Lecture # 33

Interrupts and related concepts # 2

Covered by these slides and chapter # 17
(section 17.4)

Interrupts (contd..)

- Treatment of an interrupt-service routine is very similar to that of a subroutine.
- However there are significant differences:
  - A subroutine performs a task that is required by the calling program.
  - Interrupt-service routine may not have anything in common with the program it interrupts.
  - Interrupt-service routine and the program that it interrupts may belong to different users.
  - As a result, before branching to the interrupt-service routine, not only the PC, but other information such as condition code flags, and processor registers used by both the interrupted program and the interrupt service routine must be stored.
  - This will enable the interrupted program to resume execution upon return from interrupt service routine.

Interrupts (contd..)

- Saving and restoring information can be done automatically by the processor or explicitly by program instructions.
- Saving and restoring registers involves memory transfers:
  - Increases the total execution time.
  - Increases the delay between the time an interrupt request is received, and the start of execution of the interrupt-service routine. This delay is called interrupt latency.
- In order to reduce the interrupt latency, most processors save only the minimal amount of information:
  - This minimal amount of information includes Program Counter and processor status registers.
- Any additional information that must be saved, must be saved explicitly by the program instructions at the beginning of the interrupt service routine.

Interrupts (contd..)

- When a processor receives an interrupt-request, it must branch to the interrupt service routine.
- It must also inform the device that it has recognized the interrupt request.
- This can be accomplished in two ways:
  - Some processors have an explicit interrupt-acknowledge control signal for this purpose.
  - In other cases, the data transfer that takes place between the device and the processor can be used to inform the device.

Interrupts (contd..)

- Interrupt-requests interrupt the execution of a program, and may alter the intended sequence of events:
  - Sometimes such alterations may be undesirable, and must not be allowed.
  - For example, the processor may not want to be interrupted by the same device while executing its interrupt-service routine.
- Processors generally provide the ability to enable and disable such interruptions as desired.
- One simple way is to provide machine instructions such as *Interrupt-enable* and *Interrupt-disable* for this purpose.
- To avoid interruption by the same device during the execution of an interrupt service routine:
  - First instruction of an interrupt service routine can be Interrupt-disable.
  - Last instruction of an interrupt service routine can be Interrupt-enable.

Interrupts (contd..)

- Previously, before the processor started executing the interrupt service routine for a device, it disabled the interrupts from the device.
- In general, same arrangement is used when multiple devices can send interrupt requests to the processor.
  - During the execution of an interrupt service routine of device, the processor does not accept interrupt requests from any other device.
  - Since the interrupt service routines are usually short, the delay that this causes is generally acceptable.
- However, for certain devices this delay may not be acceptable.
  - Which devices can be allowed to interrupt a processor when it is executing an interrupt service routine of another device?

Hardware Interrupts

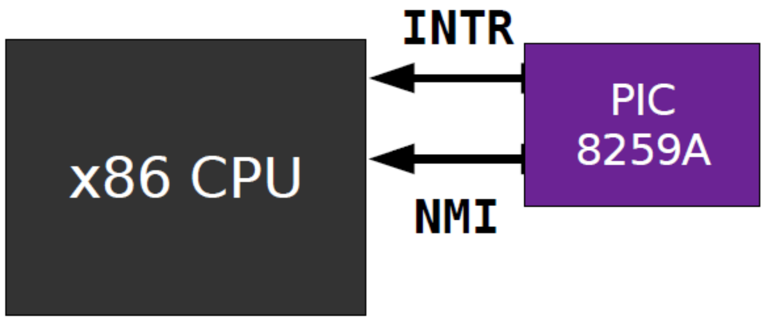
Hardware Interrupt Types:

Non-Maskable Interrupt

- Never ignored

INTR Maskable

- Ignored when IF is 0



PIC: Programmable Interrupt Controller (8259A)

- Has 16 wires to devices (IRQ0 – IRQ15)
- Can be programmed to map IRQ0-15 → vector number

X86 Interrupt Vectors

- Every Exception/Interrupt type is assigned a number:
  - **its vector**

- When an interrupt occurs, the vector determines what code is invoked to handle the interrupt.

- JOS example: vector 14 → page fault handler
  vector 32 → clock handler → scheduler

0	Divide Error
2	Non-Maskable Interrupt
3	Breakpoint Exception
6	Invalid Opcode
11	Segment Not Present
12	Stack-Segment Fault
13	General Protection Fault
14	Page Fault
18	Machine Check
32-255	User Defined Interrupts

Enabling / Disabling Interrupts

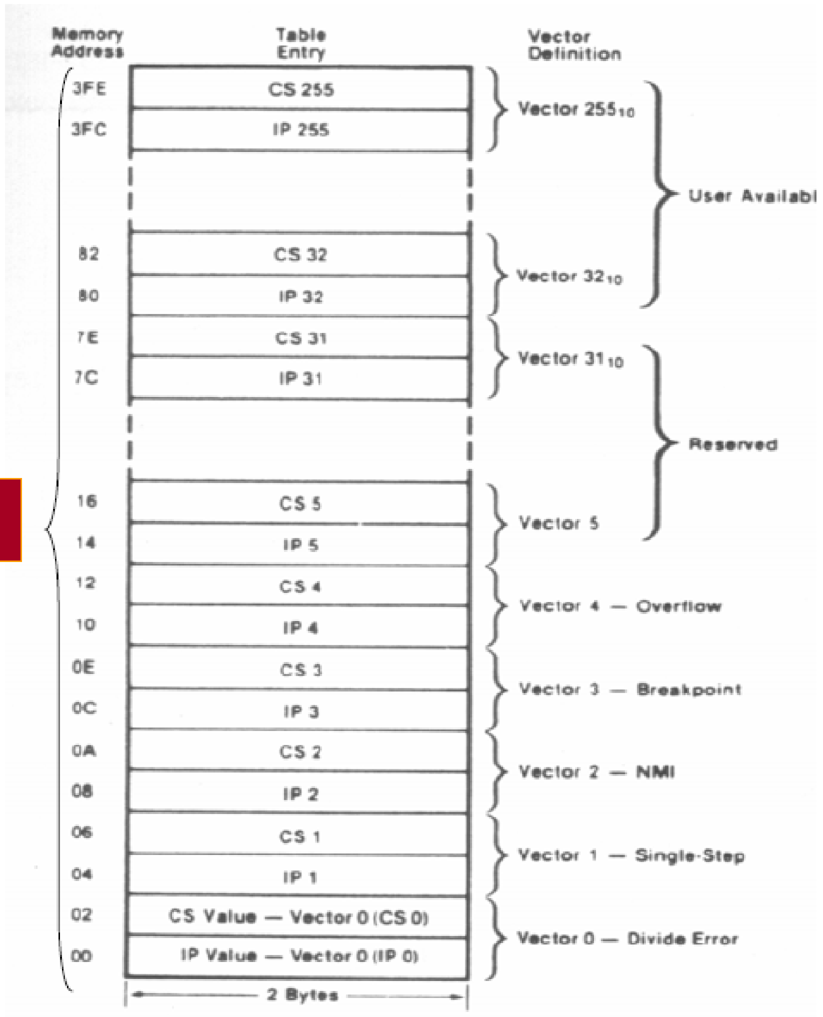
Maskable Hardware Interrupts

- Clearing the IF flag inhibits processing hardware interrupts delivered on the INTR line.
- Use the STI (set interrupt enable flag) and CLI (clear interrupt enable flag) instructions.
- IF affected by: interrupt/task gates, POPF, and IRET.

Non-Maskable Interrupt

- Invoked by NMI line from PIC.
- Always Handled immediately.
- Handler for interrupt vector 2 invoked.
- No other interrupts can execute until NMI is done.

Interrupt Vector Table (IVT)



First 1 K memory

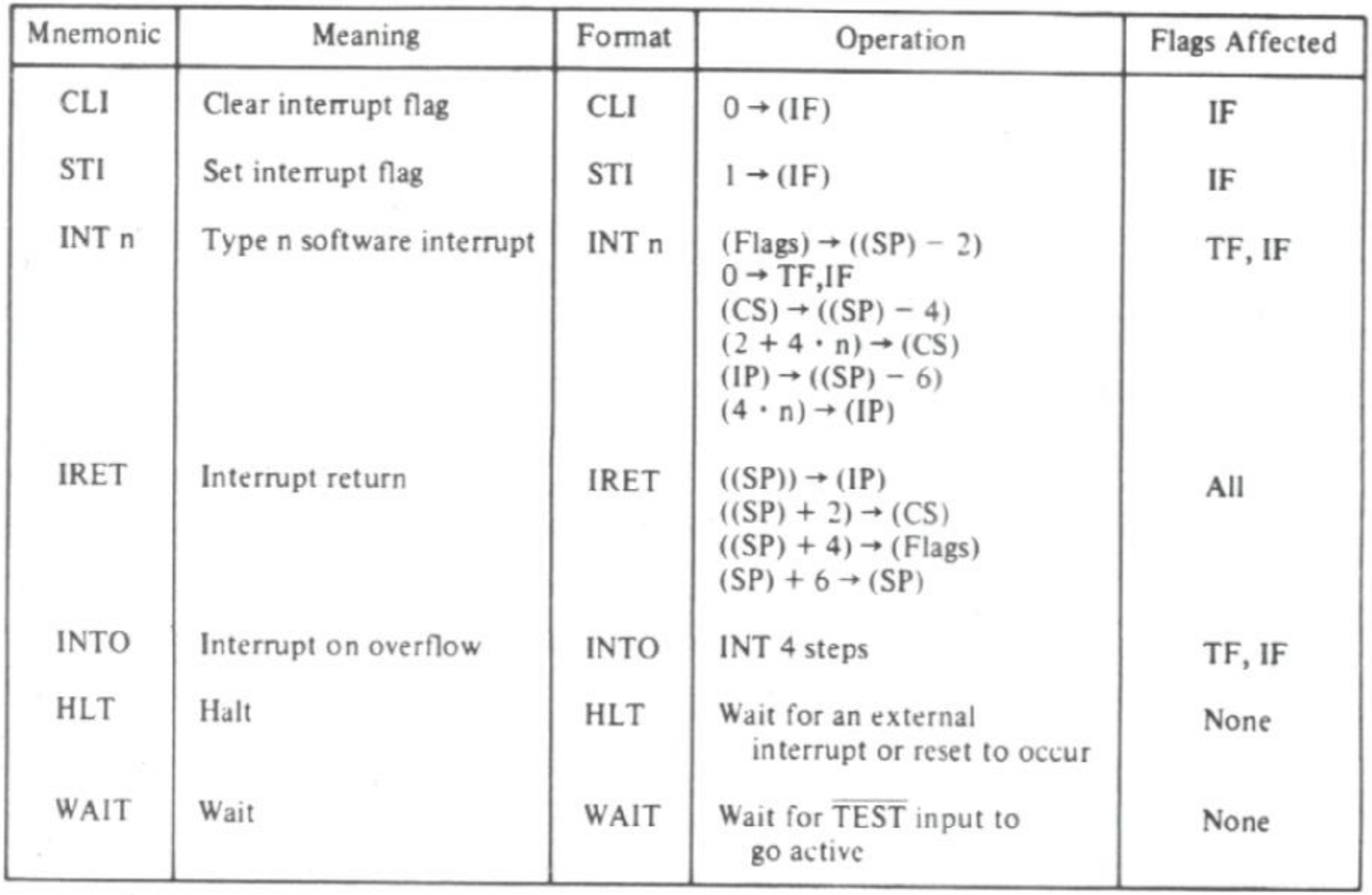
- Contains 256 address pointers (vectors)
- These pointers identify the starting location of their service routines in program memory.
- Held as firmware or loaded as system initialization

6

Interrupt Vector Table (IVT)

- Interrupt vector table consists of 256 entries each containing 4 bytes.
- Each entry contains the offset and the segment address of the interrupt vector each 2 bytes long.
- Table starts at the memory address 00000H.
- First 32 vectors are spared for various microprocessor operations.
- The rest 224 vectors are user definable.
- **The lower the vector number, the higher the priority.**

Interrupt Instructions



Mnemonic	Meaning	Format	Operation	Flags Affected
CLI	Clear interrupt flag	CLI	0 → (IF)	IF
STI	Set interrupt flag	STI	1 → (IF)	IF
INT n	Type n software interrupt	INT n	(Flags) → ((SP) − 2) 0 → TF,IF (CS) → ((SP) − 4) (2 + 4 · n) → (CS) (IP) → ((SP) − 6) (4 · n) → (IP)	TF, IF
IRET	Interrupt return	IRET	((SP)) → (IP) ((SP) + 2) → (CS) ((SP) + 4) → (Flags) (SP) + 6 → (SP)	All
INTO	Interrupt on overflow	INTO	INT 4 steps	TF, IF
HLT	Halt	HLT	Wait for an external interrupt or reset to occur	None
WAIT	Wait	WAIT	Wait for $\overline{\text{TEST}}$ input to go active	None

CALL vs INT

- A CALL FAR instruction can jump any location within the 1 MB address range but INT nn goes to a fixed memory location in the Interrupt Vector Table to get the address of the interrupt service routine

- A CALL FAR instruction is used by the programmer in the sequence of instruction in the program but externally activated hardware interrupt can come at any time

- A CALL FAR cannot be masked but INT nn in hardware can be blocked.

- A CALL FAR saves CS:IP but INT nn saves Flags and CS:IP

- At the end of the subroutine RET is used whereas for Interrupt routine IRET should be the last statement