Exceptions

- also alter the order of execution of instructions

Two types:

1) Internal exceptions – caused by software when anomalous situations occur (ex. Divide by zero, trap, etc.)
2) External exceptions – caused by hardware when external hardware component requires attention of processor (ex. Reset, interrupt caused by depressing key on keyboard)

When exception occurs, a piece of code is executed that indicates the appropriate action desired. Normal program execution is suspended and program jumps to exception handling subroutine. If there is a reason to abort, program execution is terminated (unrecoverable exceptions, such as divide by zero), otherwise, it returns to the appropriate instruction of the program (recoverable exceptions, such as interrupts).

An exception can occur either during execution of an instruction or immediately after its completion.

How does the processor convey the nature of exception?

- each exception type is mapped to a unique number ranging from 0 to 255 known as Exception Vector Number
    - divide by zero – exception number is 5
    - address error – exception number is 3

Exception vector number serves as an index to an array of 256 longwords – the exception vector table – loaded in the first 1K of main memory.

So for vector number *v*, the address of the exception handler is given by the longword contents of memory address *v*4* (Table 9.4).

Two groups of exception – Table 9.2

Before handler is invoked (like branch and execute) some housekeeping is performed to make sure that the status register SR contents are same upon return, and the return address must be saved. The following are pushed onto the Supervisor Stack.

- push return address (either current address or next instruction address)
- then push SR contents

Exceptions are handled by the monitor (operating system). Therefore the SUPERVISOR bit is SET to 1 and the processor operates in supervisor mode. The TRACE bit is also CLEARED. (Figure 9.2)

When exception occurs the processor invokes the handler, but to return from the handler, we need an appropriate return instruction – return from exception (*rte*).

- pops SR first
- then pops PC (return address)

Note – rts, rtr instructions pop values off the user stack and not from the system stack.

Internal Exceptions

1) Trace Exception ($v = 9$)
   When Trace Bit of SR is set, a trace exception is generated and handler provides maximum amount of information to user. The handler may run in an interactive mode.

```
dump_reg        equ      $6000
trace_vector    equ      $24
...             ...      .....

...             ...      .....
program         move.l   #trace_handler, trace_vector
...             ...      ......

trace_handler   jsr dump-reg
                rte

                end
```

2) Divide by Zero *(v = 5)*

- non recoverable (displays error messages and program terminates)
- Fig 9.4

3) Privileged Instruction Exception *(v = 8)*

- when one attempts to execute a privileged instruction while the processor is in user mode
- ex using rte instruction in user mode

4) Unimplemented (*v = 10, 11)* and Illegal Instruction Exceptions

- Invalid instructions, the binary strings of which do not correspond to any of the valid instructions.
- To distinguish invalid instructions –
    - Unimplemented instructions
        - No instruction starts with first nibble (4 bits) as $A – called as line $A emulation exception
        - No instruction starts with first nibble (4 bits) as $F – called as line $F emulation exception
        - These can be used to implement Floating point instructions
        - See figure 9.5
    - Illegal instructions
        - No instruction starts with first word as either $4AFA, $4AFB, $4AFC
        - Used for breakpoints – see Figure 9.6
        - Breakpoints are recorded in a breakpoint table and $4AFA is inserted in place of valid code. When instruction is executed, an exception occurs. Now it is the responsibility of the handler, to replace the invalid word with original word, execute the instruction, corrupt it again, and return to user's program

5) Trap exception

- software generated

a) Trap on overflow (trapv)

- to catch overflows in arithmetic operations
- trapv instruction is included in assembly code after each operation that may result in overflow
- if overflow bit V = 1, then initiate exception *(v = 7)*

b) Traps (*v = 32 to 47*)

- To communicate with Input/Outputs using 16 different traps
- trap #0 to trap #15

    vector number in table = vector$_4$ + 32   where 0 $\leq$vector$_4$ $\leq$15

6) Check instruction *(v = 6)*

- an instruction that initiates an exception whenever an integer is not within a specified range
- used primarily to detect the index of an array which is out of bounds

example:

```
if register d0 = 000000F2

chk #5,d0     ; initiates an exception since F2 > 5
chk #405,d0  ; does not initiate exception since F2 < 405
```

7) Address Error *(v = 3)*

- whenever a longword or a word is attempted to be accessed at a byte boundary.

External Exceptions

- result of activity outside the processor
    - power on/off (using reset exception)
    - when another device wishes to communicate using interrupts

1) Reset (*v = 0, 1*)

- signaled by asserting of external reset pin
    - all registers have undetermined values

- priority level of interrupt set to 7
- For the system to get started, 2 longwords of ROM plus enough ROM to contain initialization routines is required – used to booting
- Reset procedure
    - Current instruction immediately terminated
    - Set supervisor status bit, Reset trace bit, Priority mask bit set to 7
    - System Stack Pointer loaded with longword at $000000
    - Program counter loaded with longword at $000004
    - Begin program execution

2) Interrupts (*v = 25 to 31*)

- Need for interrupts? - allows asynchronous (cannot predict actions) interaction with external environment
    - Another way is polling
- asserting of external interrupt line (3 bits)
- 7 levels (001 to 111), 000 default (normal operation)
    - if interrupt level is greater than the current level, it is acknowledged, otherwise interrupt is ignored until mask register is below external level
- interrupts may be nested
- Service an interrupt procedure
    - Complete current instruction
    - Acknowledge external interrupt
    - Determine interrupt vector number (24 + number on interrupt line)
    - Determine address ((25 to 31) * 4)
    - Exception processing
        - SR -> WR
        - S bit set
        - T bit reset
        - Interrupt mask set to current level
        - PC -> on supervisor stack
        - WR -> on supervisor stack
        - Address -> PC

        - Interrupt handler should have "rte" to retrieve original SR which has prioi interrupt level

3) Bus Error - attempt to access out of range memory address

TABLE 9.3 Two exception groups

Where to return?		
Group	Exception type	Action
1	Illegal instruction, unimplemented instruction, privilege violation, address error, bus error	Save the PC of the current instruction
2	Trace, `trap`, `trapv`, `chk`, zero divide, interrupt	Save the PC of the "next" instruction

TABLE 9.4 Exception vector table

Vector number	Address	Assignment
0	$000000	Reset: Initial SSP
1	$000004	Reset: Initial PC
2	$000008	Bus error
3	$00000C	Address error
4	$000010	Illegal instruction
5	$000014	Division by zero
6	$000018	CHK instruction
7	$00001C	TRAPV instruction
8	$000020	Privilege violation
9	$000024	Trace
10	$000028	Line $A emulator
11	$00002C	Line $F emulator
12 13 14	$000030 $000034 $000038	Reserved by Motorola
15	$00003C	Uninitialized interrupt vector
16 : 23	$000040 ...... $00005F	Reserved by Motorola
24	$000060	Spurious interrupt
25	$000064	Level 1 Interrupt autovector
26	$000068	Level 2 Interrupt autovector
27	$00006C	Level 3 Interrupt autovector
28	$000070	Level 4 Interrupt autovector
29	$000074	Level 5 Interrupt autovector
30	$000078	Level 6 Interrupt autovector
31	$00007C	Level 7 Interrupt autovector
32 : 47	$000080 ...... $0000BF	TRAP Instruction vectors
48 : 63	$0000C0 ...... $0000FF	Reserved by Motorola
64 : 255	$000100 ...... $0003FF	User interrupt vectors

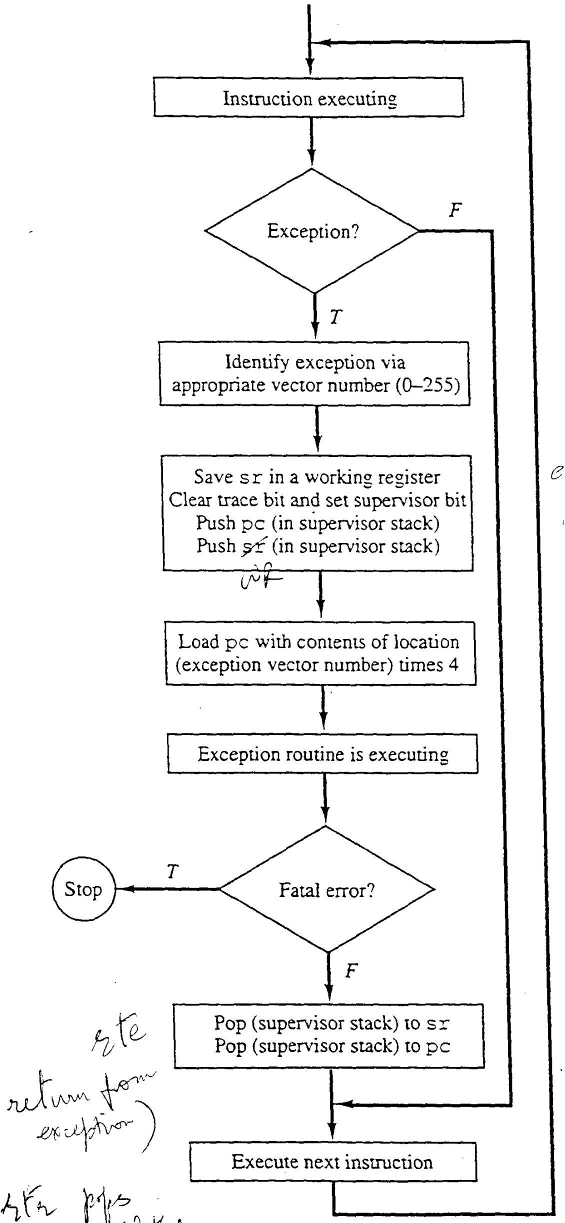
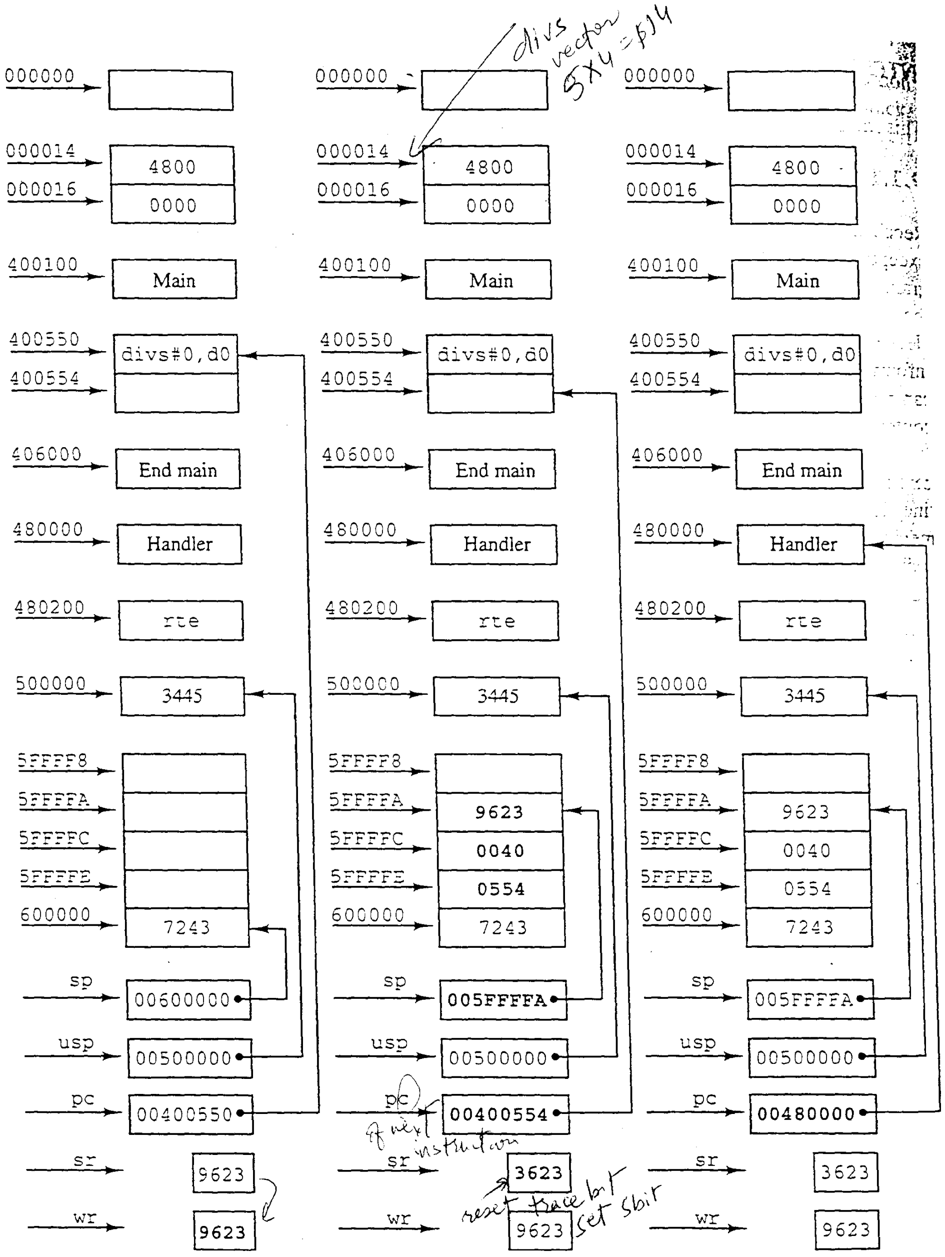

Instruction executing
Exception?
F
T
Identify exception via appropriate vector number (0–255)
Save sr in a working register
Clear trace bit and set supervisor bit
Push pc (in supervisor stack)
Push ~~sr~~ (in supervisor stack)
wr
Load pc with contents of location (exception vector number) times 4
Exception routine is executing
Stop
T
Fatal error?
F
Pop (supervisor stack) to sr
Pop (supervisor stack) to pc
Execute next instruction

Figure 9.2 Sequence of events that occur in response to exception of internal exception (except that of address error).

exceptions are handled by OS
∴ supervisor bit is set

rte
(return from exception)

rtr pops
values off the
user stack not
supervisor stack



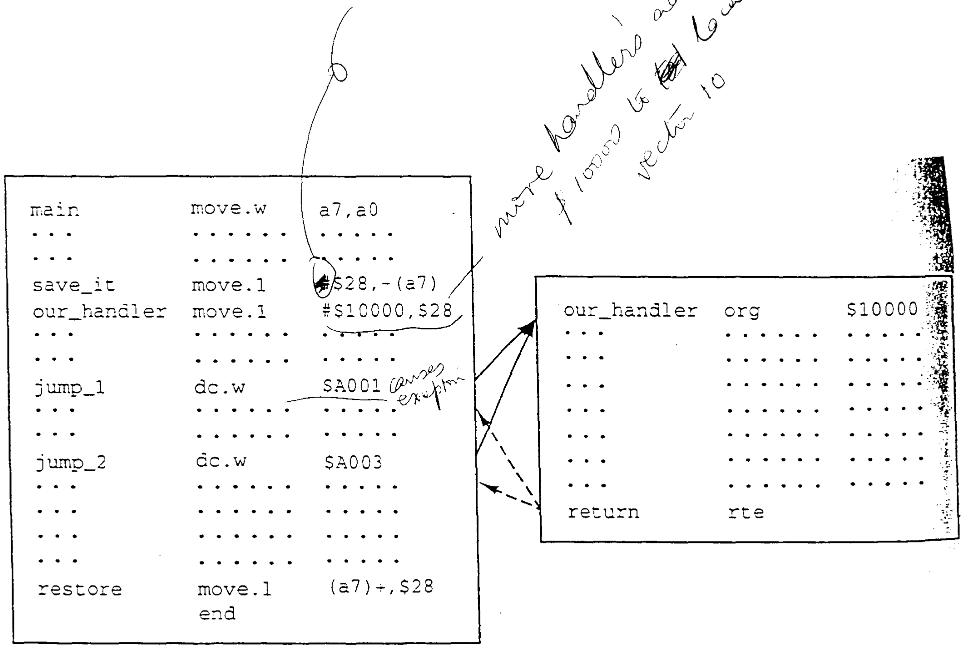
(a) Save sr

SR to WR

(b) Push pc and wr

(c) Jump to handler

Figure 9.4 Effect of division by zero exception.



```
main            move.w    a7,a0
...             ......    .....
...             ......    .....
save_it         move.l    #$28,-(a7)
our_handler     move.l    #$10000,$28
...             ......    .....
...             ......    .....
jump_1          dc.w      $A001
...             ......    .....
...             ......    .....
jump_2          dc.w      $A003
...             ......    .....
...             ......    .....
...             ......    .....
...             ......    .....
restore         move.l    (a7)+,$28
                end
```

```
our_handler     org       $10000
...             ......    .....
...             ......    .....
...             ......    .....
...             ......    .....
...             ......    .....
...             ......    .....
return          rte
```

Figure 9.5 Using line *A* emulator.

Location	First word
004004	3038
004016	5540
004020	6000

(a) Breakpoint table

Location object code

```
006000    0003
006002    FDFD

004000
004000    3E7C 7000
004004    3038 6002
004008    6100 000C
00400C    31C0 6002
004010    3E3C 00E4
004014    4E4E

004016    3F00
004018    5540
00401A    6600 0008

00401E    301F
004020    6000 0006

004024    61F0
004026    C0DF
004028    4E75
```

(b) Input program

Location object code

```
006000
006000    0003  ] dc
006002    FDFD  ]

004000          org $4000
004000    3E7C 7000
004004    4AFC 6002
004008    6100 000C
00400C    31C0 6002
004010    3E3C 00E4
004014    4E4E

004016    3F00            factorial
004018    4AFC              subroutine
00401A    6600 0008

00401E    301F
004020    4AFC 0006

004024    61F0
004026    C0DF
004028    4E75 ——— rts
```

(c) Corrupted program

Figure 9.6

The responsibility of the break handler is to replace the invalid word with original word, execute the instruction, corrupt it again, and return to user's program.

TABLE 9.14 Four exception groups according to sequence of steps required for handling an exception

Exception sequence steps		
Group	Exception type	Action
1	Illegal instruction. unimplemented instruction. privilege violation. trap, trapv, chk, trace. zero divide	1. Save SR in a WR. 2. Set supervisor bit and clear trace bit. 3. Save "PC" in system stack. 4. Push WR in system stack. 5. Jump to location indicated by exception vector.
2	Bus error. address error	1. Save SR in WR. 2. Set Supervisor bit. clear trace bit. 3. Push "PC" into the system stack. 4. Push WR in system stack. 5. "Invalid" address is pushed into the system stack. 6. Access type word is pushed onto the stack. 7. Jump to location indicated by exception vector.
3	(External) reset	1. Set supervisor bit. clear trace bit. set mask's interrupt level to 7. 2. Move longword contents of location exception vector zero to register a7. 3. Jump to location indicated by the longword contents of exception vector one.
4	Interrupt.	1. Save SR in WR. 2. Set supervisor bit. clear trace bit. set interrupt level to that of the servicing interrupt. 3. Processor informs device that issued interrupt that latter is serviced. 4. Device responds by presenting processor with exception vector number *u*. 5. Push "PC" into system stack. 6. Push WR in system stack. 7. Jump to location indicated by exception vector *u*.

TABLE 9.15 Exception priorities

Exception priorities

highest priority

Level	Exception type	Action
0	Address error, bus error, external reset	Abort current instruction and then process exception
1	Illegal instruction, unimplemented instruction, trace, interrupt, privilege violation	Complete current instruction and then process exception
2	`trap`, `trapv`, `chk`, divide by zero	Instruction execution initiates exception processing