Microprocessors & Microcontrollers

Second Year

Electrical Engineering Department

College of Engineering

Basrah University

2024-2025

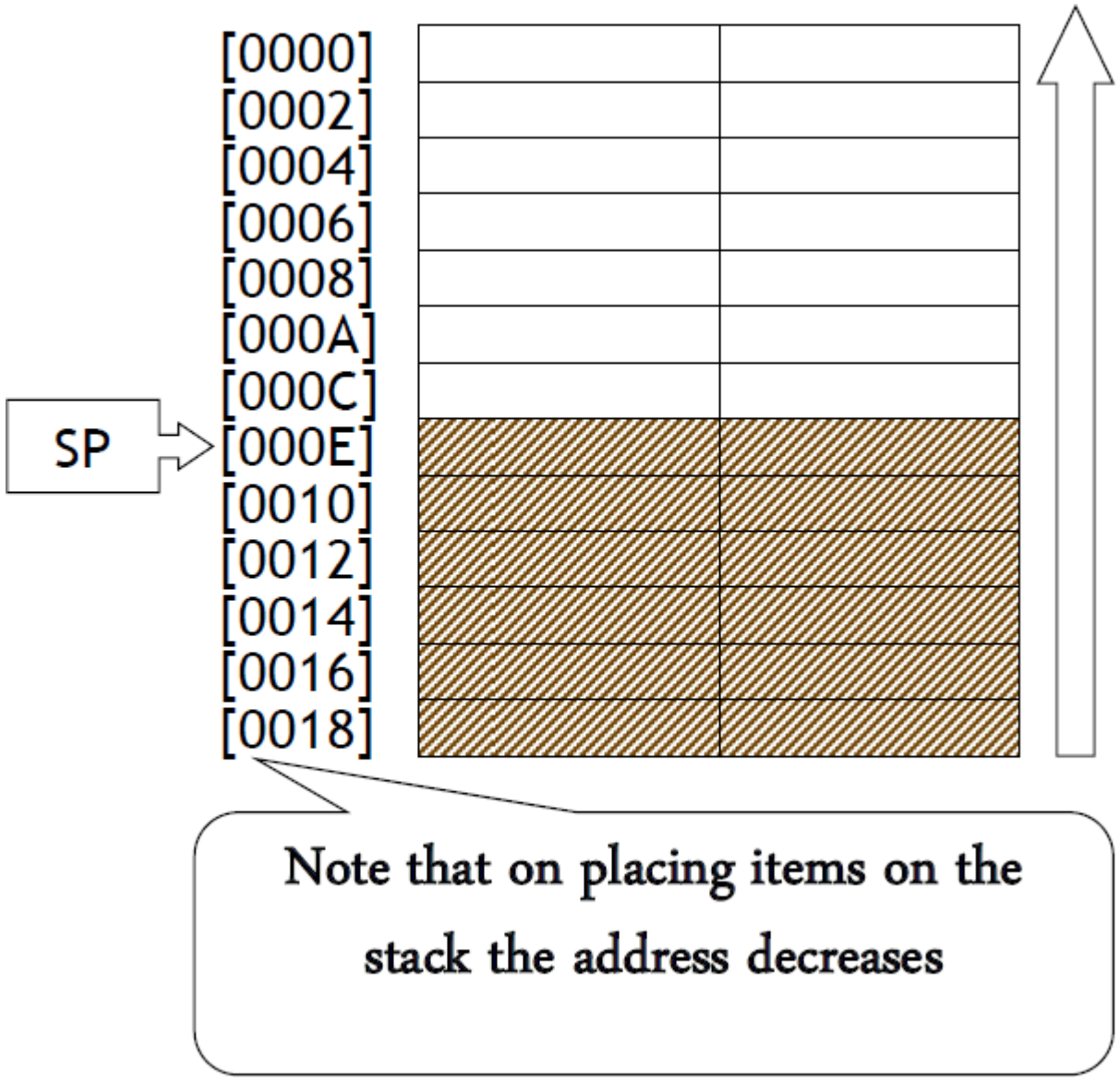
3: Stack and Interrupts of 8086

Stack of 8086

- The stack is a block of memory that may be used for temporarily storing the contents of registers inside CPU.
- Stack is accessed by using SP and SS.
- Stack is a Top Down Data Structure whose elements are accessed by using a pointer (SP,SS).
- The stack is required when CALL instruction is used.
    - Push
    - Pop
    - Top of stack
    - Stack pointer
    - LIFO
- The Stack is a portion of memory which, like a stack of plates in a canteen, is organized on a Last-In-First-Out basis.
- Thus the item which was put last on the stack is the first to be withdrawn

The Stack Pointer

- The Stack pointer keeps track of the position of the last item placed on the stack (i.e. the Top of Stack)
- The Stack is organized in words, (i.e. two bytes at a time). Thus the stack pointer is incremented or decremented by 2.
- The Stack Pointer points to the last occupied locations on the stack



Stack instructions

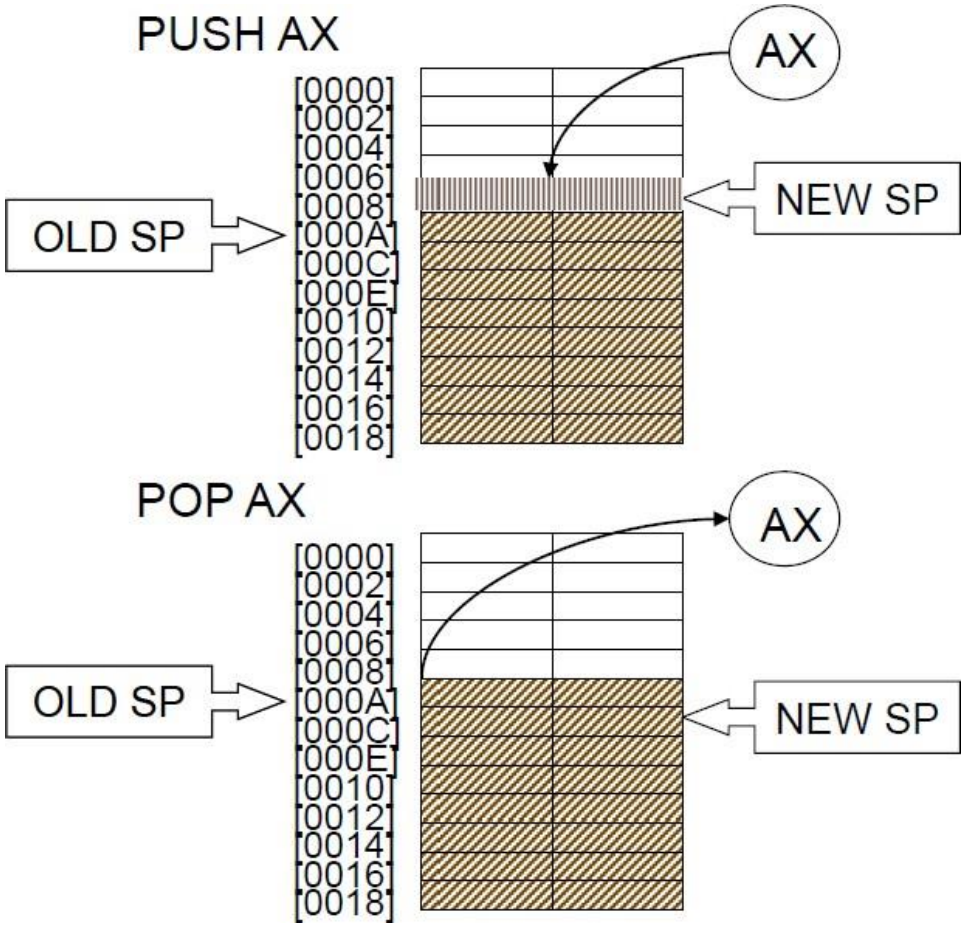
Name	Mnemonic and Format	Description
Push onto Stack	*push src*	(sp)←(sp)-2 ((sp))←(src)
Pop from Stack	*pop dst*	(dst)←((sp)) (sp)←(sp)+2
Push Flags	*pushf*	(sp)←(sp)-2 ((sp))←(psw)
Pop Flags	*popf*	(psw)←((sp)) (sp)←(sp)+2

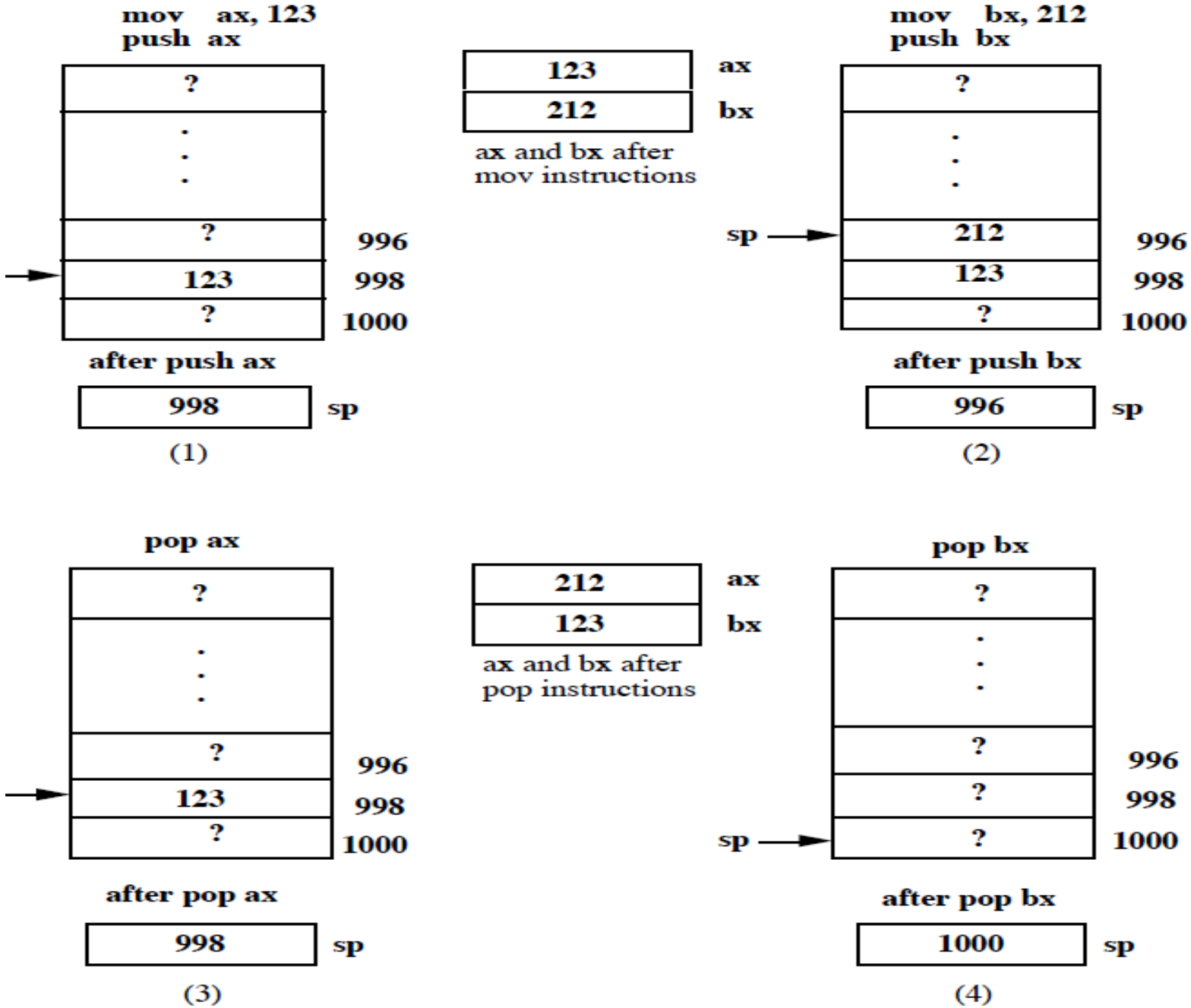
Flags: Only affected by the popf instruction

Addressing mode: src and dst should be words and cannot be immediate. dst cannot be the IP or CS register

PUSH & POP

- The two set of instructions which explicitly modify the stack are the PUSH (which places items on the stack) and the POP (which retrieves items from the stack).
- In both cases, the stack pointer is adjusted accordingly to point always to the top of stack.
- Thus PUSH AX means SP=SP-2 and AX -> [SP]
- POP AX means [SP] -> AX and SP=SP+2.




mov ax, 123
push ax
?
?
123
?
996
998
1000
after push ax
998
sp
(1)
123
212
ax
bx
ax and bx after
mov instructions
mov bx, 212
push bx
?
212
123
?
sp
996
998
1000
after push bx
996
sp
(2)
pop ax
?
?
123
?
996
998
1000
after pop ax
998
sp
(3)
212
123
ax
bx
ax and bx after
pop instructions
pop bx
?
?
?
?
sp
996
998
1000
after pop bx
1000
sp
(4)

- Using the stack, swap the values of the ax and bx registers, so that ax now contains what bx contained and bx contains what ax contained. (This is not the most efficient way to exchange the contents of two variables). To carry out this operation, we need at least one temporary variable:

```
ORG 100h
mov ax, 123h
mov bx, 212h
push ax ; Store ax on stack
push bx ; Store bx on stack
pop ax ; Copy last value on stack to ax
pop bx ; Copy first value to bx
push ax
mov ax, bx
pop bx
ret
```

H.W

1) What errors are present in the following

- mov ax 5e
- mov 42, ax
- mov bx, ch
- move ax, 1h
- add 2, cx
- add 3, 6
- inc bx, 2

2) Write instructions to evaluate the arithmetic expression 6 +(8-2) leaving the result in ax using

(a) 1 register, (b) 2 registers (c) 3 registers

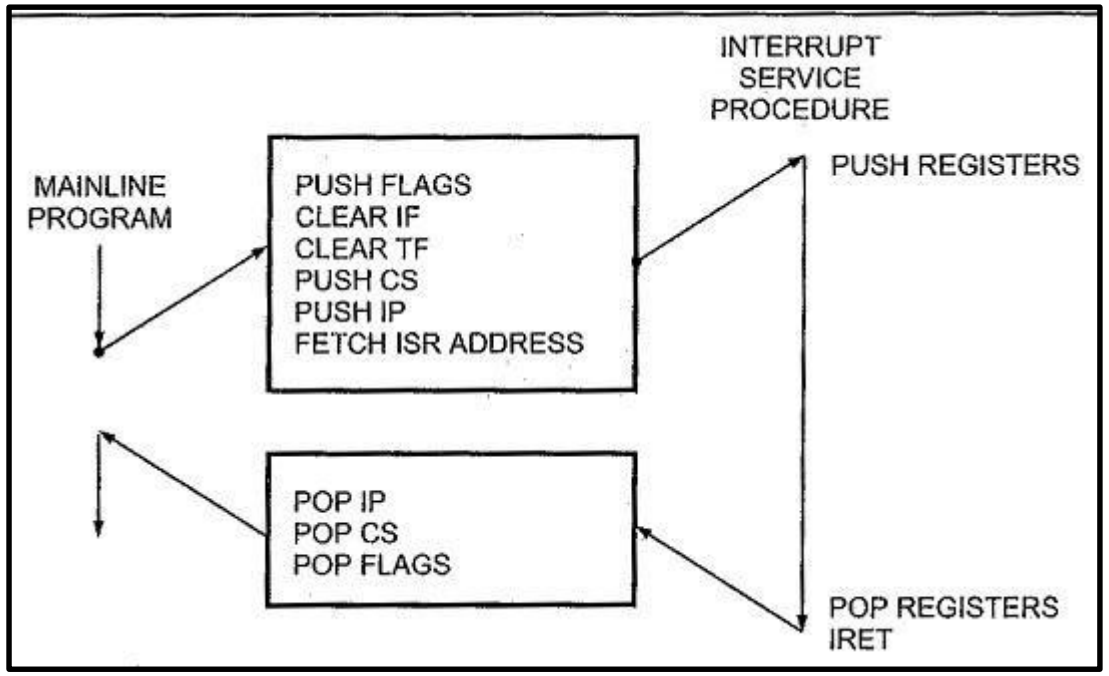
Interrupts of 8086

- An interrupt is either a hardware generated call (externally derived from a hardware signal) or software generated CALL (internally derived from the execution of an instruction or by some other internal event)
- When an interrupt occurs, the CPU is interrupted
- The CPU attends to the interrupt and then carries on where it left off

Types of Interrupts

- Hardware
  - an external signal is applied to the NMI input pin or the INTR input pin
    - NMI - non-maskable interrupt
    - INTR – interrupt used to deal with I/O devices that need attention
- Software
  - Occurs when INT instruction is executed
  - When some error condition is produced by the execution of an instruction such as divide by zero

Interrupt handling in 8086



- When an interrupt is requested…
  - the CPU finishes executing current instruction
  - pushes flag register onto stack
  - disables the INTR input by clearing the IF (interrupt flag) in the flag register
  - clears the TF (trap flag) in the flag register
  - pushes the current CS contents onto the stack
  - pushes the current IP contents onto the stack
  - does an indirect far jump to the start of the ISR (Interrupt Service Routine)

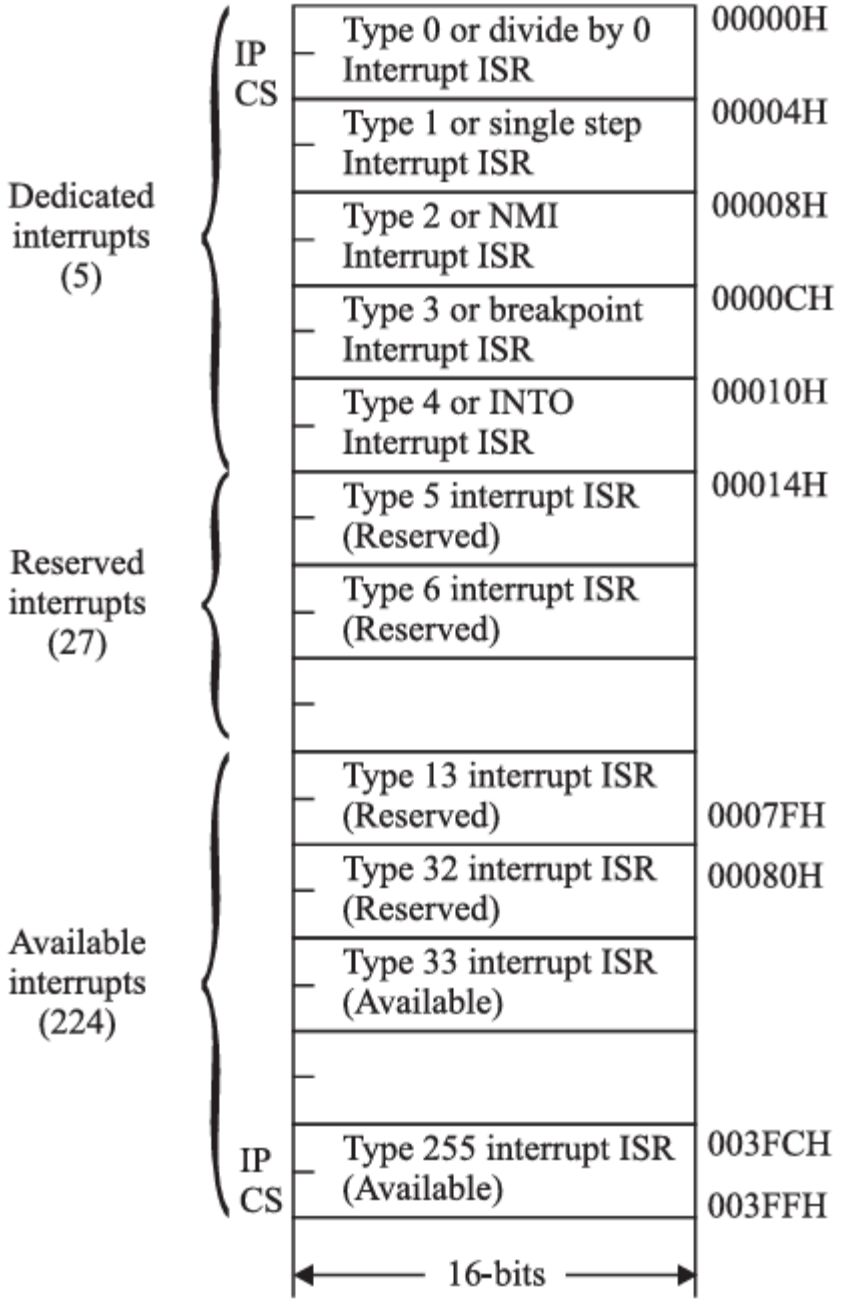
Interrupt Handling (cont.)

- Processor executes Interrupt Service Routine (ISR) like a normal procedure
- Only difference is normal procedures use RET at the end of a procedure, ISRs use an IRET instruction instead of RET
- This pops IP, CS, and the Flag registers
- ISRs can be interrupted!

Interrupt vector table

- In 8086, 1st 1k bytes of memory 00000-003ffh is set aside as a table for storing the starting addresses of interrupt service procedures(routines)
- Since 4 bytes are required to store the CS & IP values for each interrupt service procedure, the table can hold the starting addresses of up to 256 interrupts
- The starting address of an interrupt service procedure is called interrupt vector or interrupt pointer and the table is referred to as interrupt vector table or the interrupt pointer table
- Each double word interrupt vector is identified by a number from 0 to 255.
- Intel calls this number, the type of interrupt

Interrupt vector table contd..



- The lowest 5 interrupts are dedicated to specific (or dedicated) interrupts such as divide by 0, single step, NMI, interrupt caused by an instruction and overflow interrupt.
- Interrupts from 5-31 are reserved by Intel for more complex processors
- The upper 224 interrupts from type 32 to type 255 are available for users to use for hardware or software interrupts
- The vector for each interrupt requires 4 memory locations.
- Therefore, when the 8086 responds to a particular type of interrupt, it automatically multiplies the type by 4 to produce the desired address in the table to get the starting address of the Interrupt Service Procedure (ISP)

Interrupt sources

- An 8086 interrupt can come from any one of the 3 sources

1. External signal to NMI or INTR input pin. An interrupt caused by a signal applied to one of these inputs is referred to as hardware interrupt
2. Execution of interrupt instruction INT. This is referred to as software interrupt
3. An interrupt caused by some error condition in 8086 by the execution of an instruction

Ex: Divide by 0. If an attempt is made to divide an operand by 0, 8086 automatically interrupts the current executing program

Actions taken by 8086 when an interrupt occurs

- At the end of each instruction cycle, 8086 checks to see if any interrupts have been requested. If an interrupt has been requested, 8086 responds to the interrupt by stepping through the following series of major actions.
- It decrements IP by 2 and pushes the flag register on the stack
- disables INTR interrupt by clearing interrupt flag in the flag register
- It resets Trap flag in the flag register
- It decrements SP by 2 and pushes the current CS register on the stack
- It decrements SP again by 2 and pushes the current IP contents on the stack
- It does an indirect far jump to the start of the procedure that is written to respond to the interrupt

H.W: Write a report about the specific (or dedicated) interrupts (interrupts 0-4), with examples.

Interrupt cycle of 8086

- Broadly, there are two types of interrupts
- The first out of them is external interrupt and the second is internal interrupt.
- In external interrupt, an external device or a signal interrupts the processor from outside. i.e. the interrupt is generated outside the processor. For ex: a keyboard interrupt
- The internal interrupt on the other hand is generated internally by the processor circuit or by the execution of interrupt instruction. The examples of this type are divide by 0, overflow interrupt and interrupts due to INT instructions

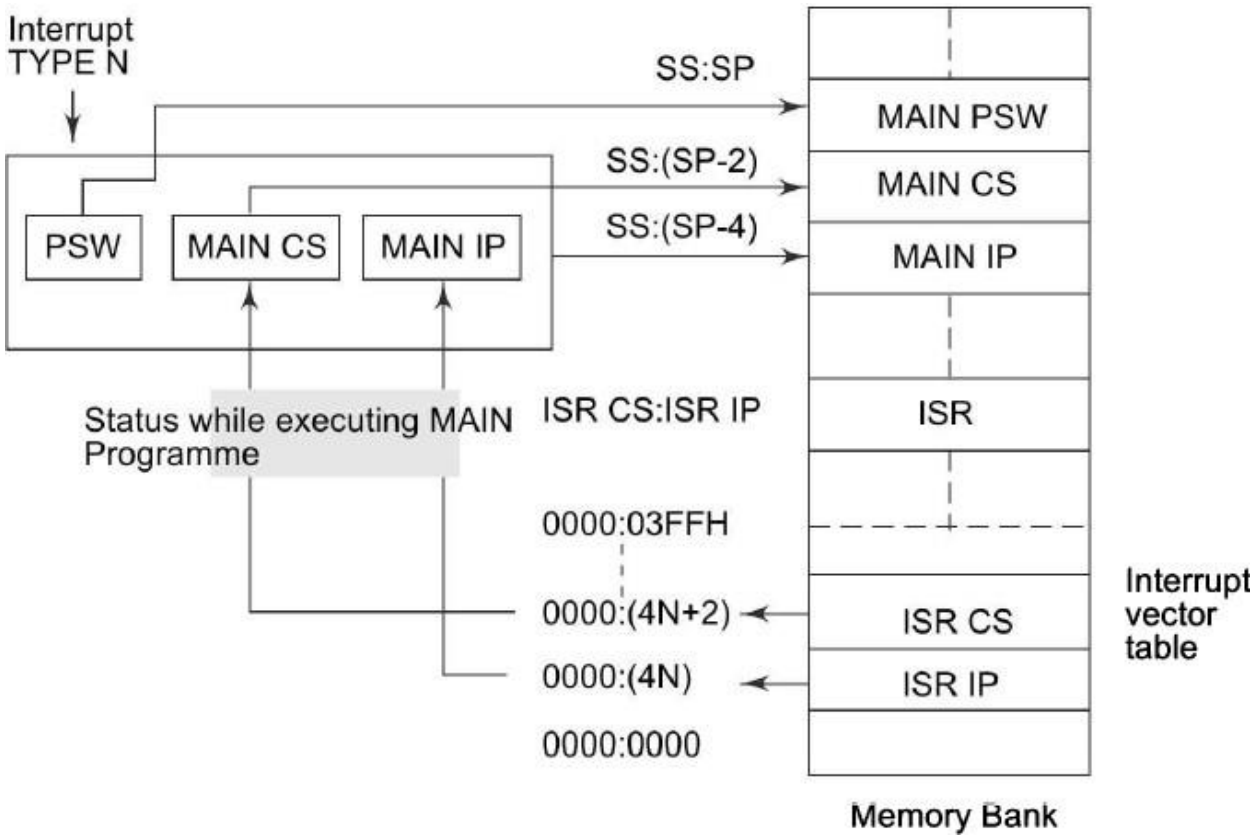
Interrupt cycle of 8086 contd..

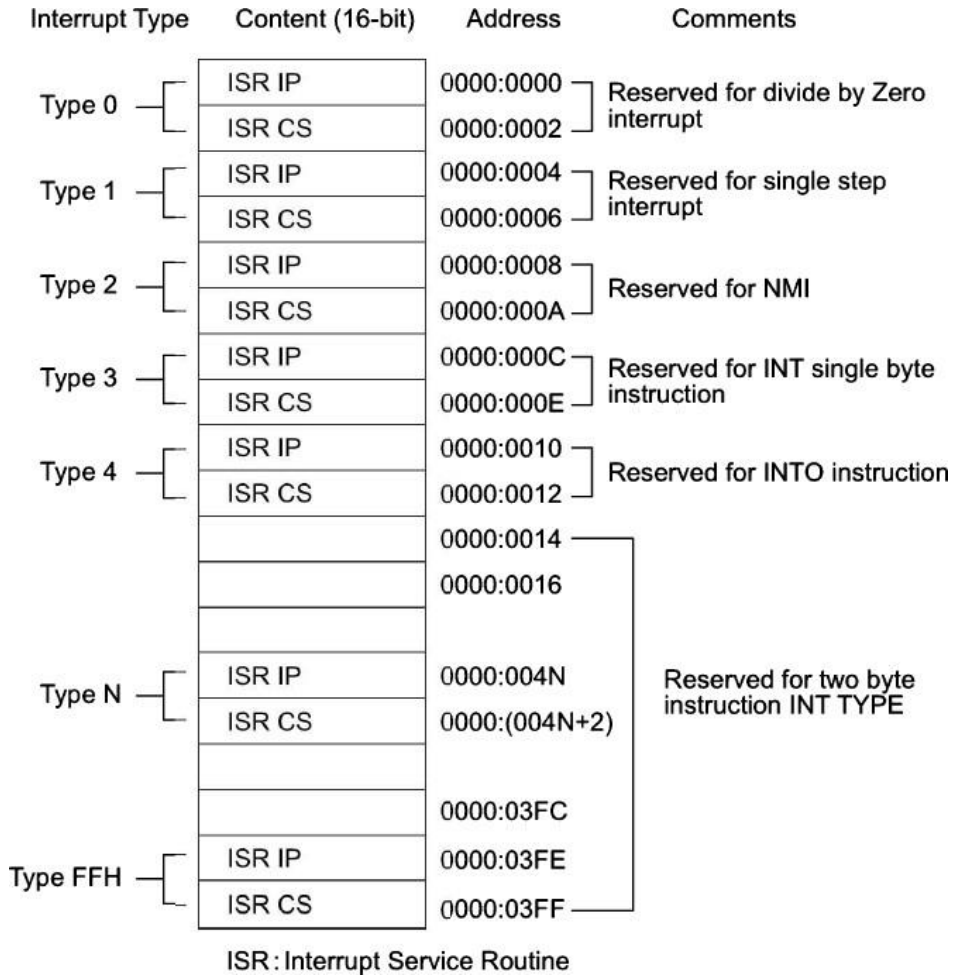
- Suppose an external device interrupts the CPU on interrupt pin, either INTR or NMI, while the CPU is executing an instruction of a program.
- The CPU first completes the execution of the current instruction
- The IP is then incremented to pint to the next instruction
- The CPU then acknowledges the requesting device on its $\overline{INTA}$ pin immediately if it is a NMI, TRAP or Divide by zero interrupt
- If it is an INT request, the CPU checks the IF flag
- IF the IF flag is set, the interrupt request is acknowledged using the $\overline{INTA}$ pin
- If the IF is not set, the interrupt requests are ignored.

Interrupt cycle of 8086 contd..

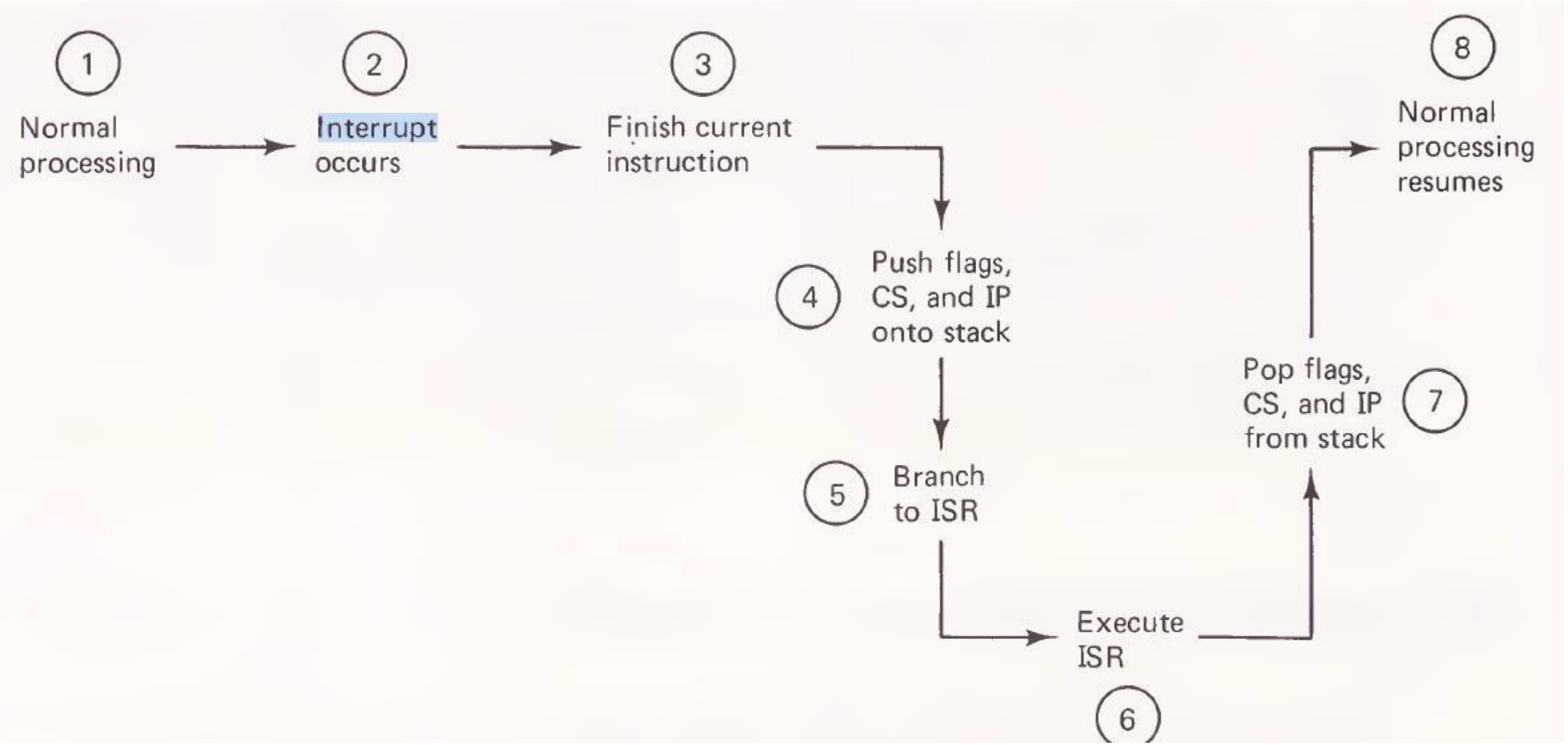
- The responses to the NMI, TRAP or Divide by zero interrupt requests are independent of the IF flag.
- After an interrupt is acknowledged, the CPU computes the vector address from the type of the interrupt and the processor executes the ISR of the corresponding interrupt.
- If further interrupts are to be responded during the time of the first interrupt is being serviced, the IF should be set by the ISR of the first interrupt
- If the interrupt flag IF is not set, the subsequent interrupt signals will not be acknowledged by the processor, till the current one is completed
- The programmable interrupt controller is used for managing such multiple interrupts based on their priorities.
- At the end of ISR, the last instruction should be IRET
- When the CPU executed IRET, the contents of flags, IP and CS which were saved before the execution of ISR are now retrieved to the respective registers.
- The execution continues onwards from this address received by IP and CS

Interrupt Response Sequence and structure of interrupt vector table





Interrupt Response Sequence contd..



NMI (non-maskable) interrupt

- NMI interrupt has the highest priority among the external interrupts
- TRAP (Single Step-Type1) is an internal interrupt having the lowest priority
- The NMI is activated on a positive transition (low to high).
- The assertion of the NMI interrupt is equivalent to an execution of instruction INT 02. I.e. Type 2 interrupt
- However, if an internal interrupt is being serviced and an NMI (or INTR if IF is set) occurs, the ISR for the internal interrupt will be suspended and the external interrupt is honored, even though it is of lower priority.
- The priority structure applies only to simultaneous interrupt requests.

Interrupt	Priority
Divide Error, Int n, Int 0	HIGHEST
NMI	↓
INTR	↓
SINGLE - STEP	LOWEST

NMI interrupt contd..

- When the NMI is activated, the current instruction being executed is completed and then the NMI is served
- In case of string manipulation instructions, this interrupt will be served only after the complete string has been manipulated
- *Another high going edge on the NMI pin, during the period in which the first NMI is served , triggers another response*
- *The signal on the NMI pin must be free of logical bounces to avoid erratic NMI responses*

NMI interrupt contd..

- NMI is a nonmaskable interrupt, which means that it cannot be blocked.
- INTR, on the other hand, is maskable via the IF flag.
- Only when this flag is set, interrupt on this input will be accepted.
- Although internal interrupts have priority over external interrupts, the NMI request will be honored as soon as the internal interrupt's ISR begins The same is not true for the INTR input.
- Because the NMI input is nonmaskable, care must be taken when using this interrupt. This is because there may be some programs that you do not want interrupted—reading or writing data to a disk drive, for example.
- For this reason, the NMI input is normally reserved for catastrophic events such as memory error or an impending power failure.

Example

- A particular interrupt has a type number n = 41H. If the ISR begins at address 09E3:0010H, determine the locations in the vector table to store this address.
- Solution: The vector address is calculated by multiplying 41H by 4. This is done most easily by rotating 41H left twice. 41H = 01000001; rotate left twice --> 10000100 = 104H or 00104H. IP is stored in the low word location and CS in the high word location.
- 00107H: 09H
- 00106H: E3H
- 00105H: 00H
- 00104H: 10H

Interrupts
MS-DOS Function Calls (INT 21h)

- MS-DOS provides a lot of functions for displaying and reading the text on the console (200 functions). The general syntax for calling the functions is :

```
MOV AH, function_number ; input parameters
INT 21h                 ; return values
```

INT 21h / AH=4Ch: Terminate Process Ends the current process (program), returns an optional 8-bit return code to the calling process. A return code of 0 usually indicates successful completion.

```
mov ah, 4Ch ; terminate process
mov al, 0 ; return code
int 21h
```

INT 21h / AH=01h: Read character from standard input, with echo, result is stored in **AL.**

```
mov ah, 01h
int 21h
```

INT 21h / AH=02h : Write Character to Standard Output :

```
mov ah, 02h
mov dl, ’A’
int 21h
```

INT 21h / AH=09 : Write a string at **DS:DX** to Standard Output . String must be terminated by '**$**'.:

```
mov dx, offset msg
mov ah, 09
int 21h
ret
msg db “Hello world $”
```

A Sample Assembly Language Program using FULL SEGMENT DEFINITION

```
STSEG  SEGMENT
    DB   64   DUP (?)
STSEG  ENDS
;-----------------------------------------------
DTSEG SEGMENT
    DATA1   DB   52H
    DATA2   DB   29H
    SUM      DB    ?
DTSEG ENDS
;-----------------------------------------------
CDSEG SEGMENT
MAIN  PROC FAR       ;This is the program entry point
    ASSUME  CS:CDSEG,   DS:DTSEG,  SS:STSEG
    MOV AX,DTSEG    ;load the data segment address
    MOV DS, AX          ;assign value to DS

    MOV AL, DATA1   ;get the first operand
    MOV BL, DATA2   ;get the second operand
    ADD AL, BL          ;add the operands
    MOV SUM, AL      ;store result in location SUM

    MOV AH, 4CH      ;set up to
    INT 21H              ;return to the Operating System (DOS)
MAIN  ENDP
CDSEG ENDS
    END    MAIN        ;this is  the program exit point
```

Program to input characters until ‘q’ and display

```
STSEG  SEGMENT
    DB   64   DUP (?)
STSEG  ENDS
;-----------------------------------------------
DTSEG SEGMENT
     STR   10 DUP('$')
DTSEG ENDS
;-----------------------------------------------
CDSEG SEGMENT
MAIN  PROC FAR        ;This is the program
entry point
     ASSUME  CS:CDSEG,   DS:DTSEG,
SS:STSEG
     MOV AX,DTSEG     ;load the data segment
address
     MOV DS, AX           ;assign value to DS

L2:  MOV AH, 01H
     INT 21H
     CMP AL, 'q'
     JE L1
     JMP L2

 L1:  MOV STR, AL
     MOV AH, 09H
     MOV DX, OFFSET STR
     INT 21H

     MOV AH, 4CH      ;set up to
     INT 21H               ;return to the Operating System (DOS)
MAIN  ENDP
CDSEG ENDS
     END    MAIN         ;this is  the program exit point
```