EE 357 Unit 9

Exceptions

Status Register

- T-bit = Trace Bit
  - Used for debugging; when set, it can be used to pause the CPU after execution of each instruction
- S-bit = Supervisor Bit
  - Coldfirecan run programs in two modes: supervisor mode (for OS and other system SW) and user mode (for user apps.)
  - S = 1 => Supervisor mode / S = 0 => User Mode
- M-bit = Master/Interrupt Bit
  - Assume 0 unless told otherwise
- I-bits (IM) = Interrupt Priority Mask
  - 3-bit value indicating which interrupt priorities (≤ I) to ignore
  - Assume = $7=111_2$ for now

	15		13	12		10 9 8		4	3	2	1	0
SR:	T	0	S	M	0	I	0 0 0	X	N	Z	V	C

Supervisor and User Mode

- User applications are designed to run in user mode
- Operating System and other system software should run in supervisor mode
- Certain features/privileges are only allowed to code running in supervisor mode

Supervisor Mode Privileges

- Privileged instructions (a few examples below…)
  - MOVE #imm$_{16}$,SR => Sets SR = imm.
  - STOP #imm$_{16}$ => Halt proc. & set SR = imm.
  - User apps. shouldn't be allowed to stop or reset the computer or arbitrarily change things in the SR
- Privileged Memory or I/O access
  - Processor supports special areas of memory or I/O space that can only be accessed in supervisor mode
- Separate Stack areas
  - 2 physical A7 registers: SSP and USP
  - SSP (System SP) is used when in Supervisor Mode
  - USP (User SP) is used when in User Mode

Exceptions

- Any event that causes a break in normal execution
  - Error Conditions
    - DIV by 0, unaligned address used for .W/.L access, etc.
  - Hardware Interrupts / Events
    - Handling a keyboard press, mouse moving, new USB device, etc.
    - Will be discussed in a future lecture
  - System Calls / Traps
    - User applications calling OS code

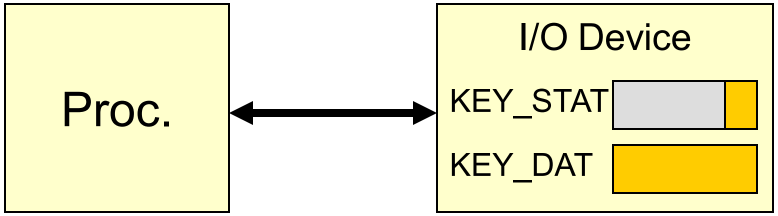
Error Condition Exceptions

- Access Error
  - No memory or I/O device responds to a read or write
- Address Error
  - Unaligned address used for W/L access
- Illegal instruction
  - Invalid opcode
- Divide by 0
- Privilege violation
  - User code attempting to perform supervisor mode operation
- Trace violation
  - Stop to call debugger
- FP Exceptions

I/O Notification

- Interrupts are a method for alerting the processor that an I/O device needs attention
- Most I/O devices have a bit in an I/O status register that is:
  - Set to ‘1’ by HW when an event occurs (e.g. key press)
  - Cleared to ‘0’ by SW when processor has handled the event

```
getkey: move.l  KEY_STAT,D0 ; get status
        andi.l  #0x00000001,D0
        beq     getkey
        move.l  KEY_DAT,D1; get key
        andi.l  0xfffffffe,KEY_STAT
```

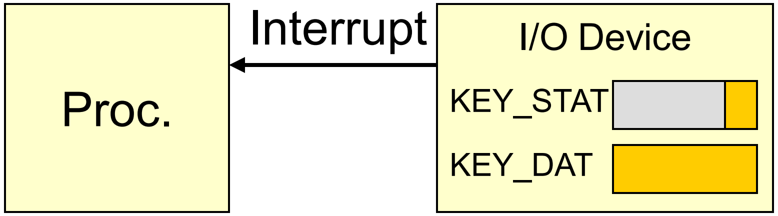


Interrupt Exceptions

- Two methods for I/O devices to indicate they need the processor’s attention
  - – Polling “busy” loop (responsibility on proc.)
    - Processor has responsibility of checking each I/O device
    - Many I/O events happen infrequently (1-10 ms) with respect to the processors ability to execute instructions (1-10 ns)
  - – Interrupts (responsibility on I/O device)
    - I/O device notifies processor only when it needs attention
    - Processor can perform “useful” work in the meantime

```
getkey: move.l  KEY_STAT,D0 ; get status
        andi.l  #0x0001,D0
        beq     getkey
        move.l  KEY_DAT,D1; get key
        andi.l  0xfffffffe,KEY_STAT
```

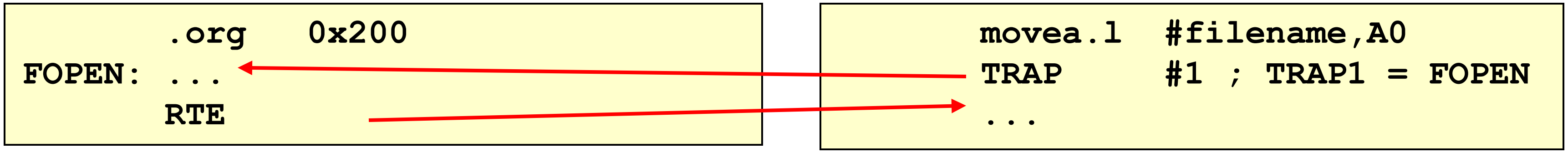
Polling Loop



Interrupt

System Calls / TRAPs

- Provide a structured method for user mode applications to call supervisor mode (OS) routines or services
  - Prevent stack corruption due to separate stacks for each mode
  - Provide a form of "dynamic" linking
- TRAP's are very similar to subroutine calls but they switch into supervisor mode when called and then back to user mode on return
- Common TRAP's or system calls are I/O service routines
  - FOPEN, FCLOSE, READ, WRITE, IOCTL, etc.



```
        .org    0x200
FOPEN:  ...
        RTE
```

```
        movea.l   #filename,A0
        TRAP      #1 ; TRAP1 = FOPEN
        ...
```

OS Code

User Code

TRAP's

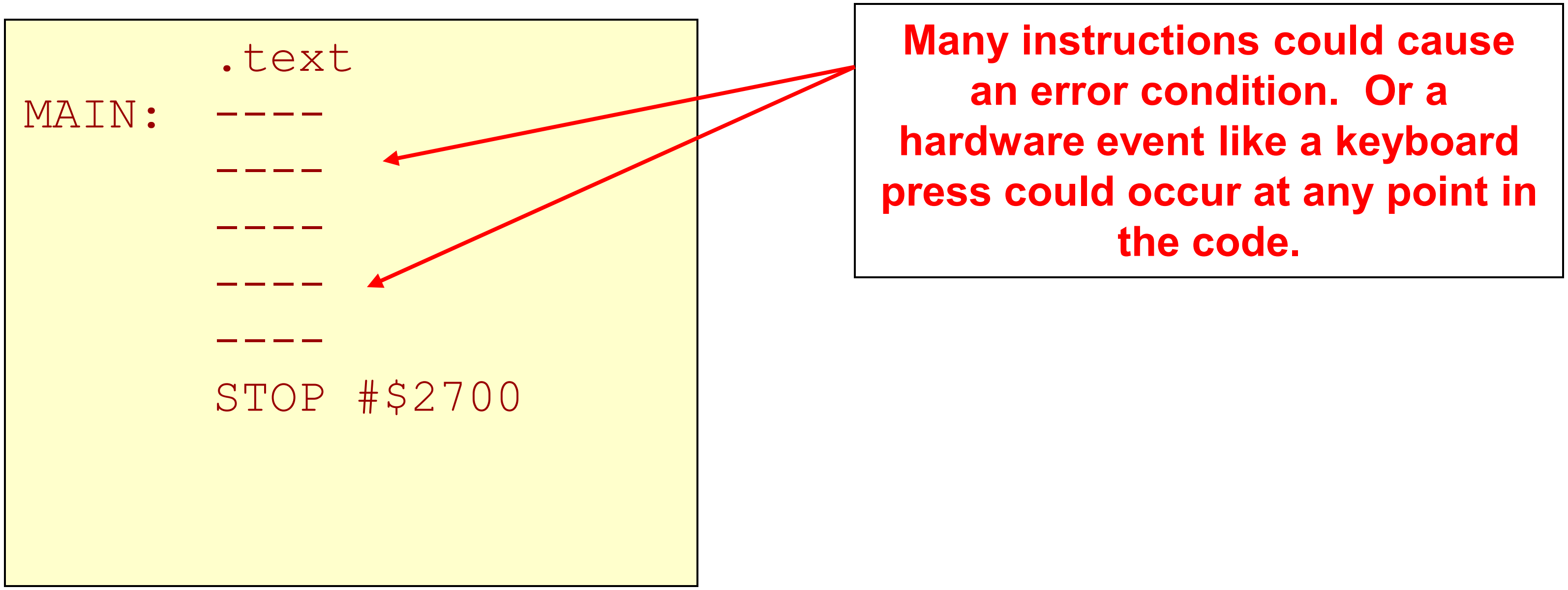
- Other systems use the terms:
  - System Calls
  - Software Interrupts
- Some TRAPS can be application-defined while others will be OS defined
- Format: TRAP #n
  - n=0..15

Exception Handling

- What do we do when an exception occurs?
  - A routine called an “exception handler” will be automatically called to attempt to handle the exception
- Problems with calling exception handlers
  - We don’t know where we will be in our code when an exception occurs, so how can we call the exception handler
  - Calling the exception handler can cause changes in state when we return to running application

Problem of Calling a Handler

- We can’t use explicit BSR/JSR instructions to call exception handlers since we don’t when they will occur



```
         .text
MAIN:    ----
         ----
         ----
         ----
         ----
         STOP #$2700
```

Many instructions could cause an error condition. Or a hardware event like a keyboard press could occur at any point in the code.

Solution for Calling a Handler

- Since we don't know when an exception will occur we have to tell the computer before hand what routine to call when the exception occurs
  - We must "register" or associate a handler routine for each exception type
  - Typically done by OS & device drivers at boot
- Associating a handler routine with an exception is accomplished using the Exception Vector Table (EVT) in memory

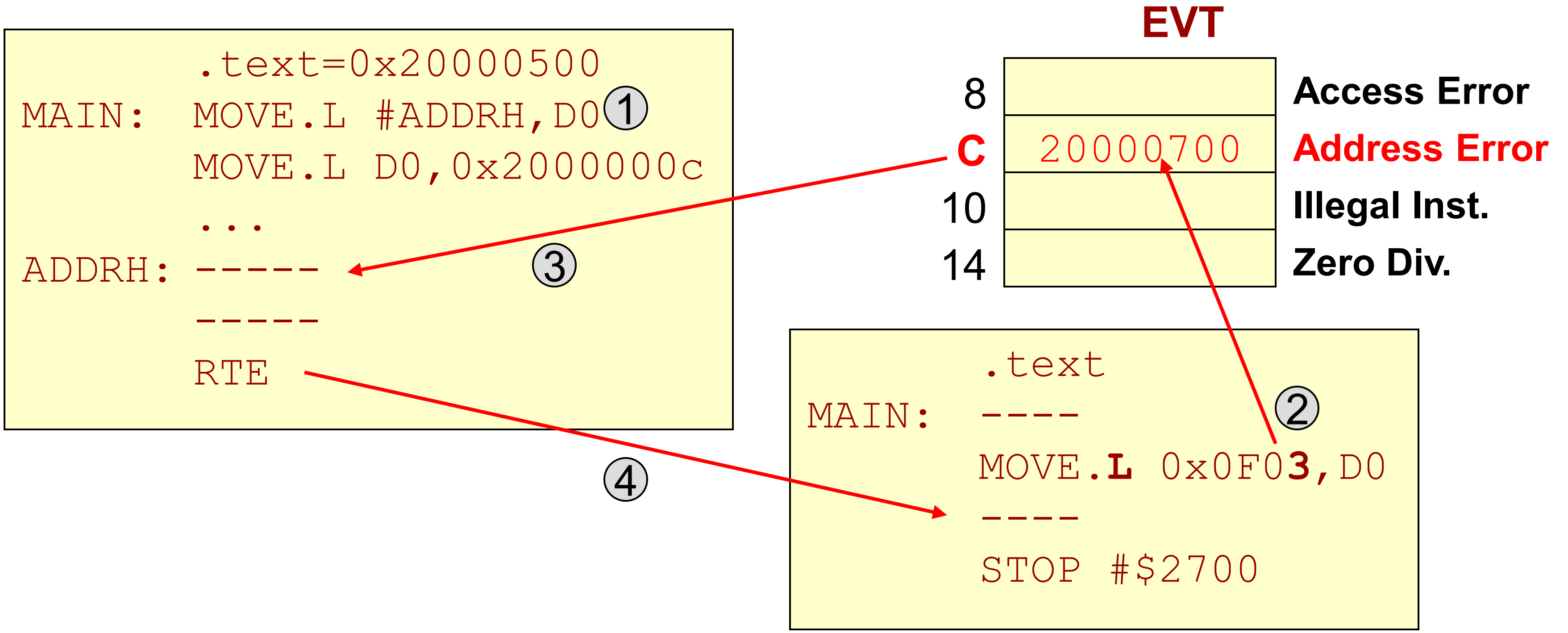
EVT

- EVT is a reserved area in memory defined from address 0x20000000-0x200003ff
- Specific entries for each exception
- Each long-word entry should be filled in with the *starting address* of the exception handler associated with that exception

0	Init. SP	
4	Init. PC	
8	Access Error	
C	Address Error	
10	Illegal Inst.	
14	Divide by 0	
18	Reserved	
1C	Reserved	EVT
20	Privilege Viol.	
	...	
64	INT1	
	...	
7C	INT7	
80	TRAP 0	
	...	
BC	TRAP 15	
C0 FC	FP/Other Errors	
100 3FC	Peripheral INT sources	

EVT Example

- If an access is made to an unaligned address the processor will automatically go to location 0x0C in memory and go to the routine at the address stored there.



```
        .text=0x20000500
MAIN:   MOVE.L #ADDRH,D0
        MOVE.L D0,0x2000000c
        ...
ADDRH:  -----
        -----
        RTE
```

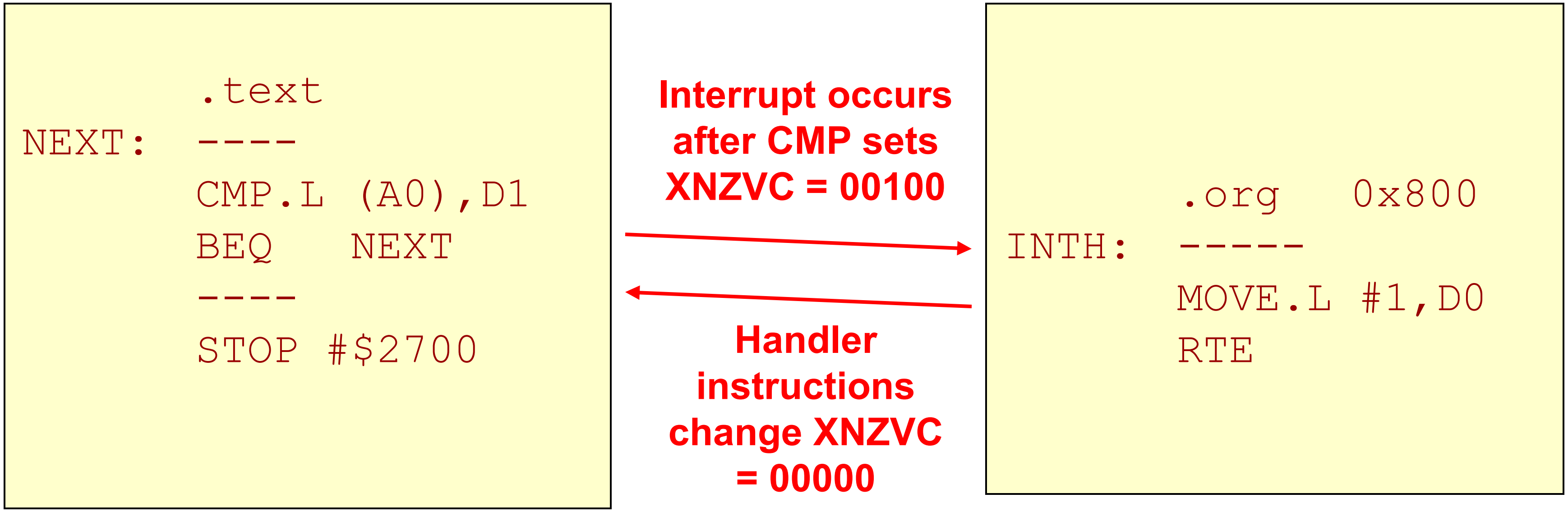
EVT

8		Access Error
C	20000700	Address Error
10		Illegal Inst.
14		Zero Div.

```
        .text
MAIN:   ----
        MOVE.L 0x0F03,D0
        ----
        STOP #$2700
```

Problem of Changed State

- Since exceptions can occur at any time, the handler instructions may change the state (e.g. condition codes) of the original code causing incorrect execution



```
        .text
NEXT:   ----
        CMP.L  (A0),D1
        BEQ    NEXT
        ----
        STOP #$2700
```

Interrupt occurs after CMP sets XNZVC = 00100

```
        .org    0x800
INTH:   -----
        MOVE.L #1,D0
        RTE
```

Handler instructions change XNZVC = 00000

BEQ would be true if no exception occurred but now BEQ will be false.

Solutions to State Changes

- When an exception occurs processor will automatically create/push an “exception stack frame” composed of 2 longwords onto the stack
  - Cause information (assume = 0x0400) + copy of SR
  - PC (return address)
- Handler’s must use RTE instead of RTS to pop off the exception stack word (i.e. 2 longwords, where as RTS pops off only 1)

31	0
Cause Info (default = 0x0400)	Copy of SR
Copy of PC (Return Address)	

Exception Handling

- When an exception occurs the processor will automatically perform the following 4 steps

1. Make a copy of SR & PC for exception stack frame
2. Set S=1, T=0 (Change into supervisor mode)
3. Push exceptions stack frame onto stack
4. Call exception handler routine using the appropriate EVT entry

Traps

```
      MOVE.L #T1,D0
      MOVE.L D0,0x20000084
      ----
      TRAP  #1
      ----
      ----

      .org 0x400
T1:   MOVE.L
      ADD.L
      MOVE.L
      RTE
```

Exception Vector Table

80	00000000	TRAP 0
84	00000000	TRAP 1
88	00000000	TRAP 2
8C	00000000	TRAP 3

Stack

3570	
3574	
3578	
357c	

SP: 357c → 357c

SR:	T		S	M		I		X	N	Z	V	C

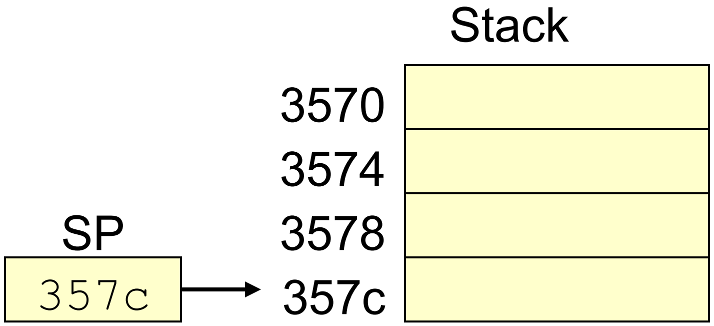
Traps – Step 0

⓪ *Initialize EVT with address of Trap Handler*

```
        MOVE.L #T1,D0
        MOVE.L D0,0x20000084
        ----
        TRAP  #1
        ----
        ----

        .org 0x400
T1:     MOVE.L
        ADD.L
        MOVE.L
        RTE
```

Exception Vector Table

80	00000000	TRAP 0
84	20000900	TRAP 1
88	00000000	TRAP 2
8C	00000000	TRAP 3



Stack

SR:	T		S	M		I		X	N	Z	V	C

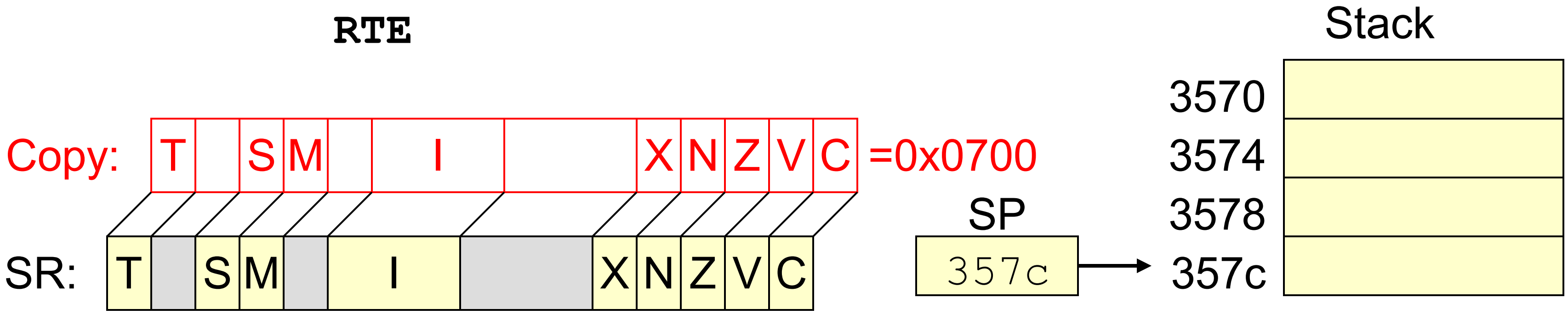
Traps – Step 1

```
      MOVE.L #T1,D0
      MOVE.L D0,0x20000084
      ----
      TRAP  #1
      ----
      ----

      .org 0x400
T1:   MOVE.L
      ADD.L
      MOVE.L
      RTE
```

① *Copy SR*

Exception Vector Table

80	00000000	TRAP 0
84	20000900	TRAP 1
88	00000000	TRAP 2
8C	00000000	TRAP 3

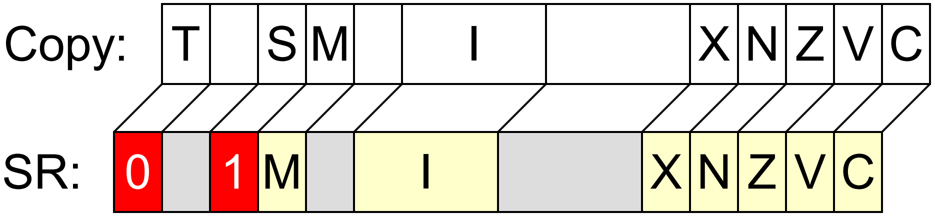


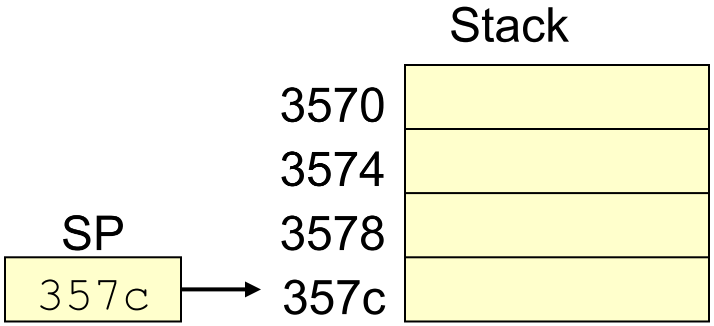
Traps – Step 2

```
     MOVE.L #T1,D0
     MOVE.L D0,0x20000084
     ----
     TRAP  #1          ② S=1,T=0
     ----
     ----

     .org 0x400
T1:  MOVE.L
     ADD.L
     MOVE.L
     RTE
```

Exception Vector Table

80	00000000	TRAP 0
84	20000900	TRAP 1
88	00000000	TRAP 2
8C	00000000	TRAP 3





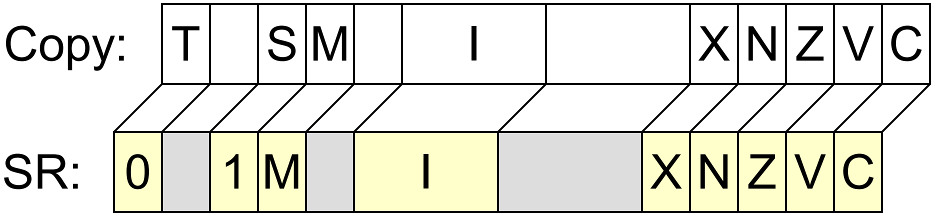
Traps – Step 3

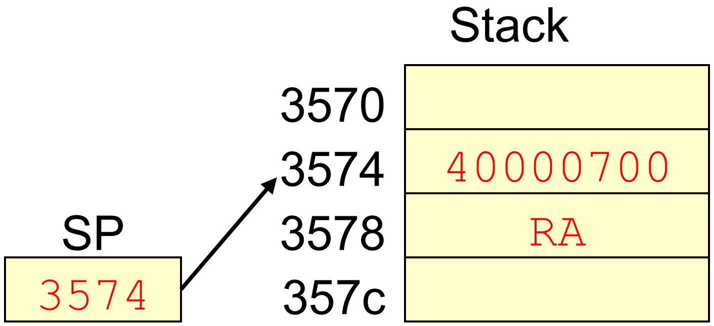
```
      MOVE.L #T1,D0
      MOVE.L D0,0x20000084
      ----
      TRAP   #1
RA    ----
      ----

      .org 0x400
T1:   MOVE.L
      ADD.L
      MOVE.L
      RTE
```

③ *Push Frame*

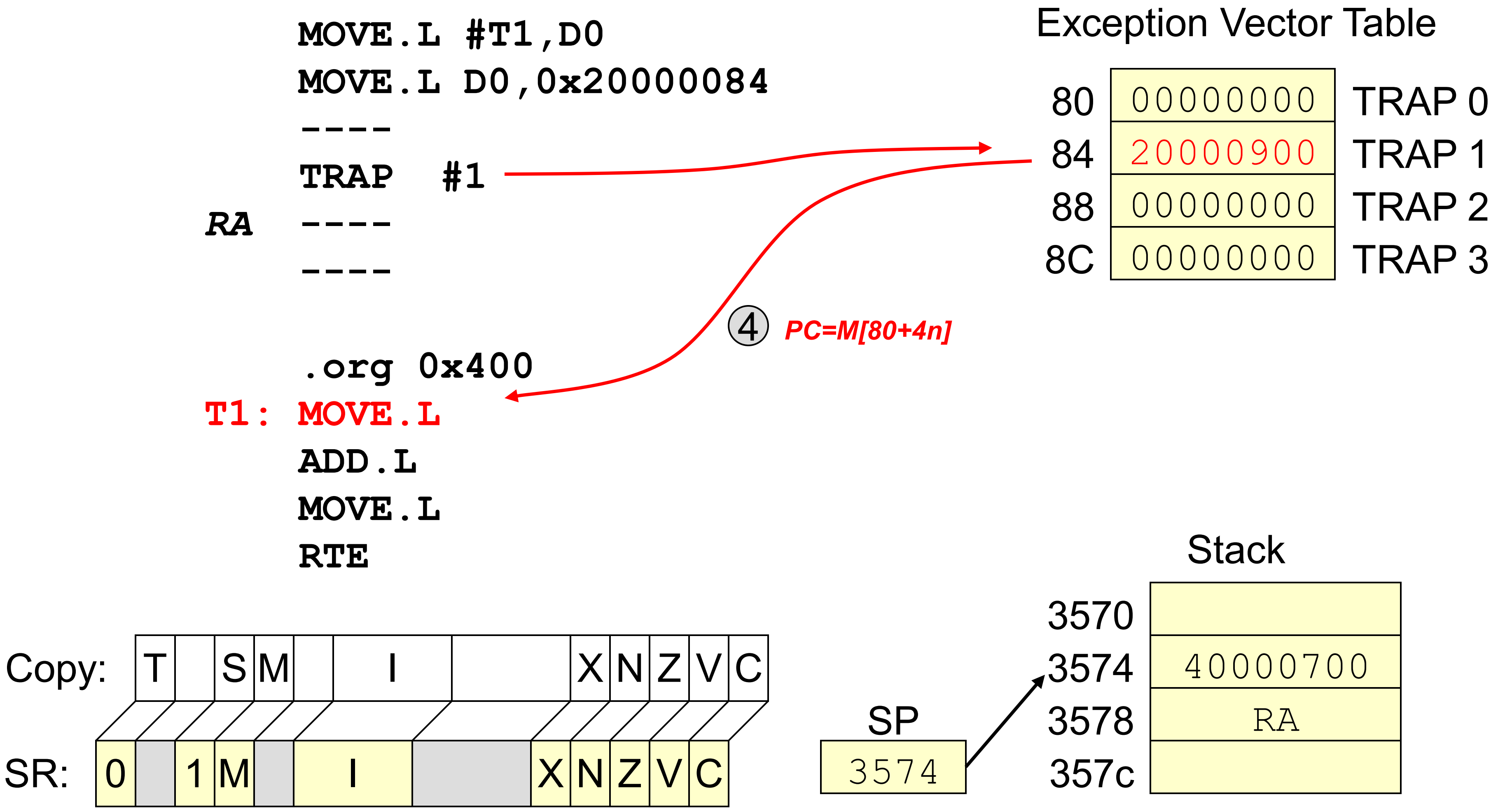
Exception Vector Table

80	00000000	TRAP 0
84	20000900	TRAP 1
88	00000000	TRAP 2
8C	00000000	TRAP 3





Traps – Step 4



```
      MOVE.L #T1,D0
      MOVE.L D0,0x20000084
      ----
      TRAP   #1
RA    ----
      ----

      .org 0x400
T1:   MOVE.L
      ADD.L
      MOVE.L
      RTE
```

Exception Vector Table

80	00000000	TRAP 0
84	20000900	TRAP 1
88	00000000	TRAP 2
8C	00000000	TRAP 3

4 PC=M[80+4n]

Stack

3570	
3574	40000700
3578	RA
357c	

SP

3574

Copy:	T		S	M		I		X	N	Z	V	C
SR:	0		1	M		I		X	N	Z	V	C

Traps – Step 5

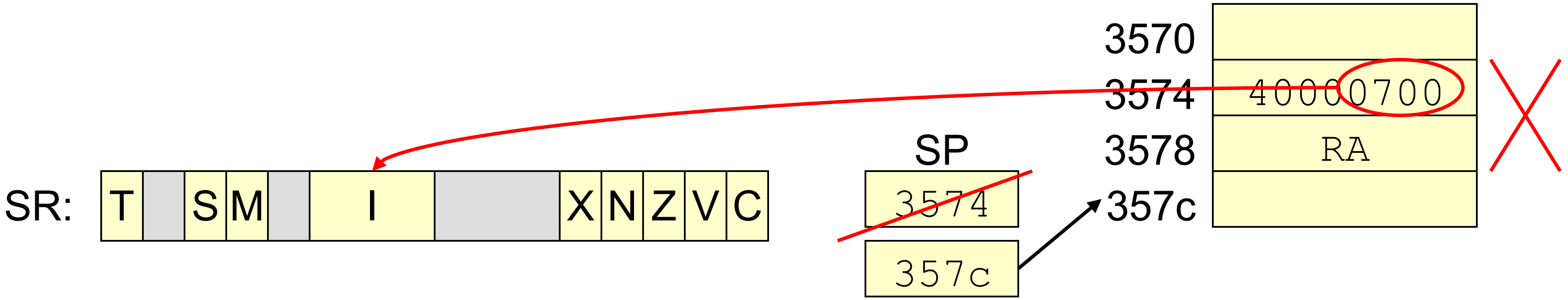
```
      MOVE.L #T1,D0
      MOVE.L D0,0x20000084
      ----
      TRAP  #1
RA    ----
      ----

      .org 0x400
T1:   MOVE.L
      ADD.L
      MOVE.L
      RTE
```

Exception Vector Table

80	00000000	TRAP 0
84	20000900	TRAP 1
88	00000000	TRAP 2
8C	00000000	TRAP 3

5 *Return using RA from stack and restore SR*

Stack



Another TRAP Example

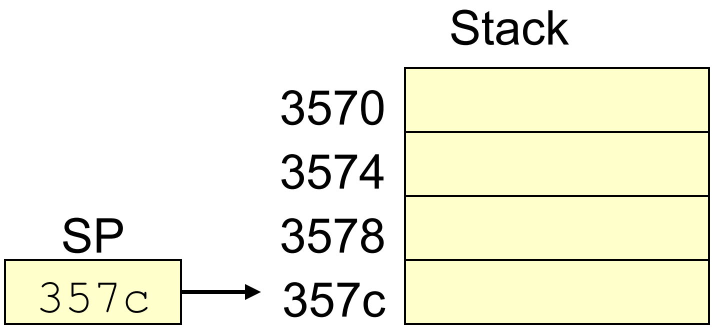
```
        MOVE.L #FOPEN,D0
        MOVE.L D0,0x20000088
        ...
        CLR.L D0
        TRAP  #2
        ----

        .org 0x200
FOPEN:  MOVE.W  #0xF123,D1
        ----
        RTE
```

Exception Vector Table

80	00000000	TRAP 0
84	00000000	TRAP 1
88	00000000	TRAP 2
8C	00000000	TRAP 3

TRAP 2 will be associated with FOPEN

	0	0	0	0 0	1 1 1	0 0 0	0	0	0	0	0
SR:	T		S		I		X	N	Z	V	C



Another TRAP Example

```
         MOVE.L #FOPEN,D0
         MOVE.L D0,0x20000088
         ...
         CLR.L D0
         TRAP  #2
         ----

         .org 0x200
FOPEN:   MOVE.W  #0xF123,D1
         ----
         RTE
```

Exception Vector Table

80	00000000	TRAP 0
84	00000000	TRAP 1
88	20000700	TRAP 2
8C	00000000	TRAP 3

EVT is initialized

SR:											
	0	0	0	0 0	1 1 1	0 0 0	0	0	0	0	0
SR:	T		S		I		X	N	Z	V	C

Stack

3570	
3574	
3578	
357c	

SP

357c →

Another TRAP Example

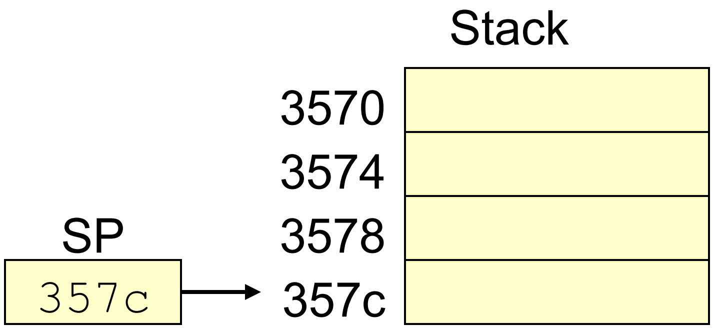
```
         MOVE.L #FOPEN,D0
         MOVE.L D0,0x20000088
         ...
         CLR.L  D0
         TRAP   #2
         ----

         .org 0x200
FOPEN:   MOVE.W  #0xF123,D1
         ----
         RTE
```

Exception Vector Table

80	00000000	TRAP 0
84	00000000	TRAP 1
88	20000700	TRAP 2
8C	00000000	TRAP 3

CLR.L D0 causes the Z flag to be set to 1

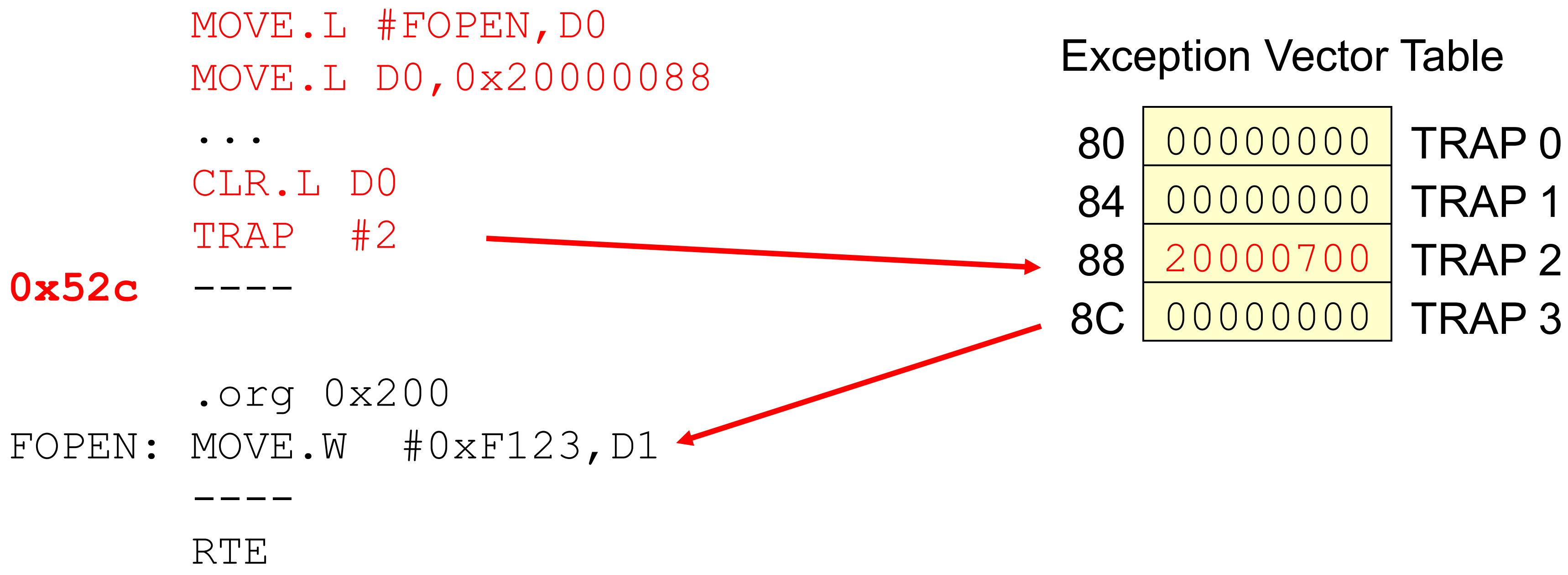
	0	0	0	0 0	1 1 1	0 0 0	0	0	1	0	0
SR:	T		S		I		X	N	Z	V	C



Stack

SP		
	3570	
	3574	
	3578	
357c	357c	

Another TRAP Example



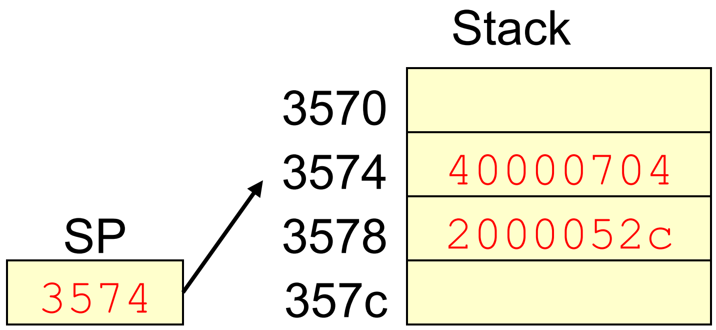
```
         MOVE.L #FOPEN,D0
         MOVE.L D0,0x20000088
         ...
         CLR.L D0
         TRAP  #2
0x52c    ----

         .org 0x200
FOPEN:   MOVE.W  #0xF123,D1
         ----
         RTE
```

Exception Vector Table

80	00000000	TRAP 0
84	00000000	TRAP 1
88	20000700	TRAP 2
8C	00000000	TRAP 3

TRAP #2 copies RA and SR onto the stack, enters supervisor mode, and calls handler.

	0	0	1	0 0	1 1 1	0 0 0	0	0	1	0	0
SR:	T		S		I		X	N	Z	V	C



Stack

SP
3574

3570	
3574	40000704
3578	2000052c
357c	

Another TRAP Example

```
        MOVE.L #FOPEN,D0
        MOVE.L D0,0x20000088
        ...
        CLR.L D0
        TRAP  #2
0x52c   ----

        .org 0x200
FOPEN:  MOVE.W  #0xF123,D1
        ----
        RTE
```

Exception Vector Table

80	00000000	TRAP 0
84	00000000	TRAP 1
88	20000700	TRAP 2
8C	00000000	TRAP 3

Inside the TRAP call, the SR is free to change (N=1,Z=0 due to the MOVE)

	0	0	1	0 0	1 1 1	0 0 0	**0**	**1**	**0**	**0**	**0**
SR:	T		S		I		X	N	Z	V	C

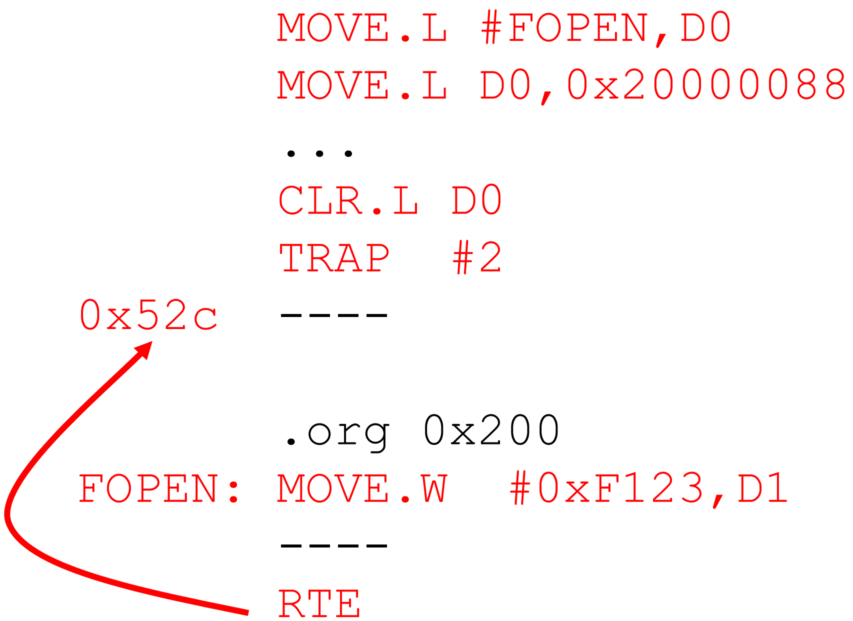
SP

3574

Stack

3570	
3574	40000704
3578	2000052c
357c	

Another TRAP Example



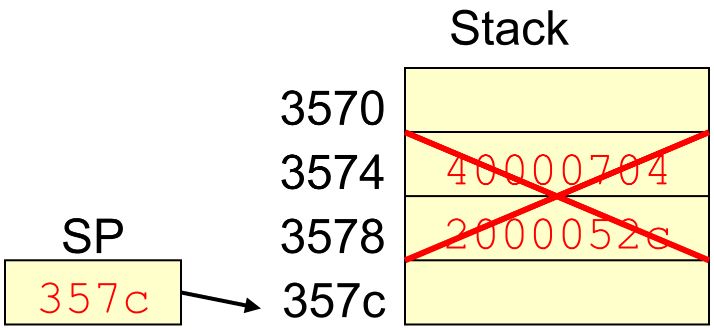
```
        MOVE.L #FOPEN,D0
        MOVE.L D0,0x20000088
        ...
        CLR.L  D0
        TRAP   #2
0x52c   ----

        .org 0x200
FOPEN:  MOVE.W  #0xF123,D1
        ----
        RTE
```

Exception Vector Table

80	00000000	TRAP 0
84	00000000	TRAP 1
88	20000700	TRAP 2
8C	00000000	TRAP 3

RTE restores SR and returns to RA. Now we are back in user mode with the Z=1 again.

	0	0	0	0 0	1 1 1	0 0 0	0	0	1	0	0
SR:	T		S		I		X	N	Z	V	C



Stack

SP

357c

3570	
3574	~~40000704~~
3578	~~2000052c~~
357c	

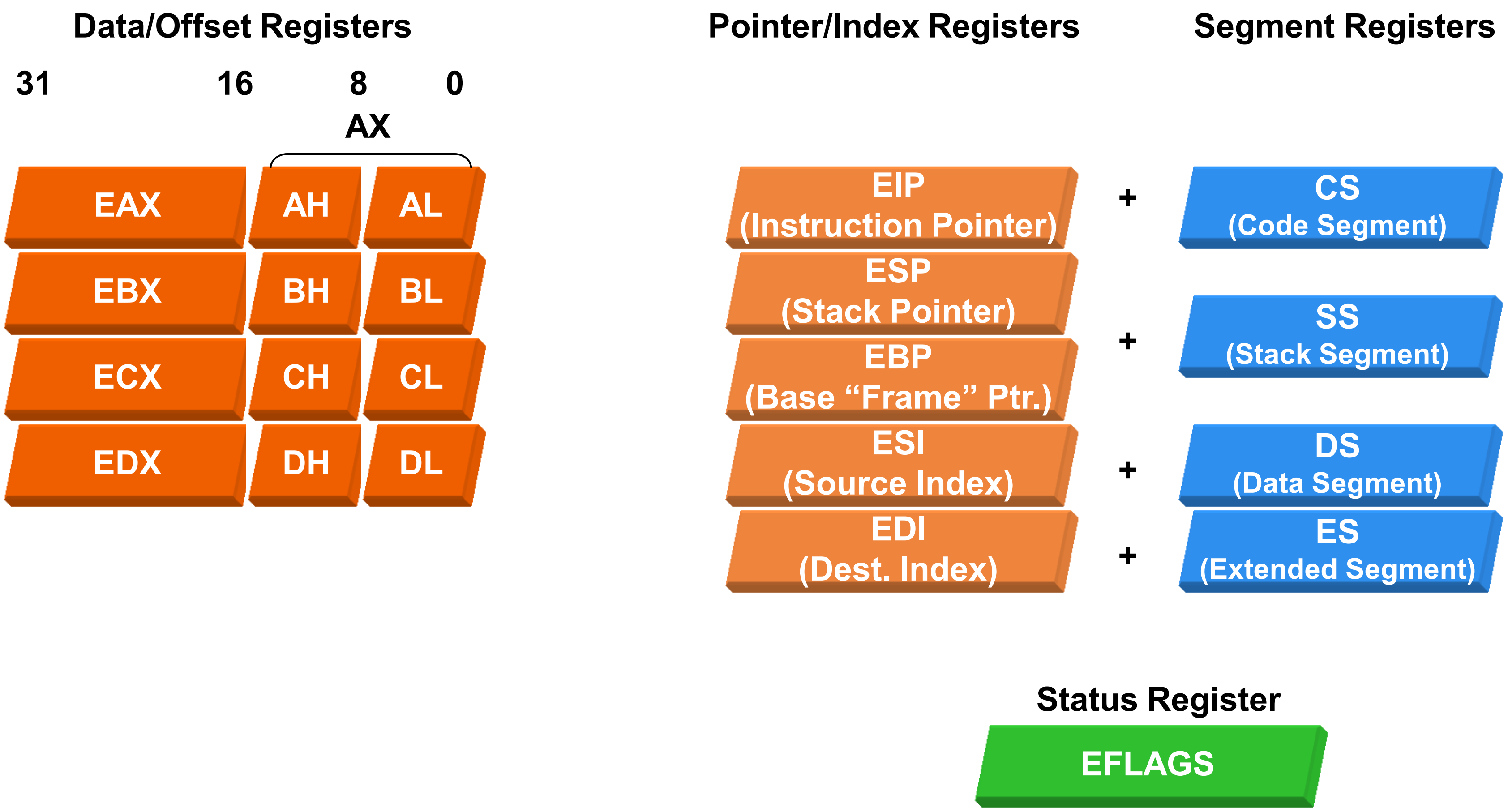
Bonus Material

Intel Architectures

Intel Architectures

Processor	Year	Address Size	Data Size
8086	1978	20	16
80286	1982	24	16
80386/486	’85/’89	32	32
Pentium	1993	32	32
Pentium 4	2000	32	32
Core 2 Duo	2006	64	64

Intel (IA-32) Architectures



Data/Offset Registers

31 16 8 0

AX

EAX AH AL

EBX BH BL

ECX CH CL

EDX DH DL

Pointer/Index Registers

EIP (Instruction Pointer)

ESP (Stack Pointer)

EBP (Base “Frame” Ptr.)

ESI (Source Index)

EDI (Dest. Index)

Segment Registers

+ CS (Code Segment)

+ SS (Stack Segment)

+ DS (Data Segment)

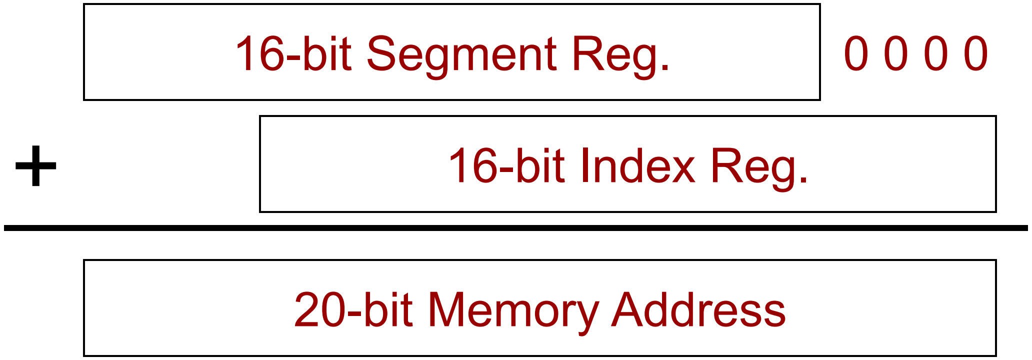
+ ES (Extended Segment)

Status Register

EFLAGS

Real Mode Addressing

- How to make 20-bit addr. w/ 16-bit reg's???
  - Use 2 16-bit registers
  - (Segment register * 16) + Index Reg.
- Format:
  - Seg Reg:Index Reg (e.g. CS:IP)



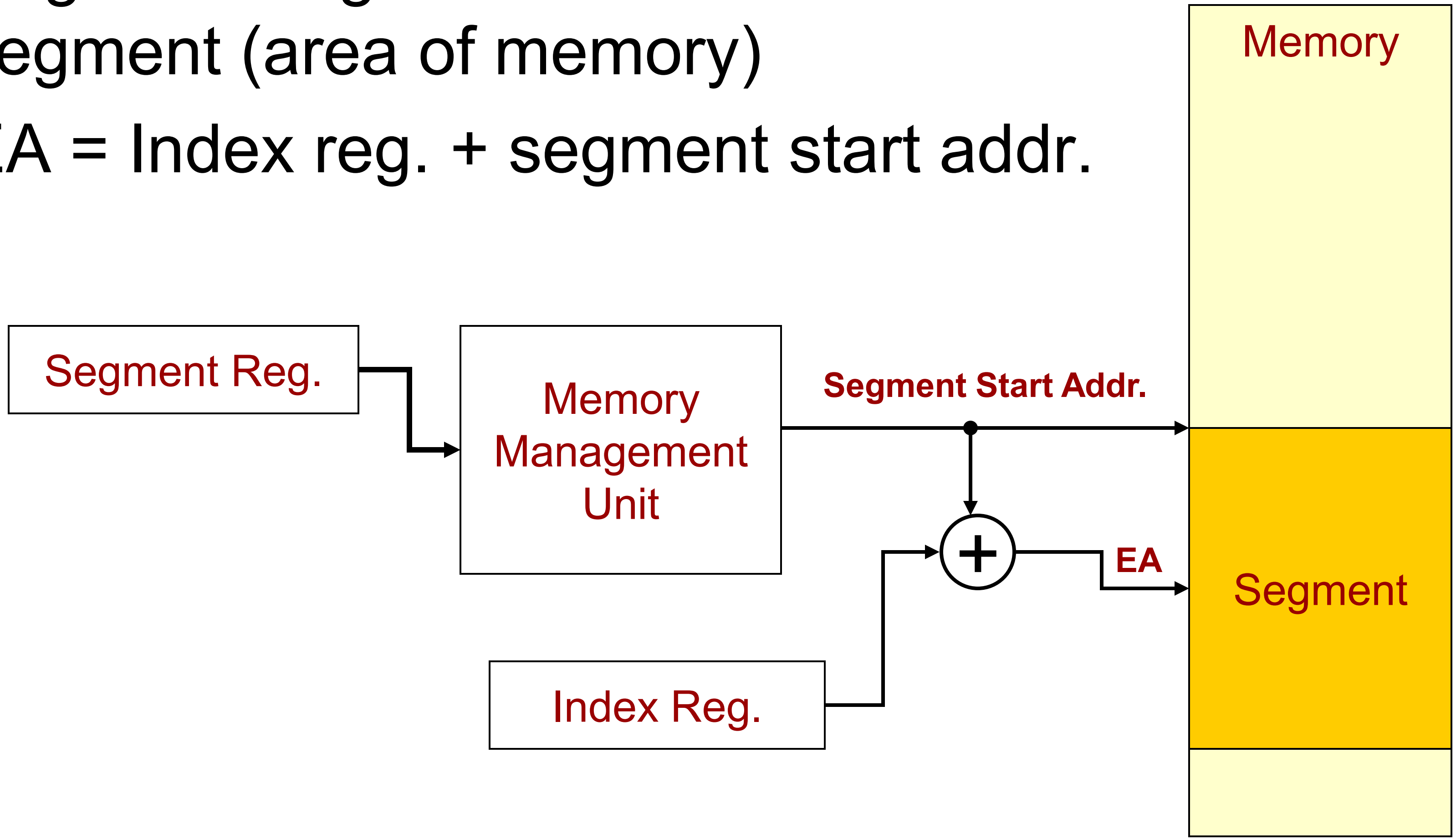
Examples:

$1A5C:$405E → $1A5C0 + $405E = $1E61E

$74AB:$E892 → $74AB0 + $E892 = $83345

Protected Mode Addressing

- Segment Register value selects a segment (area of memory)
- EA = Index reg. + segment start addr.



IA-32 Addressing Modes

Name	Example	Effective Address
Immediate	MOV EAX,5	Operand = Value
Direct	MOV EAX,[100]	EA = Addr
Register	MOV EAX,EDX	EA = Reg
Register indirect	MOV EAX,[EBX]	EA = (Reg)
Base w/ Disp.	MOV EAX,[EBP+60]	EA = (Reg) + Disp
Index w/ Disp.	MOV EAX,[ESI*4+10]	EA = (Reg)*S + Disp.
Base w/ Index	MOV EAX,[EBP + ESI*4]	EA = (Reg1)+(Reg2)*S
Base w/ Index & Disp.	MOV EAX,[EBP+ESI*4+100]	EA = (Reg1)+(Reg2)*S + Disp.