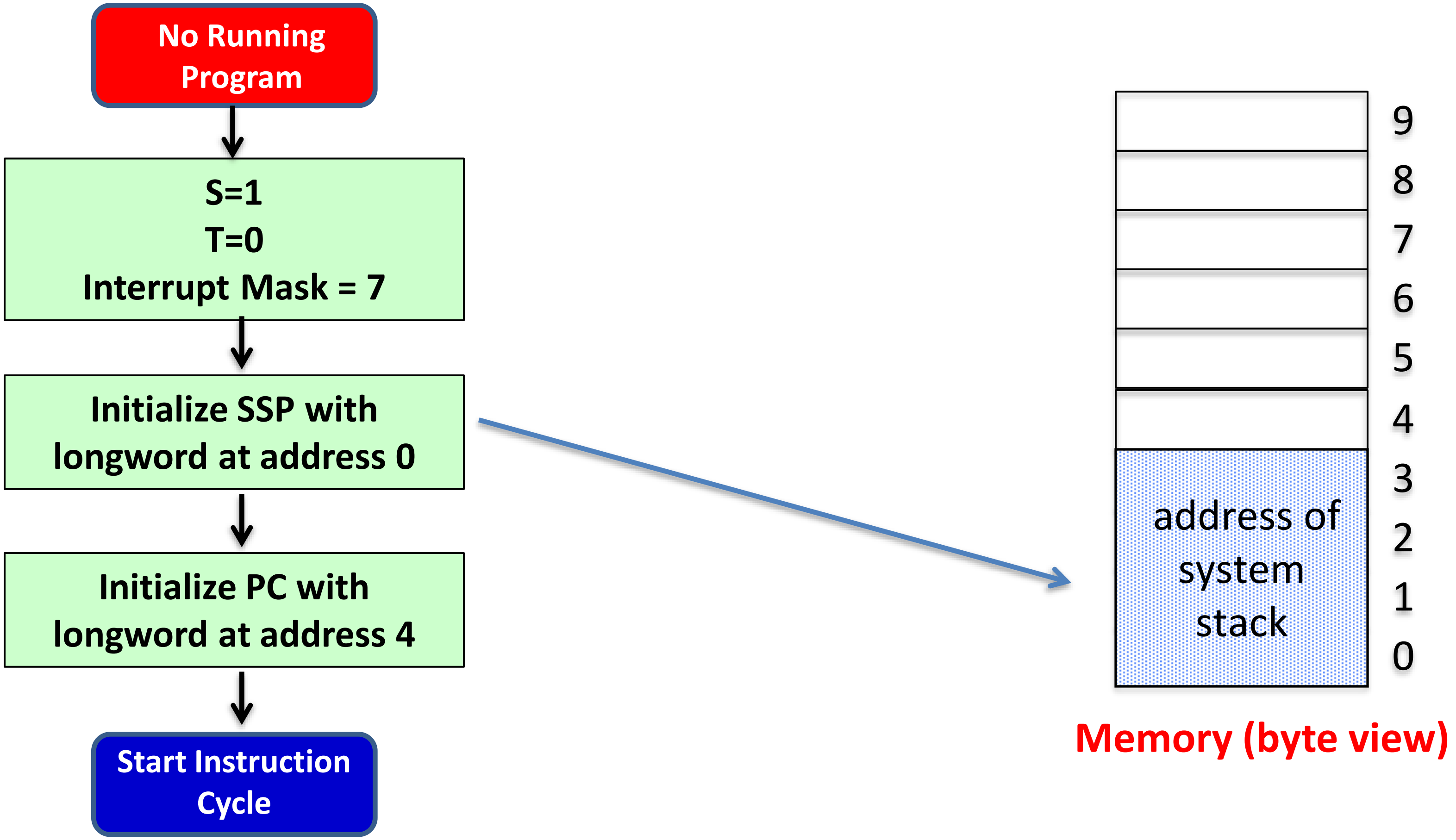
Note:
- possible to pass a integer to the trap handler in a data register
- Within the trap handler, the integer is used as an index into a jump table that points to the desired routine

Case Study: TRAP #15

- Illustrate the steps performed by both the normal and exception-processing states when TRAP #15 (task number 6) is used to display the ASCII character ‘X’ on the terminal.

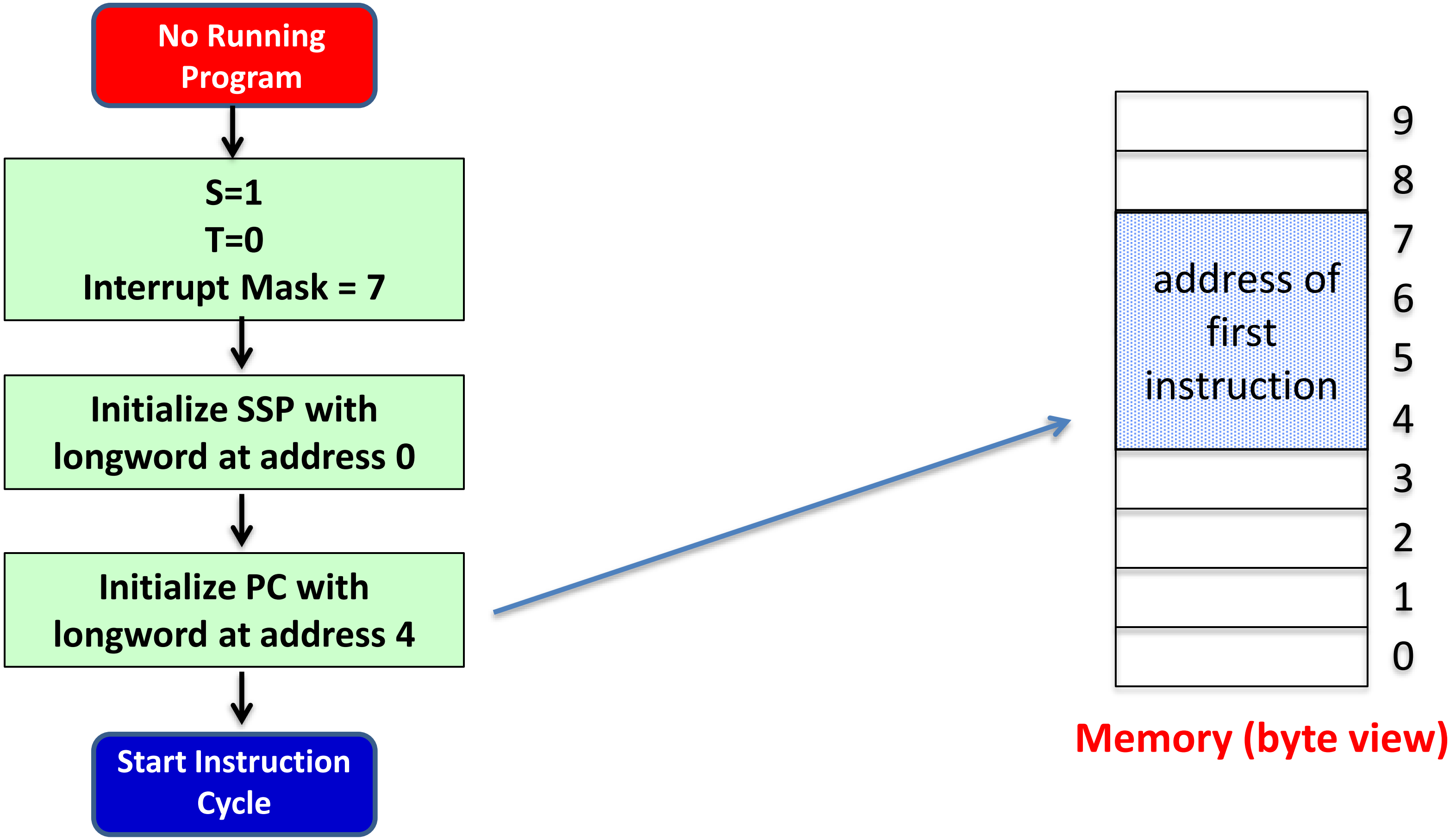
Reset Exception

- The reset exception occurs with the 68000 is first powered up



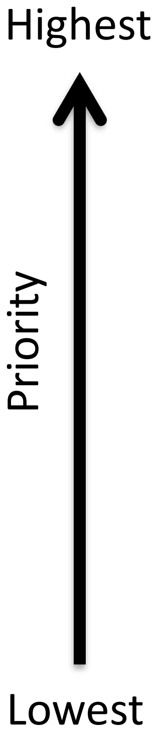
Reset Exception

- The reset exception occurs with the 68000 is first powered up



No Running Program
S=1
T=0
Interrupt Mask = 7
Initialize SSP with longword at address 0
Initialize PC with longword at address 4
Start Instruction Cycle
address of first instruction
9
8
7
6
5
4
3
2
1
0
Memory (byte view)

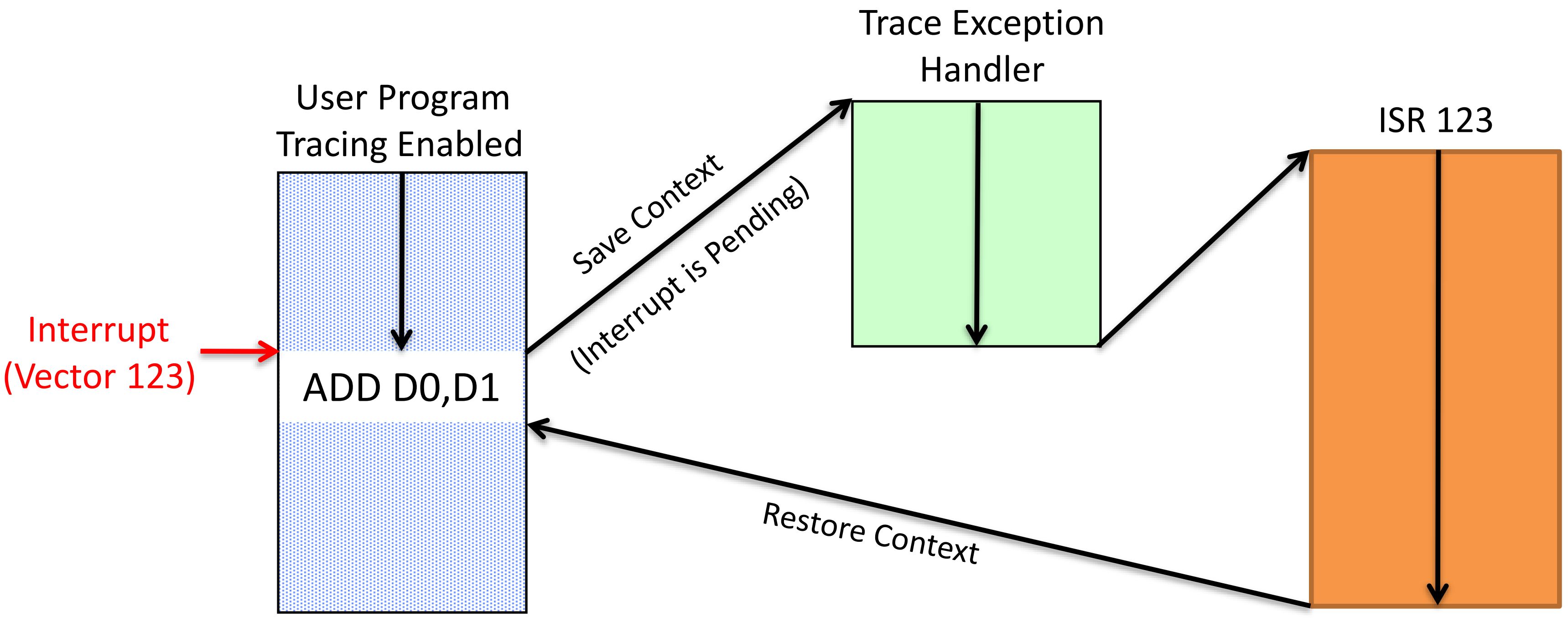
Exception Grouping and Priority



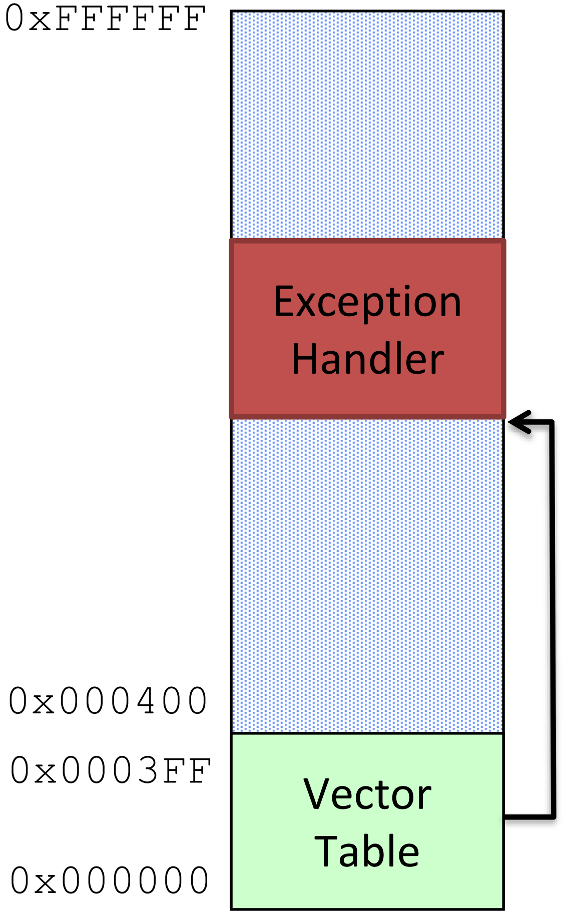
Group	Exception	Processing
0	Reset Address Error Bus Error	Processing begins within 2 clock cycles
1	Trace Interrupt Illegal Instruction Privilege Violation	Processing begins before the next instruction
2	TRAP, TRAPV, CHK Zero Divide	Processing is started by normal instruction execution

Problem

- What happens if the processor receives an interrupt request during the execution of a (user) instruction when tracing is on (i.e., T=1)?



Writing Your Own TRAP Handler



Step 1: Writing the exception handler and place in memory (remember to save/restore working registers)

Step 2: Load the address of the handler into the vector table

Problem

- Write a TRAP #5 exception handler to clear the bytes in memory between address 1 and address 2 inclusive. Assume address 1 and 2 are passed to the handler in address registers A1 and A2, respectively, and address 1 is lower than address 2.

Summary

- An exception is an abrupt change in control flow in response to some change in processor state
  - Internal, Synchronous
    - Called “exceptions”
      - Requests for I/O
      - Integer division by 0
      - Attempts to access protected memory
  - External, Asynchronous
    - Called “interrupts”
      - User presses a key on keyboard
      - Disk controller finishes reading data
- Each exception has a unique vector number and vector address
- Exceptions are similar to function calls
  - Control transfers from original code to other code
  - Other code executes
  - Control returns to original code

Summary

- Exceptions are different from function calls
  - Processor pushes additional state onto the stack
    - Status register and Program Counter
  - Processor pushes state onto the Operation Systems' stack, not user's stack
  - Handler (or service routine) runs in privileged state, not user state
    - Handler can execute all instructions and access all ISA resources including memory
  - Control usually returns to next instruction in suspended code
  - Control sometimes does not return at all!
- Multiple exceptions are processed according to priority
- Reset exception is unique and is used to powerup a system