

Coprocessor Operations using NASM

R. V. Bidwe

PICT, Pune.

rvbidwe@pict.edu

Agenda

1. Processor Vs Co-processor
2. Writing a ALP for Co-processor
3. Sample ALP for Co-processor
4. What is TSR?
5. How TSR's are executed in system
6. Sample TSR program

1. Processor Vs. Co-processor

- Difference between Processor and Co-processor
 - i. **Functions and Features**
 - ii. **Instruction Set**
 - iii. **Register Organization**

Processor

- Processor performs operations on Integer numbers.
- Integer numbers are usually stored in Hexadecimal format with Packed BCD representation.
- Can perform Arithmetic and Logical Operations.
- Can handle data types: 16,32, and 64 bit integers.

Co-processor

- Processor performs operations on both Floating Point numbers and Integer numbers.
- Floating Point numbers are represented using IEEE standards.
- Can perform upto 68 additional arithmetic, trigonometric, exponential, & logarithmic instructions.
- Can handle data types: 16,32, and 64 bit integers; 32,64, and 80 bit floating-point real numbers; and up to 18-digit (BCD) operands.

Instruction Set

- As 80386 and 80387 are completely different processors, they have different
 - **Instruction Set**
 - **Bandwidth**
 - **Clock speed**

- For processor instructions, Set of Operands are expected, where actually operations are performed.
- But for Coprocessor instructions, Many of those do not have operands. In this case by default operations will be performed on TOP value.

80386 Register Organization

General registers

EAX		AX
EBX		BX
ECX		CX
EDX		DX

ESP		SP
EBP		BP
ESI		SI
EDI		DI

Program status

FLAGS register
Instruction pointer

80387 Register Organization

	79	78	64	63	0	Tag Field
						1 0
R0	Sign	Exponent	Significand			
R1						
R2						
R3						
R4						
R5						
R6						
R7						

Register use in Co-processor

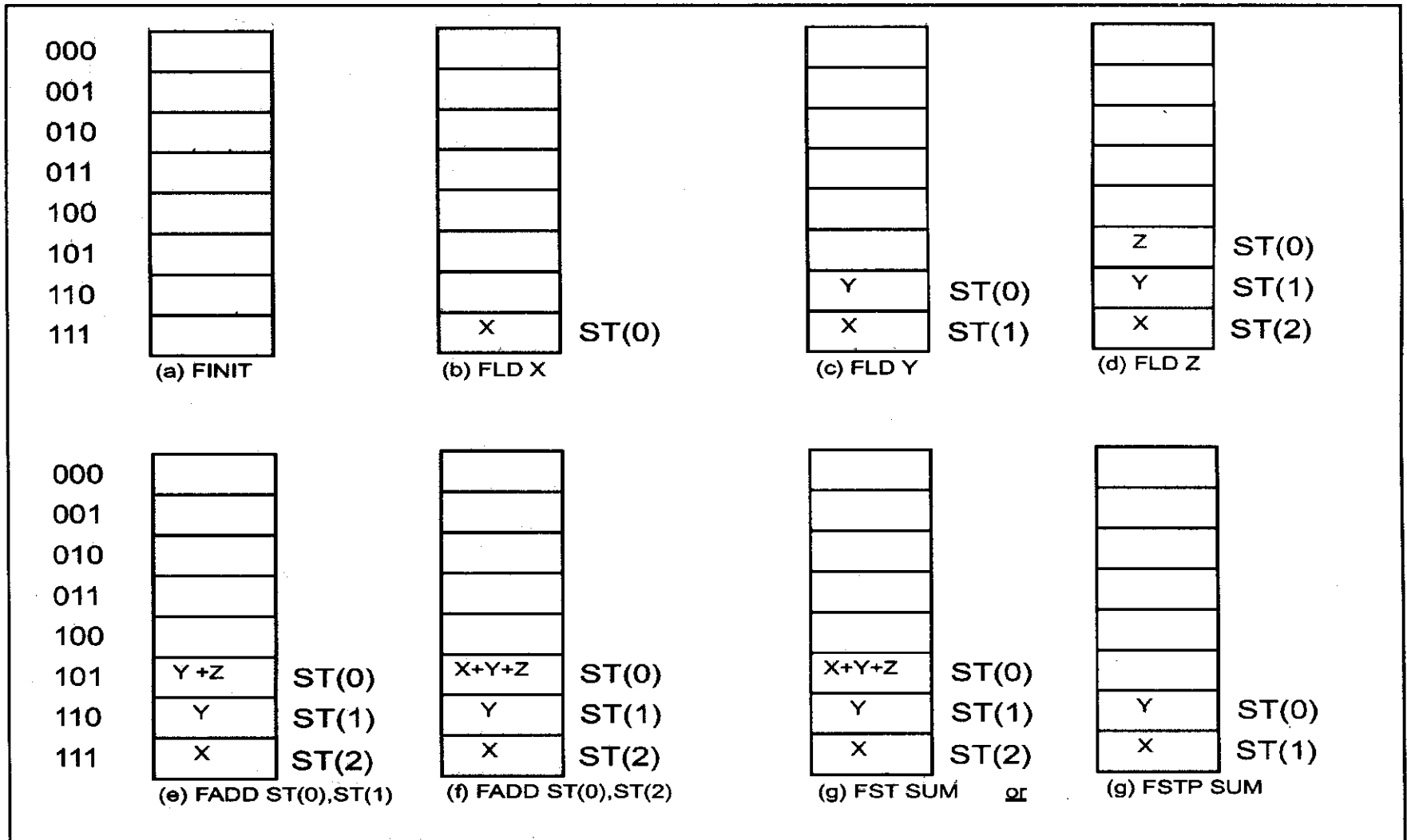


Figure 20-2. Stack Diagram for Example 20-5

Some IMP Instructions

- **FINIT: (Initialize Floating Point Unit)**
 - Initialize FPU after checking for pending unmasked floating-point exceptions.
 - Syntax: FINIT (no operand)

- **FLD:** Push one of seven commonly used constants (in double extended-precision floating-point format) onto the FPU register stack. **It sets TOP to ST0.**

Mnemonic	Description
FLD1	Push 1 to ST0
FLDL2T	Push $\log_2(10)$ to ST0
FLDL2E	Push $\log_2(e)$ to ST0
FLDPI	Push PI to ST0
FLDLG2	Push $\log_{10}(2)$ to ST0
FLDLN2	Push $\log_e(2)$ to ST0
FLDZ	Push 0 to ST0

- **FBST/ FBSTP: Store BCD Integer / Pop**
 - Converts the value in the ST(0) register to an 18-digit packed BCD integer, stores the result in the destination operand, and pops the register stack (For FBSTP).

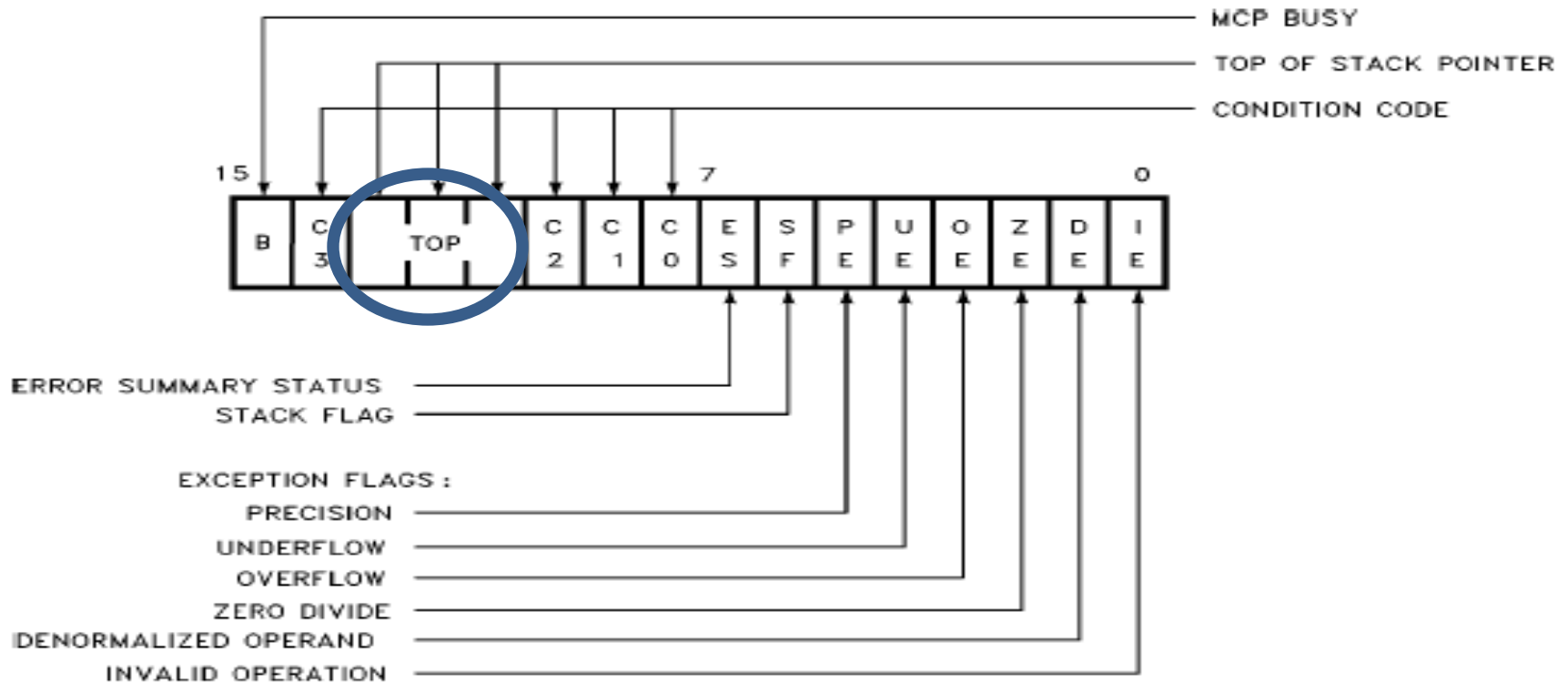
Instruction	64-Bit Mode
FBST m80bcd	Store ST(0) in m80bcd
FBSTP m80bcd	Store ST(0) in m80bcd and pop ST(0).

FBSTP Instruction

If ST0: (1234567890ABCDEF4321) in IEEE precision format.

Memory location	Data
1000000A h	
10000009 h	12
10000008 h	34
10000007 h	56
10000006 h	78
10000005 h	90
10000004 h	AB
10000003 h	CD
10000002 h	EF
10000001 h	34
10000000 h	21

Coprocessor Status Word



ES is set if any unmasked exception bit is set; cleared otherwise.
See Table 2.2 for interpretation of condition code.

TOP values:

000 = Register 0 is Top of Stack
001 = Register 1 is Top of Stack

⋮

111 = Register 7 is Top of Stack

For definitions of exceptions, refer to the section entitled
"Exception Handling"

Writing a ALP for Co-processor

(By considering few limitations of ALP)

- Steps
 - Initialize Co-processor (TOP is set to 0)
 - Load zero to top of stack
 - Perform the required operation
 - Get answer from stack and store it in a BCD format
 - Convert the answer to ASCII and Print
 - End

Sample ALP using Co-processor

Case Study No. 1: Calculating Mean,
Variance and Standard Deviation.

Case Study No. 2: Calculating Roots of
Quadratic Equations.

Case Study 1

- If a, b, c, d are the four numbers then
 - **Mean = f** = $(a+b+c+d)/4$
 - **Variance = g** = $[(a-f)^2 + (b-f)^2 + (c-f)^2 + (d-f)^2]/4$
 - **SD = h** = $\text{SquareRoot}(\text{Variance})$

- **Input:**

- array: dd 102.56,198.21,100.67,230.78,67.93
- arraycnt: dw 05
- dpoint: db '.'

Step 1: Initialize Co-processor (TOP is set to 0)

—FINIT

Step 2: Load zero to top of stack

—FLDZ

- **Step 3: Calculating Mean**

```
mov rcx,0
mov cx,[arraycnt]
mov rsi,array
up:
fadd dword[rsi]           ;Add value to ST0.
add rsi,4
loop up

fidiv word[arraycnt]
fst dword[mean]
```

- **Step 4: Calculating Variance**

```
mov rcx,00  
mov cx,[arraycnt]  
mov rsi, array
```

```
FLDZ
```

```
up1:
```

```
FLDZ
```

```
;Load zero to ST0, Sets TOP=ST0
```

```
FLD dword[rsi]
```

```
;Load value to ST0.
```

```
FSUB dword[mean]
```

```
FMUL ST0
```

```
;multiply ST0 with ST0
```

```
FADD
```

```
;Add ST0 with ST1, answer stored to ST1.
```

```
; Sets TOP=ST1.
```

```
add rsi,4
```

```
loop up1
```

```
FIDIV word[arraycnt]
```

```
FST dword[variance]
```

- **Step 5: Calculating Standard Deviation**

FLD dword[variance] ;Load variance to ST0.

FSQRT

FST dword[deviation]

- **Step 6: Printing the answer**

```
fbstp tword[resbuff]
```

```
Mov byte[cnt],9
```

```
mov rsi,resbuff+9
```

```
up2:
```

```
push rsi
```

```
mov bl,byte[rsi]
```

```
call disp8_proc
```

```
;Convert value to ASCII
```

```
linuxsyscall 01,01,dispbuff,2
```

```
;Display answer
```

```
pop rsi
```

```
dec rsi
```

```
Dec byte[cnt]
```

```
Jnz up2
```

Case Study 2

- Calculating the roots of quadratic equation:
Determinant (delta)= b^2-4ac

If determinant > 0 ,

$$\text{root1} = \frac{-b + \sqrt{(b^2 - 4ac)}}{2a}$$

$$\text{root2} = \frac{-b - \sqrt{(b^2 - 4ac)}}{2a}$$

Two Real Roots

If determinant $= 0$,

$$\text{root1} = \text{root2} = \frac{-b}{2a}$$

Equal Real Roots

If determinant < 0 ,

$$\text{root1} = \frac{-b}{2a} + i \frac{\sqrt{-(b^2 - 4ac)}}{2a}$$

$$\text{root2} = \frac{-b}{2a} - i \frac{\sqrt{-(b^2 - 4ac)}}{2a}$$

**Two Imaginary
Roots**

section .data

ff1: db "%lf +i %lf",10

ff2: db "%lf -i %lf",10

formatpi: db "%d",10

formatpf: db "%lf",10

formatsf: db "%lf"

four: dq 4

two: dq 2

%macro myprintf 1

mov rdi,formatpf

;Data Type

sub rsp,8

movsd xmm0,[%1]

mov rax,1

;Number of arguments

call printf

add rsp,8

%endmacro

```
%macro myscanf 1
    mov rdi,formatsf
    mov rax,0
    sub rsp,8
    mov rsi,rsi
    call scanf
    mov r8,qword[rsp]
    mov qword[%1],r8
    add rsp,8
%endmacro
```

section .text

global main

main:

extern printf

extern scanf

;-----accept inputs

myscanf a

myscanf b

myscanf c

- **Step 1: Calculate Determinant**

;----calculate b square

fld qword[b]

fmul qword[b]

fstp qword[b2]

;-----calculate 4ac

fild qword[four]

fmul qword[a]

fmul qword[c]

fstp qword[fac]

fld qword[b2]

fsub qword[fac]

fstp qword[delta]

;----calculate 2a

fild qword[two]

fmul qword[a]

fstp qword[ta]

btr qword[delta],63 ;-----tests the bit, sets the
carry flag if set and
clears the bit too

jc imaginary

- **Step 2: Printing the real roots**

```
fld qword[delta]
fsqrt
fstp qword[rdelta]
```

```
fldz
fsub qword[b]
fadd qword[rdelta]
fdiv qword[ta]
fstp qword[r1]
myprintf r1                ;Print Root 1
```

```
fldz
fsub qword[b]
fsub qword[rdelta]
fdiv qword[ta]
fstp qword[r2]
myprintf r2                ;Print Root 2
```

- **Step 3: Printing an imaginary roots**

[As we can not directly print “i”, Following variables are defined in Section .data

```
ff1: db "%lf +i %lf",10
```

```
ff2: db "%lf -i %lf",10 ]
```

```
fld qword[delta]
```

```
fsqrt
```

```
fstp qword[rdelta]
```

```
fldz
```

```
fsub qword[b]
```

```
fdiv qword[ta]
```

```
fstp qword[realn]
```

```
fld qword[rdelta]
```

```
fdiv qword[ta]
```

```
fstp qword[img1]
```

```
myprintf img1
```

```
;Print imaginary roots
```

Printing Imaginary roots

```
mov rdi,ff1
sub rsp,8
movsd xmm0,[realn]
movsd xmm1,[img1]
mov rax,2
call printf
add rsp,8
```

```
mov rdi,ff2
sub rsp,8
movsd xmm0,[realn]
movsd xmm1,[img1]
mov rax,2
call printf
add rsp,8
```


Terminate-and-stay-resident (TSR)

- A terminate and stay resident (TSR) program is one that is **set up to be loaded and then remain in computer memory so that it is quickly accessible** when a user presses a certain keyboard combination.
- A TSR is a software program that remains in memory until it is needed, and then performs some function.
- Eg. **Screen saver, virus scanner, notepad, calculator.**

TSR's Vs. Normal Programs

- Normal programs produces results after executing all instructions written in it.
- Instead TSR program generates an Interrupt after required event has occurred.
- For every generated interrupt a separate ISR is stored in system.
- After executing that ISR, results will be displayed in the system.

Sample TSR Program

- Case Study 1: TSR to print Real Time Clock (RTC) on screen.

Steps

- **Make Program Resident**
 - Clear Interrupt Flag
 - Push Data (if any)
 - Get Interrupt Vector from IVT
 - Set Interrupt Vector from IVT
 - Make Program resident
- **Write ISR**
- **Exit**

ISR for RTC

- Push Data
- Get Address of Video RAM
- Get System Clock
 - CH=Hours, CL=Mins, DH=Sec
- Convert Values and Print

- **Step 1: Make Program Resident**

INIT:

```
MOV AX,CS                ;Initialize data
MOV DS,AX

CLI                      ;Clear Interrupt Flag

MOV AH,35H               ;Get Interrupt vector Data and
                        store it
MOV AL,08H
INT 21H

MOV OLD_IP,BX
MOV OLD_CS,ES

MOV AH,25H               ;Set new Interrupt vector
MOV AL,08H
LEA DX,MY_TSR
INT 21H

MOV AH,31H               ;Make program Transient
MOV DX,OFFSET INIT
STI
INT 21H
```

- **Step 2: ISR for RTC**

MY_ISR:

```
PUSH AX  
PUSH BX  
PUSH CX  
PUSH DX  
PUSH SI  
PUSH DI  
PUSH ES
```

```
MOV AX,0B800H           ;Address of Video RAM  
MOV ES,AX  
MOV DI,3650
```

```
MOV AH,02H              ;To Get System Clock  
INT 1AH                 ;CH=Hrs, CL=Mins,DH=Sec
```

Thank You...!!!