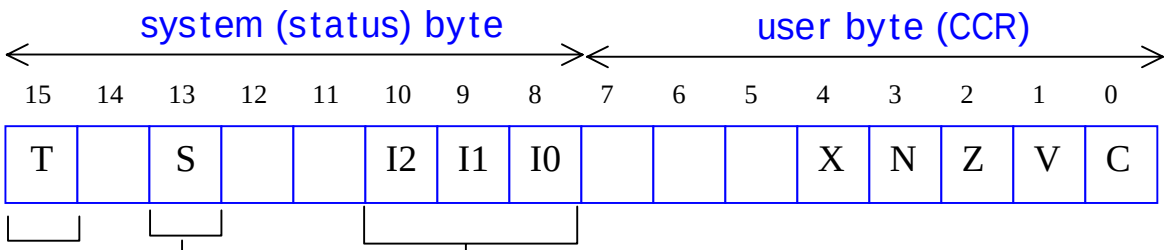
The Status Register



Interrupt Level

Sets the lowest priority interrupt that the CPU will respond to

State

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

S = 1 – Supervisor Mode (operating system/monitor)

S = 0 – User Mode (user programs)

Trace

Sets a post-instruction routine into action

T = 0 – Trace off

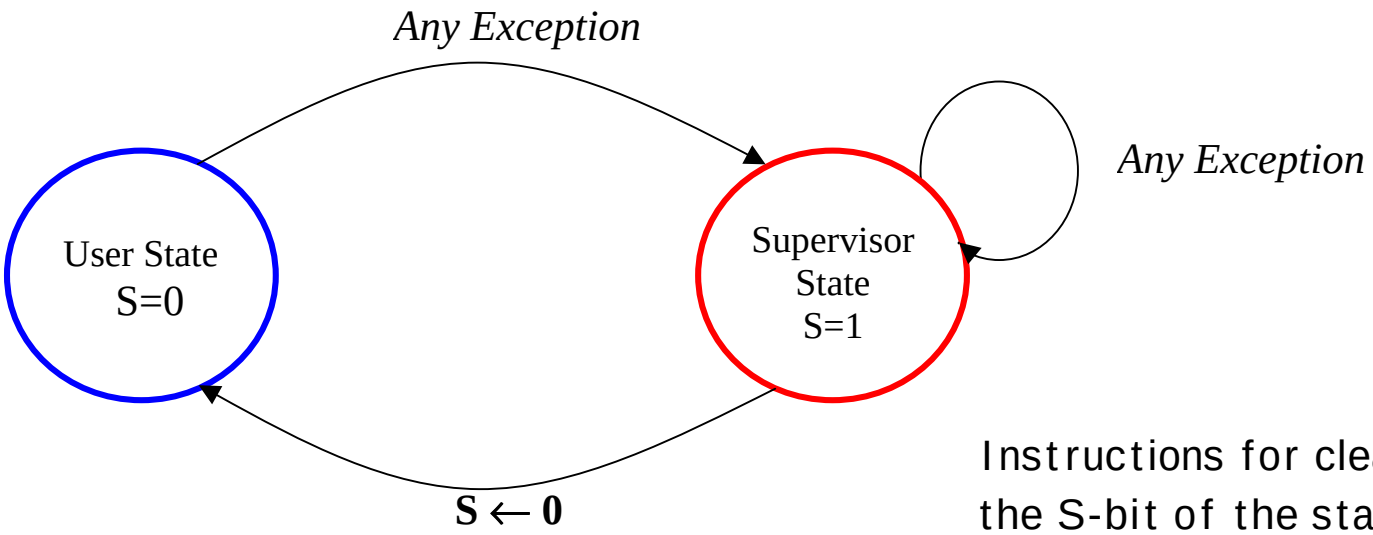
T = 1 – Trace on

Supervisor Mode Versus User Mode

	User Mode	Supervisor Mode
Active SP	USP	SSP
Other Stacks	A0-A6	USP,A0-A6
Status Register Access		
Read:	Entire SR	Entire SR
Write:	CCR bits only	Entire SR

Available Instructions:	All except
	`AND #data,SR`
	`EOR #data,SR`
	`MOVE <ea>,SR`
	`OR #data,SR`
	`MOVE USP,An`
	`MOVE An,USP`
	`RESET`
	`STOP`
	`RTE`
		All

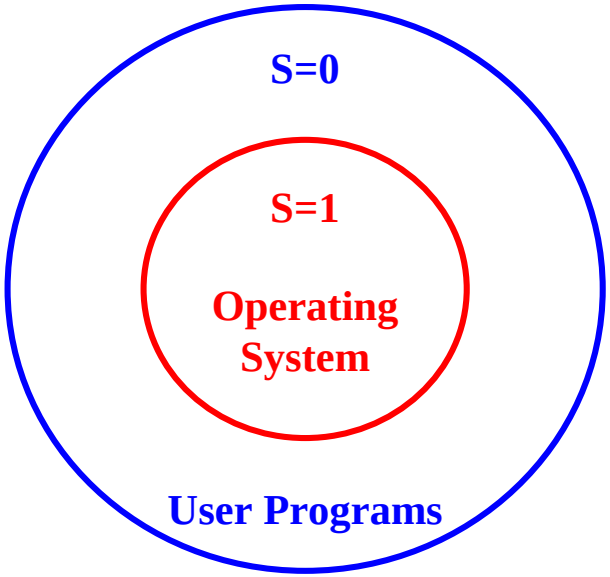
How Does the 68000 Change States?



Instructions for clearing or setting the S-bit of the status register:

```
MOVE.W <EA>,SR
ANDI.W  #N,SR
EORI.W  #N,SR
RTE
```

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
T		S			I2	I1	I0				X	N	Z	V	C



1. When the O/S is first loaded:

- The State bit (S) is set to 1
- The O/S is then loaded by the bootstrap program in the BIOS

2. When a user program is loaded:

- The O/S copies the program into an available part of memory
- Initializes the user stack pointer
- For the sake of "protecting" other users, the State bit (S) is set to 0
- The O/S then jumps to the user program's entry point

Note:

- If the user program attempts to modify the (S) bit, an exception (privilege violation) will occur and control will be returned to the O/S
- The user program can request services from the O/S using exceptions (i.e., TRAP instructions)

Exceptions and Exception Handlers

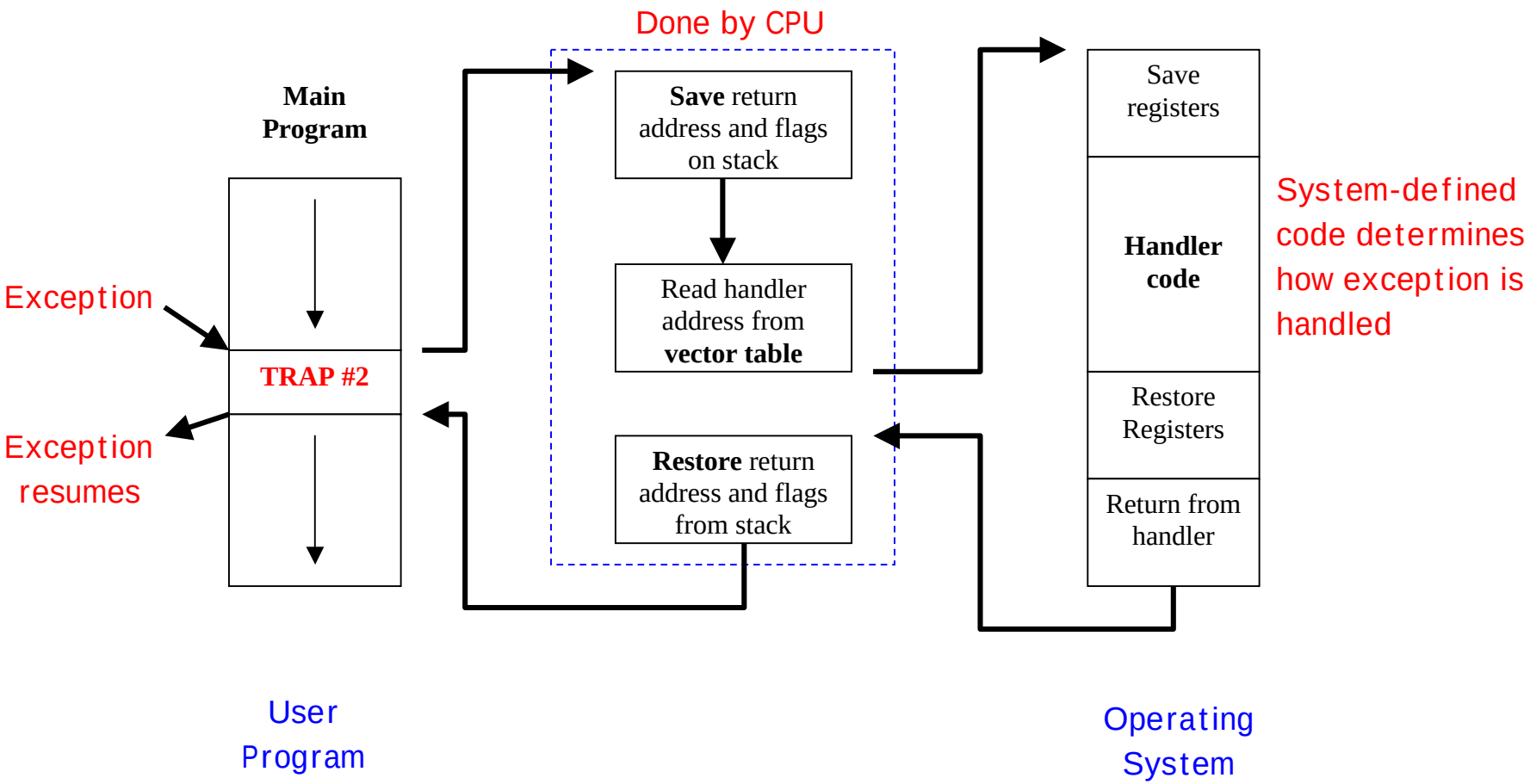
Exceptions are really *calls* to the operating system.

- These calls may be made *explicitly* or *automatically*
- Exceptions are divided into two classes: *internal* and *external*
- Each type of exception has its own *exception handler*.

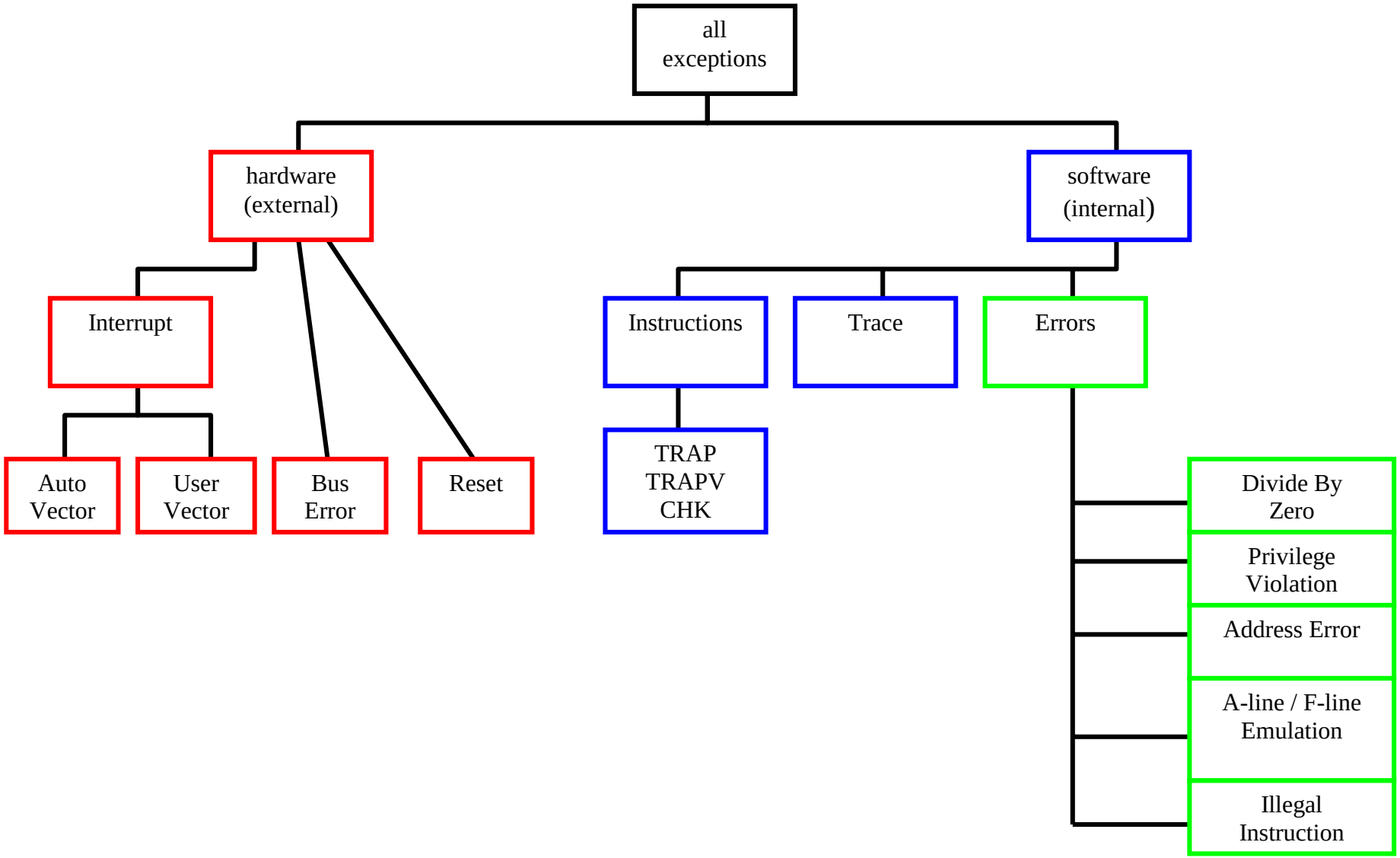
An *exception handler* is a piece of code that takes care of processing the exception once encountered.

- Exception handlers are normally part of the *operating system*
- The systems programmer puts the *address of the exception handler in a table* (called the vector table)

Typical Exception Processing During Normal Program Execution



The 68000's Exceptions



Vector Table

Vector Address:

Memory location that contains the 32-bit address of exception-handling routine.

Vector Number:

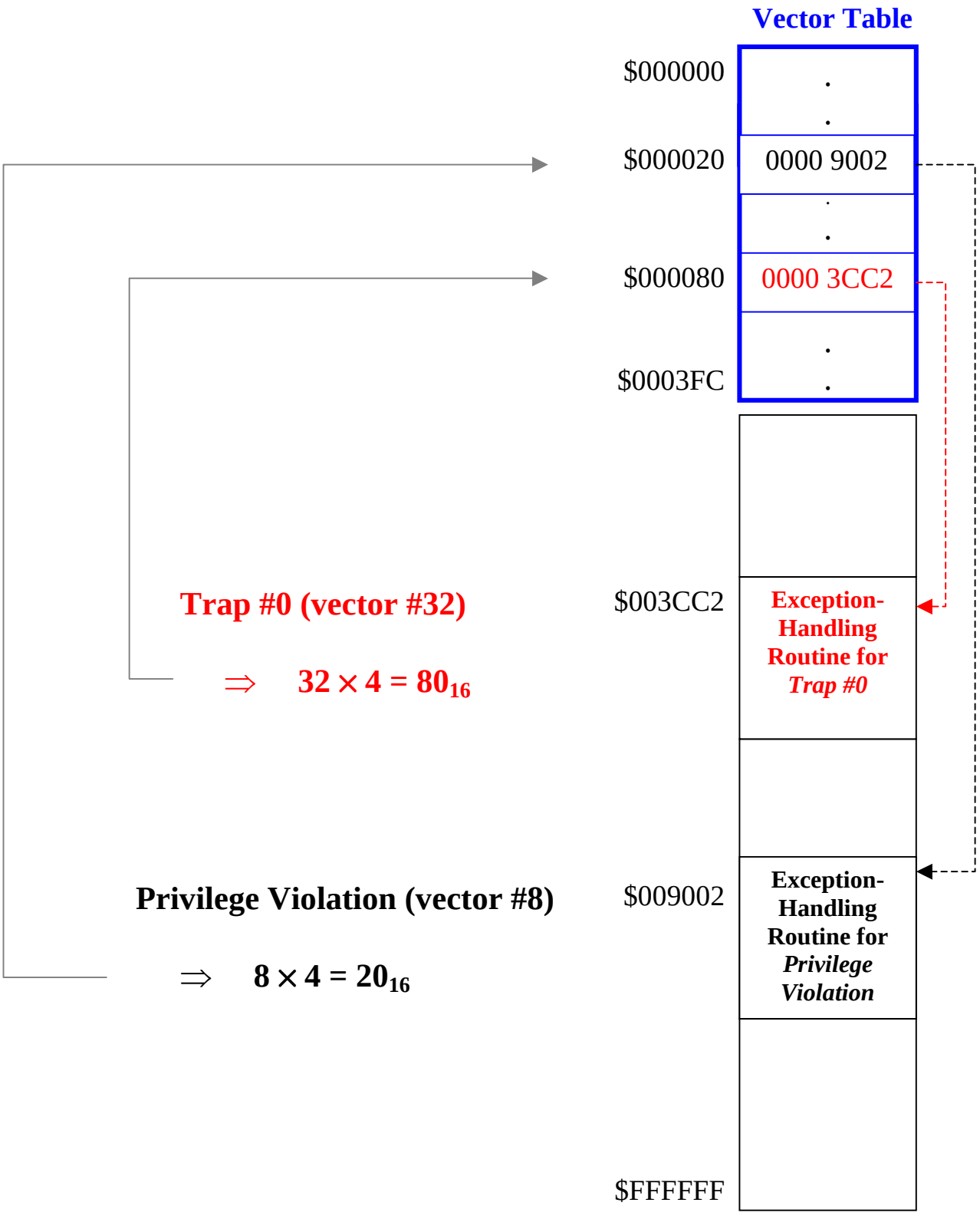
A value when multiplied by four, gives the (vector) address of an exception vector.

e.g., Zero Divide

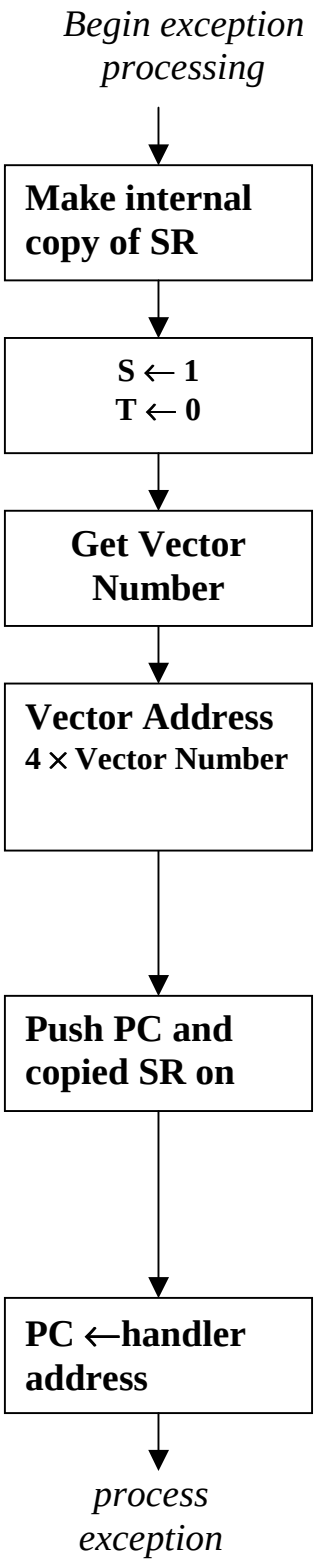
$5 \times 4 = 14_{16}$

Memory (Vector) Address		Vector Number
0	Reset	0
8	Bus Error	2
C	Address Error	3
10	Illegal Instruction	4
14	Zero Divide	5
18	Check Instruction	6
1C	TRAPV instruction	7
20	Privilege violation	8
24	TRACE	9
28	LINE A Emulator	10
2C	LINE F Emulator	11
30	Unassigned Reserved	12
3C	Uninitialized Interrupt	15
40	Unassigned Reserved	16
60	Spurious Interrupt	24
64	Level 1 Autovector	25
68	Level 2 Autovector	26
6C	Level 3 Autovector	27
70	Level 4 Autovector	28
74	Level 5 Autovector	29
78	Level 6 Autovector	30
7C	Level 7 Autovector	31
80	16 Trap Instruction Vectors	32
C0	Unassigned Reserved	48
100	192 User Interrupt Vectors	64
3FF		255

Example


Vector Table
$000000
$000020
0000 9002
$000080
0000 3CC2
$0003FC
$003CC2
Exception-Handling Routine for Trap #0
$009002
Exception-Handling Routine for Privilege Violation
$FFFFFF
Trap #0 (vector #32)
⇒ 32 × 4 = 80₁₆
Privilege Violation (vector #8)
⇒ 8 × 4 = 20₁₆

Exception Processing Sequence – Calling the Exception Handler



temp_register ← SR save status register in temporary location

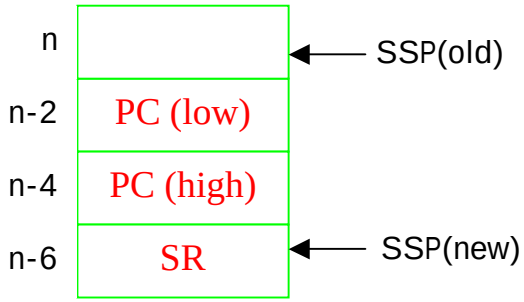
S ← 1 force supervisor state

T ← 0 turn trace off

Generated internally by 68000 (except for user interrupts)

Address ← 4 × Vector Number read table to get

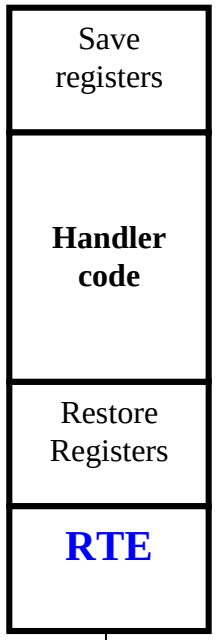
Handler ← Memory[Address] exception handler address



PC ← Handler run exception handler

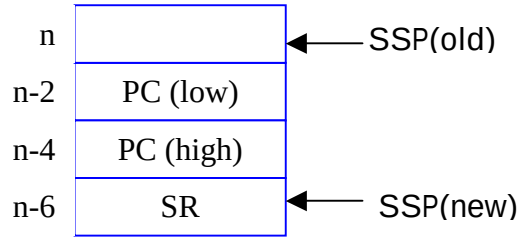
mechanism for processing interrupts will be shown later

Exception Processing Sequence – Part 2



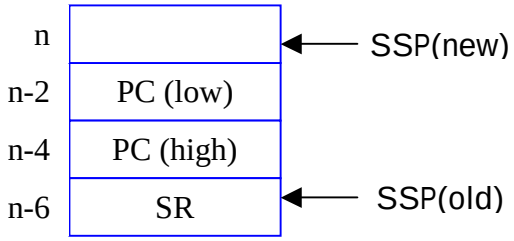
continue processing normally

RTE restores old PC and SR



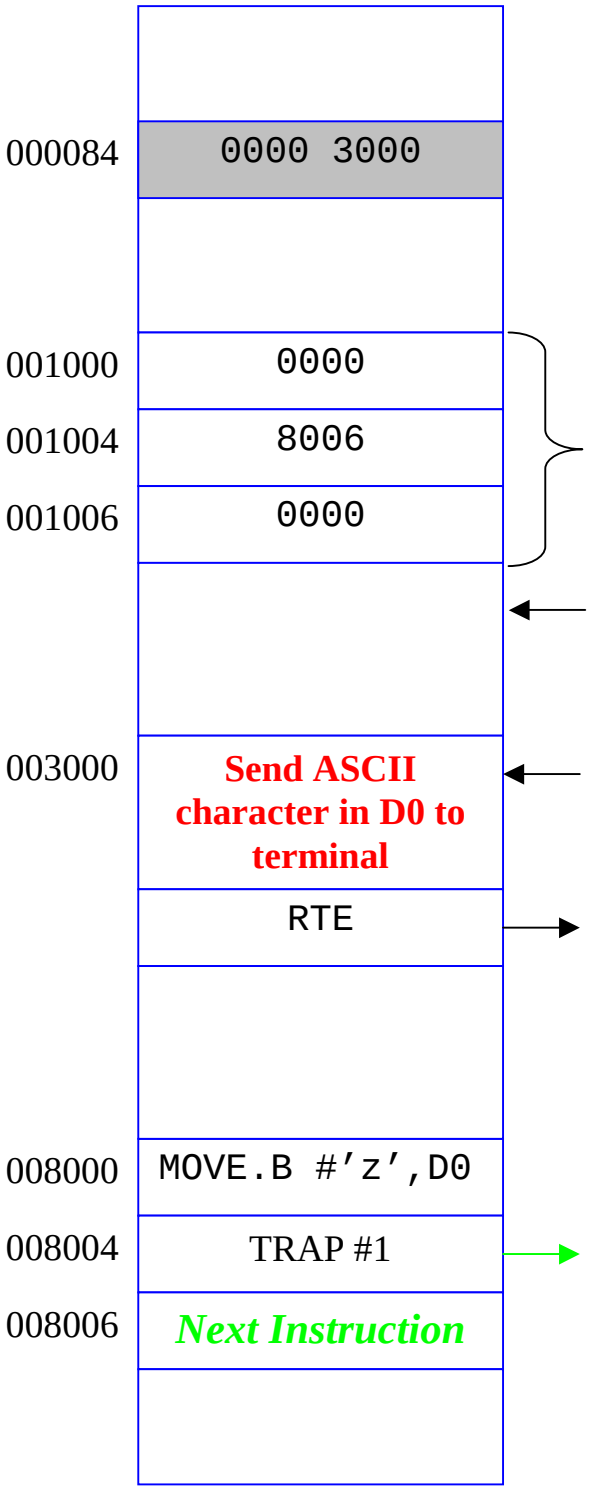
before

SR ← Memory[SSP] restore old SR
SSP ← SSP + 2
PC ← Memory[SSP] restore old PC
SSP ← SSP + 4



after

An Example Using Trap #1



3. Trap #1 (vector 33)

   Vector address = $4 \times 33 = 84_{16}$

4. Push PC and old SR on stack

SSP (old)

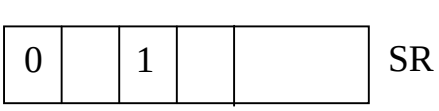
5. Execute exception handler: PC=$003000

6. Exit exception handler – restore old value of PC and SR.

   PC = $008006, SR=$0000

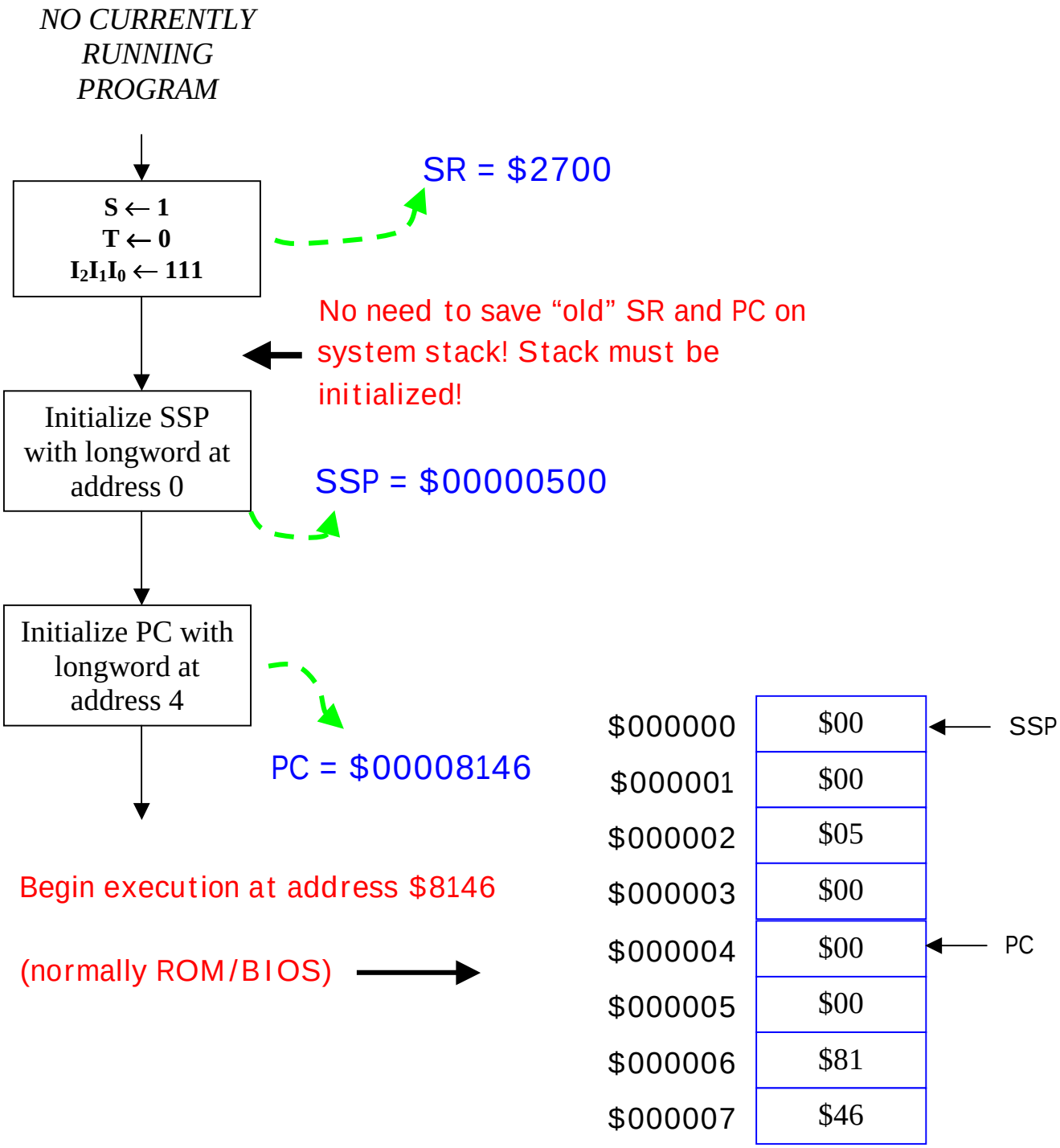
Exception Occurs

1. Internal copy of SR made (SR=$0000)
2. New copy of SR made (SR=$2000)



Reset Exception

The reset exception occurs when the 68000 is first powered up.

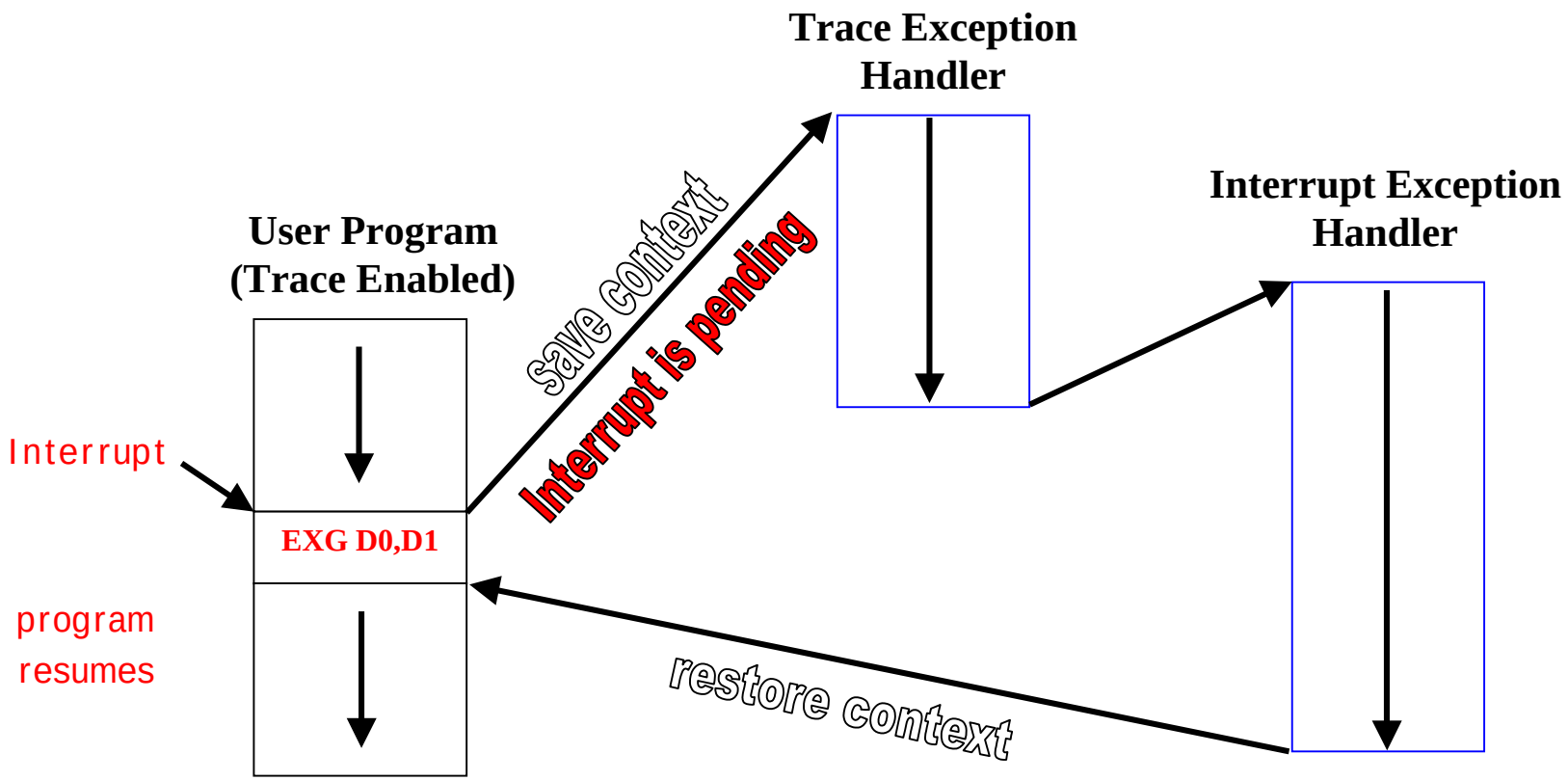


Handling Multiple Exceptions

The 68000's exceptions are divided into three groups according to their priority and characteristics.

Group	Exception	Time at which processing begins
0	Reset Bus error Address Error	Within two clock cycles
1	Trace Interrupt Illegal Privilege	Before next instruction
2	TRAP TRAPV CHK Zero Divide	Started by normal execution of instruction

Example


Trace Exception Handler
Interrupt Exception Handler
User Program (Trace Enabled)
save context
Interrupt is pending
Interrupt
EXG D0,D1
program resumes
restore context

Trap Instructions on the 68KMB

All operating systems provide a powerful, complex library of routines that programmers can use in their own programs. In order to call an operating system routine you must use one of the sixteen TRAP instructions supported by the 68000.

Syntax: `TRAP #N`