CSE213

Microcontroller Programming

Interrupts

Introduction

- In this chapter, the coverage of basic I/O and programmable peripheral interfaces is expanded by examining a technique called interrupt-processed I/O.
- An interrupt is a hardware-initiated procedure that interrupts whatever program is currently executing.
- This chapter provides examples and a detailed explanation of the interrupt structure of the Intel family of microprocessors.

Chapter Objectives

Upon completion of this chapter, you will be able to:

- Explain the interrupt structure of the Intel family of microprocessors.
- Explain the operation of software interrupt instructions INT, INTO, IRET
- Explain how the interrupt enable flag bit (IF) modifies the interrupt structure.
- Learn the interrupts supported by the emulator.

INTERRUPT CONCEPT

Example 1: Man in office

- Imagine a man working at his office. During his work, he may be subject to different interrupts.
- He may choose to deal with, postpone or ignore the interrupt.
- For example:
  - A friend comes in and asks for a favor. Since he is busy, tells the friend to come back later.
  - Telephone rings, but he ignores the phone.
  - Some interrupts should not be ignored - fire alarm!
- We need to be able to respond to interrupts properly.

Example 2: Bus driver

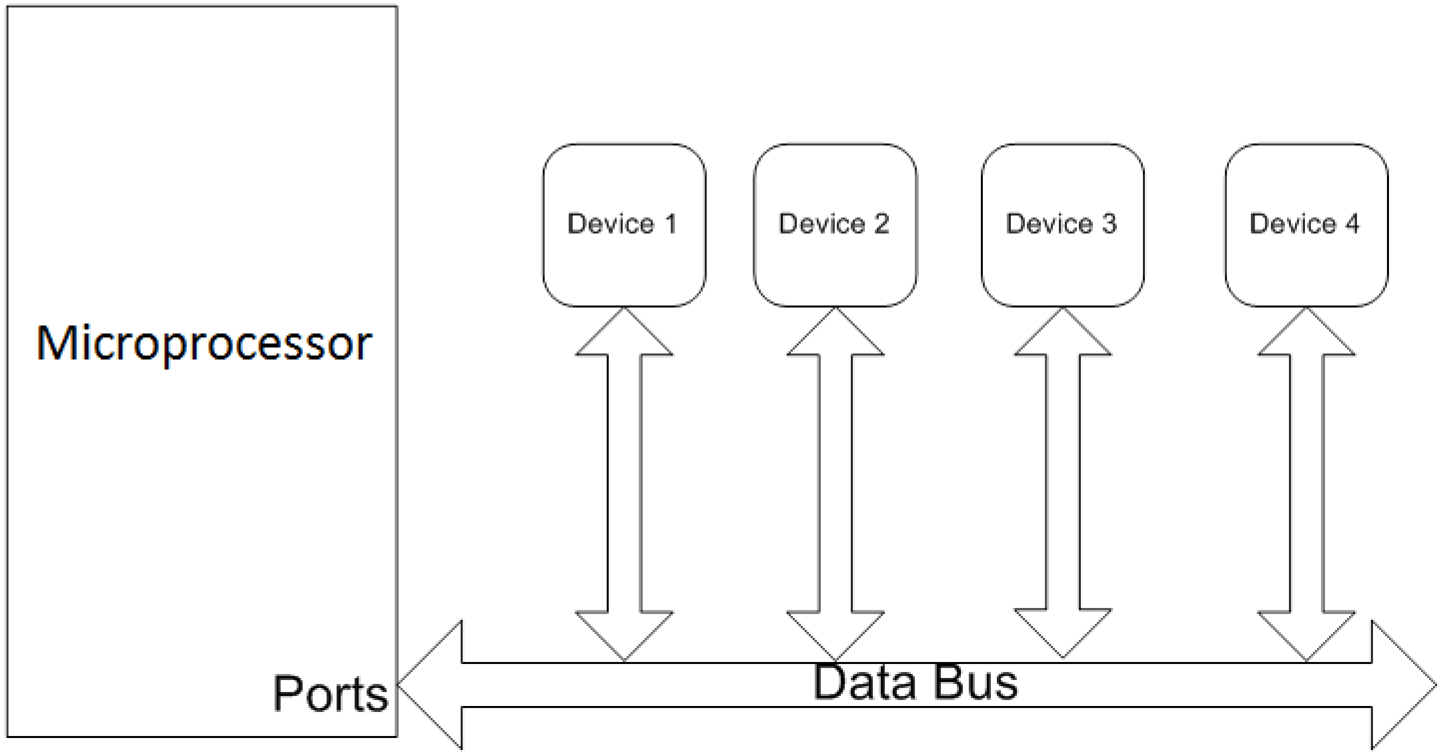
- The bus driver is normally driving the bus.
- If interrupted by any of the passengers who presses the “STOP” button, he will;
    a) Pull over the bus at the next bus stop
    b) Open door
    c) Wait for passenger(s) to leave
    d) Close door
    e) Continue driving
- As a summary, he will initiate a fixed sequence of responses when interrupted, then continue to drive.
- One of the major tasks of a processor is to handle **interrupts**.

INTERRUPTS and I/O

- Synchronisation of tasks is a very important function on a computer.
- Visualise all the components in a typical computer trying to communicate with the processor and the processor trying to communicate with them.
- A fast computer normally has to interact with relatively slow I/O devices, slow because they are mechanical, or because they have to work at a human friendly speeds.
- Two ways of interaction
  - Polling (Handshaking)
  - Interrupt

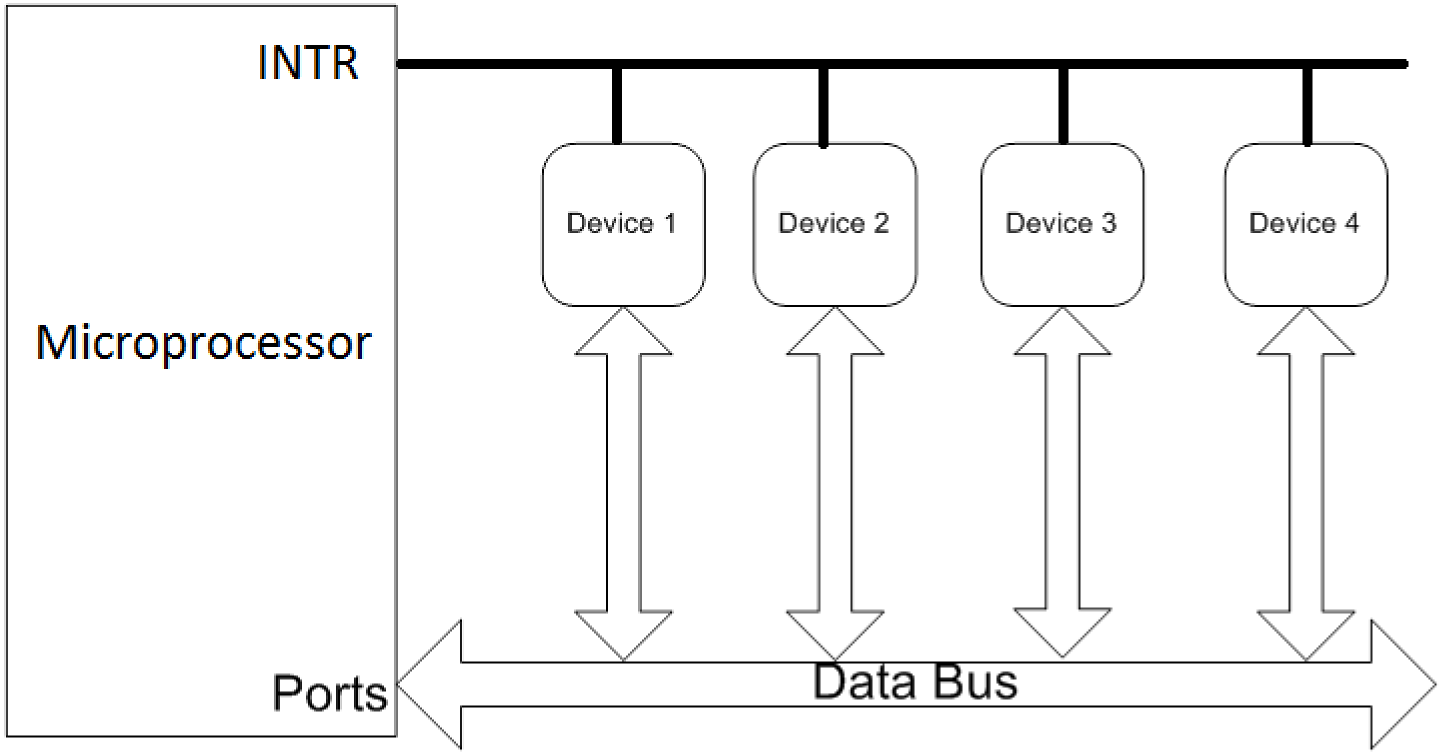
Polling

- “Ask” each device sequentially if it needs service. Note, no devices may need servicing during the poll.



Interrupt

- Device "interrupts" CPU to indicate that it needs service.



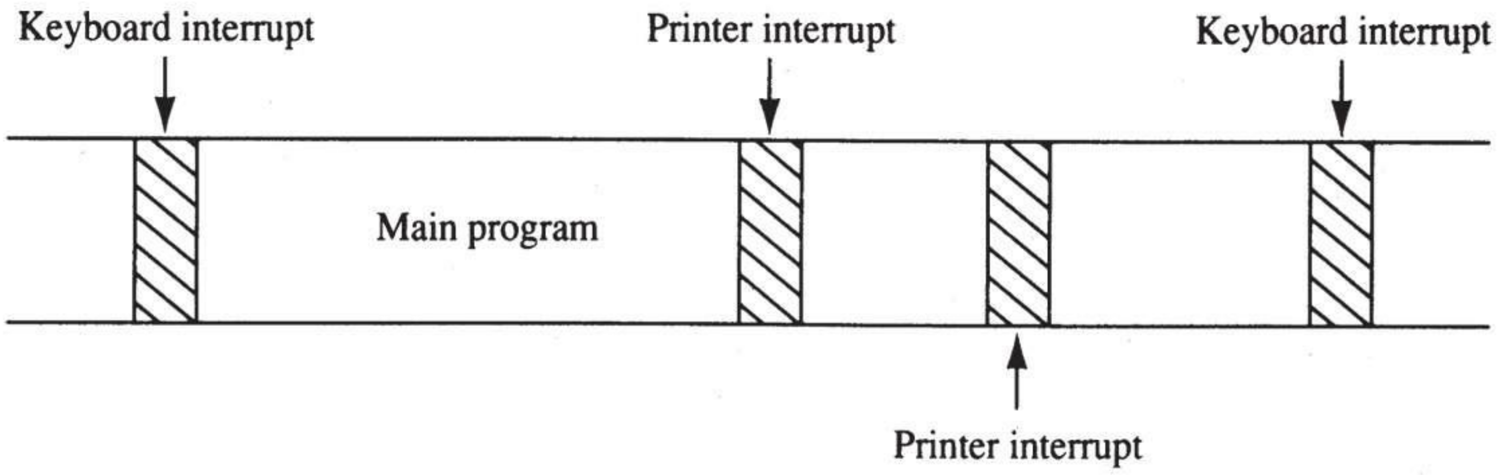
Which method is good?

- So, the computer need not to “freeze” while waiting for an input. It can go off and execute other programs, etc.
- In fact a program that is waiting for an input or output to complete requires very little attention from the CPU. The CPU’s attitude can be summed up as “Get back to me when you have something”.
- This is very important for multi-tasking systems. While one task is waiting for I/O to complete, another can use CPU cycles. And most programs spend most of their time waiting on I/O.
- For example, if you print a file from a Word Processor, transfer of characters to the printer takes place in the background using interrupts. You can still type away (and run other programs).

What needs to interrupt?

- External Devices
  - Keyboard, Mouse, Sensors, etc.
- Internal Events
  - Reset, Illegal Operation, User Interrupt, etc.

Figure 12–1 A time line that indicates interrupt usage in a typical system.



- a time line shows typing on a keyboard, a printer removing data from memory, and a program executing
- the keyboard interrupt service procedure, called by the keyboard interrupt, and the printer interrupt service procedure each take little time to execute

Types of Interrupts

- **Software interrupt** is an interrupt generated within a processor by executing an instruction. Software interrupts are often used to implement system calls.
- **Hardware interrupt** is an interrupt generated by hardware such as I/O devices or peripherals
- **Maskable interrupt**: is a hardware interrupt that may be ignored.
- **Non-maskable interrupt** (NMI) is a hardware interrupt that can never be ignored. NMIs are used for the highest priority tasks such as timers, reset, etc.

Interrupts on Intel

- Intel processors include two hardware pins (INTR and NMI) that request interrupts.
- And one hardware pin ($\overline{\text{INTA}}$) to acknowledge the interrupt requested through INTR.
- The processor also has software interrupts such as INT, INTO
- Flag bit IF (interrupt flag) is also used with the interrupt structure and special return instruction IRET
  - IRETD in the 80386, 80486, or Pentium

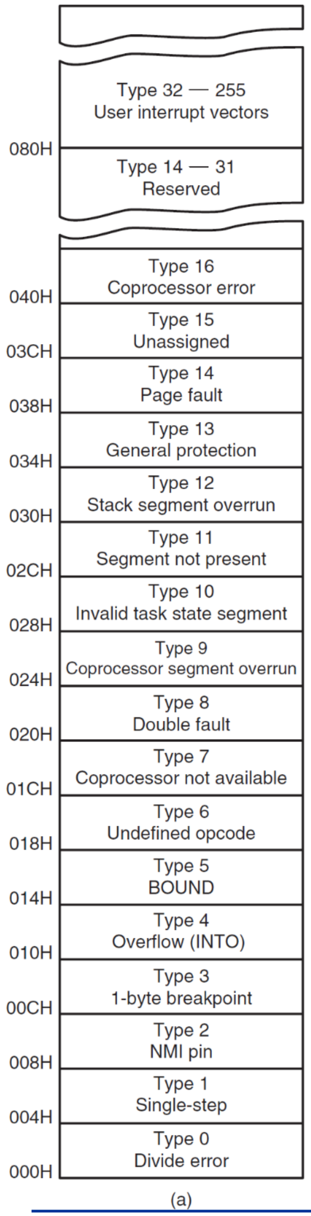
Interrupt Vectors

- Interrupt vectors and the vector table are crucial to an understanding of hardware and software interrupts.
- The **interrupt vector table** is located in the first 1024 bytes of memory at addresses 000000H–0003FFH.
  - contains 256 different four-byte interrupt vectors
- An interrupt vector contains the address (segment and offset) of the interrupt service routine/procedure (ISR).

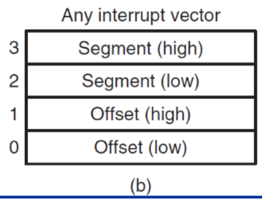
Interrupt Service Routine (ISR)

- ISR is a special subroutine that is designed to “service” the interrupt
  - Also called an “interrupt handler”
- In other words, ISR contains the subroutine that is to be executed when a particular interrupt occurs.

Figure 12–2 (a) The interrupt vector table for the microprocessor and (b) the contents of an interrupt vector.



- the first five interrupt vectors are identical in all Intel processors
- Intel reserves the first 32 interrupt vectors
- the last 224 vectors are user-available
- each is four bytes long in real mode and contains the starting address of the interrupt service procedure.
- the first two bytes contain the offset address
- the last two contain the segment address



Software Interrupt Instructions

- Common software interrupt instructions are available to the microprocessor:
- INT is unconditional
- INTO is conditional.
- IRET is a special interrupt return instruction.

- INTO checks or tests the overflow flag (O).
  - If O = 1, INTO calls the procedure whose address is stored in interrupt vector type 4
  - If O = 0, INTO performs no operation and the next sequential program instruction executes
- The INT *n* instruction calls the interrupt service procedure at the address represented in vector number *n*.
- The IRET instruction is a special return instruction used to return for both software and hardware interrupts.
  - much like a far RET, it retrieves the return address from the stack

Operation of a Real Mode Interrupt

- When the processor completes executing the current instruction, it determines whether an interrupt is active by checking:
  - (1) instruction executions
  - (2) single-step
  - (3) NMI
  - (4) coprocessor segment overrun
  - (5) INTR
  - (6) INT instructions in the order presented

- If one or more are present:
  - 1. Flag register contents are pushed on the stack.
  - 2. Interrupt flag is cleared, disabling the INTR pin.
  - 3. Contents of the code segment register (CS) are pushed onto the stack.
  - 4. Contents of the instruction pointer (IP) are pushed onto the stack.
  - 5. Interrupt vector contents are fetched and placed into IP and CS so the next instruction executes at the interrupt service procedure addressed by the vector.
  - 6. When ISR is done, previous status is restored from the stack.

Interrupt Flag Bits

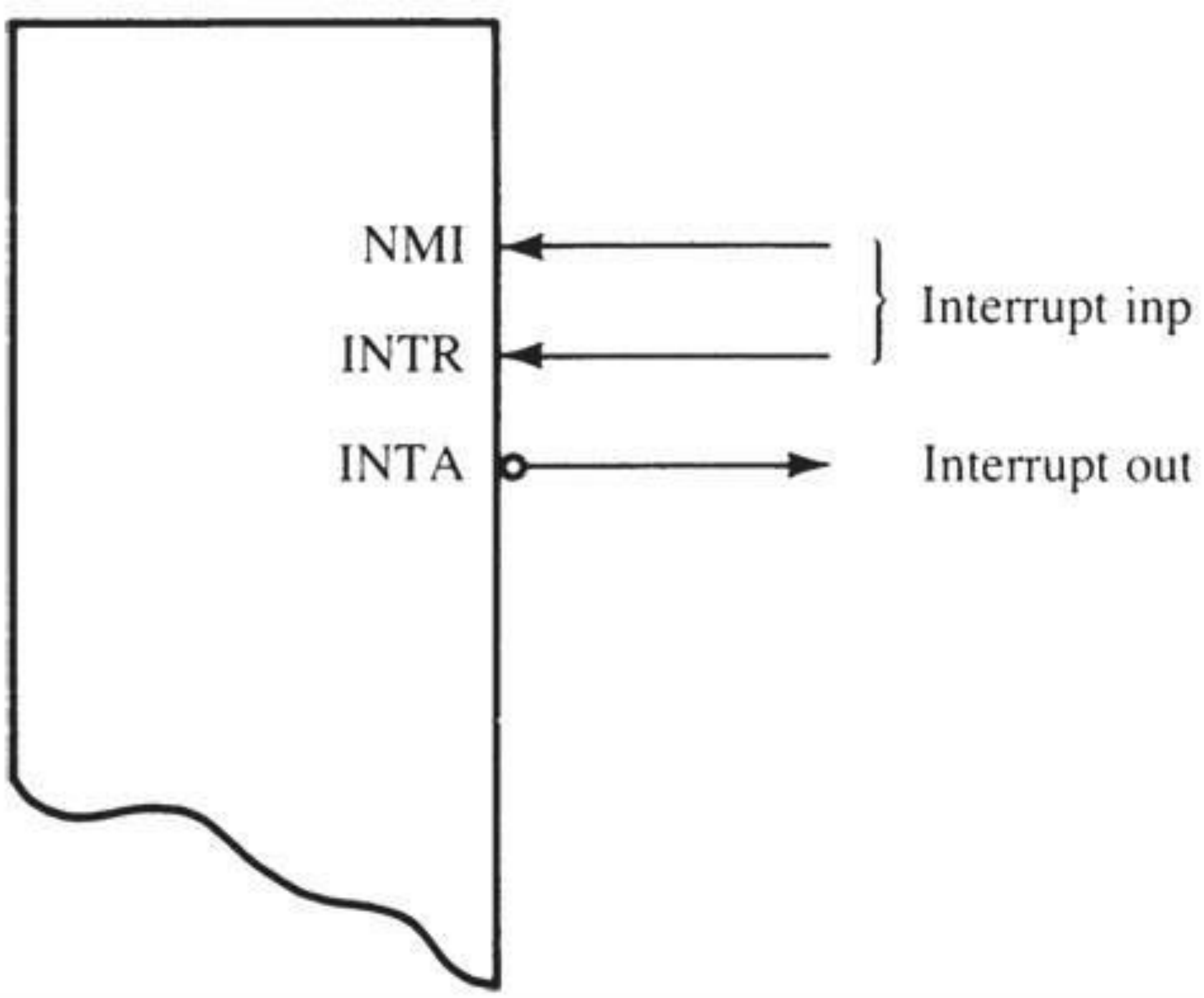
- The interrupt flag (IF) is cleared after the contents of the flag register are stacked during an interrupt.
- When IF is set, it *allows* the INTR pin to cause an interrupt
- When IF is cleared, it *prevents* the INTR pin from causing an interrupt

Hardware Interrupts

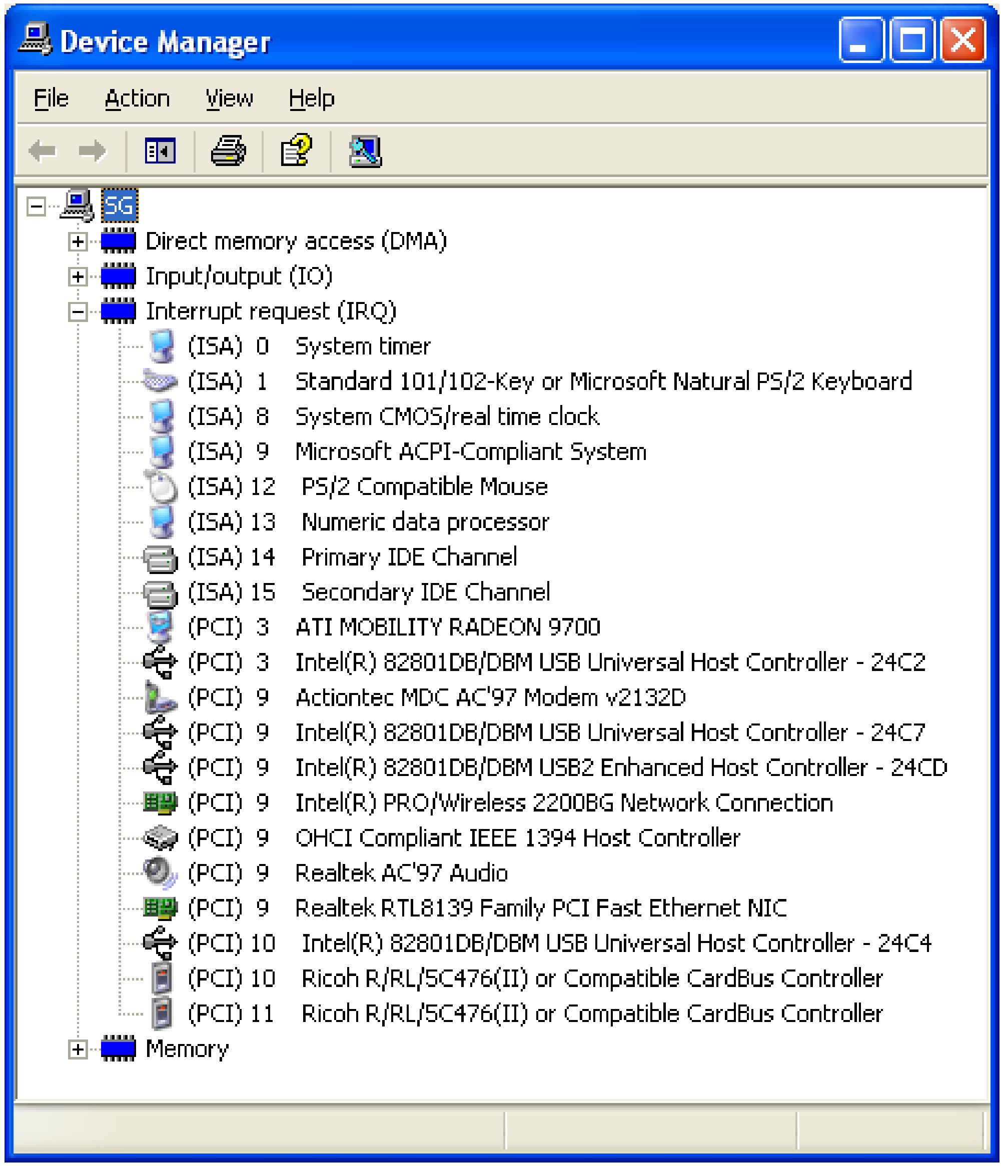
- The two hardware interrupt inputs:
  - non-maskable interrupt (NMI)
  - interrupt request (INTR)
- When NMI input is activated, a type 2 interrupt occurs
  - because NMI is internally decoded
- The INTR input must be externally decoded to select a vector.

- Any interrupt vector can be chosen for the INTR pin, but we usually use an interrupt type number between 20H and FFH.
- Intel has reserved interrupts 00H - 1FH for internal and future expansion.
- $\overline{\text{INTA}}$ is also an interrupt pin on the processor.
  - it is an output used in response to INTR input to apply a vector type number to the data bus connections D_7–D_0
- Figure 12–5 shows the three user interrupt connections on the microprocessor.

Figure 12–5 The interrupt pins on all versions of the Intel microprocessor.



Interrupts on a Personal Computer



```
SG
  Direct memory access (DMA)
  Input/output (IO)
  Interrupt request (IRQ)
    (ISA)  0   System timer
    (ISA)  1   Standard 101/102-Key or Microsoft Natural PS/2 Keyboard
    (ISA)  8   System CMOS/real time clock
    (ISA)  9   Microsoft ACPI-Compliant System
    (ISA) 12   PS/2 Compatible Mouse
    (ISA) 13   Numeric data processor
    (ISA) 14   Primary IDE Channel
    (ISA) 15   Secondary IDE Channel
    (PCI)  3   ATI MOBILITY RADEON 9700
    (PCI)  3   Intel(R) 82801DB/DBM USB Universal Host Controller - 24C2
    (PCI)  9   Actiontec MDC AC'97 Modem v2132D
    (PCI)  9   Intel(R) 82801DB/DBM USB Universal Host Controller - 24C7
    (PCI)  9   Intel(R) 82801DB/DBM USB2 Enhanced Host Controller - 24CD
    (PCI)  9   Intel(R) PRO/Wireless 2200BG Network Connection
    (PCI)  9   OHCI Compliant IEEE 1394 Host Controller
    (PCI)  9   Realtek AC'97 Audio
    (PCI)  9   Realtek RTL8139 Family PCI Fast Ethernet NIC
    (PCI) 10   Intel(R) 82801DB/DBM USB Universal Host Controller - 24C4
    (PCI) 10   Ricoh R/RL/5C476(II) or Compatible CardBus Controller
    (PCI) 11   Ricoh R/RL/5C476(II) or Compatible CardBus Controller
  Memory
```

Interrupts on Emulator

- There are many useful software interrupts that can be used with emulator.
- **The instruction INT value,** where **value** can be a number between 0 to 255 (or 0 to 0FFh) is used.
- List of supported interrupts and their descriptions are available within the emulator documentation.

Example: INT 10h

- **INT 10h** sub-function **0Eh** is used to print a single character to the screen.
- It takes the character to be printed from the **AL** register.
- It doesn’t change the value of the **AH** register.

```
ORG    100h     ; instruct compiler to make simple single segment .com file.

; The sub-function that we are using
; does not modify the AH register on
; return, so we may set it only once.

MOV    AH, 0Eh    ; select sub-function.

; INT 10h / 0Eh sub-function
; receives an ASCII code of the
; character that will be printed
; in AL register.

MOV    AL, 'H'    ; ASCII code: 72
INT    10h        ; print it!

MOV    AL, 'e'    ; ASCII code: 101
INT    10h        ; print it!

MOV    AL, 'l'    ; ASCII code: 108
INT    10h        ; print it!

MOV    AL, 'l'    ; ASCII code: 108
INT    10h        ; print it!

MOV    AL, 'o'    ; ASCII code: 111
INT    10h        ; print it!

MOV    AL, '!'    ; ASCII code: 33
INT    10h        ; print it!

RET               ; returns to operating system.
```

Set Cursor Position

- **INT 10h / AH = 2**
- Inputs:
  - DH = row
  - DL = column
  - BH = page number (0..7)
- Example:

```
mov dh, 10
mov dl, 20
mov bh, 0
mov ah, 2
int 10h
```

Get Cursor Position and Size

- **INT 10h / AH = 03h**
- Inputs:
  - BH = page number.
- Returns:
  - DH = row.
  - DL = column.
  - CH = cursor start line.
  - CL = cursor bottom line.

Read Character and Attribute

- **INT 10h / AH = 08h** - read character and attribute at cursor position.
- Input:
  - **BH** = page number.
- Returns:
  - **AH** = attribute.
  - **AL** = character.

Write Character and Attribute

- **INT 10h / AH = 09h** - write character and attribute at cursor position.
- Inputs:
  - **AL** = character to display.
  - **BH** = page number.
  - **BL** = attribute.
  - **CX** = number of times to write character.

Attributes

- Character attribute is 8 bit value, low 4 bits set fore color, high 4 bits set background color.

HEX	BIN	Color	HEX	BIN	Color
0	0000	Black	8	1000	Dark Gray
1	0001	Blue	9	1001	Light Blue
2	0010	Green	A	1010	Light Green
3	0011	Cyan	B	1011	Light Cyan
4	0100	Red	C	1100	Light Red
5	0101	Magenta	D	1101	Light Magenta
6	0110	Brown	E	1110	Yellow
7	0111	Light Gray	F	1111	White

Write Character

- **INT 10h / AH = 0Ah** - write character only at cursor position.
- Inputs:
  - **AL** = character to display.
  - **BH** = page number.
  - **CX** = number of times to write character.