# SYSC3601
# Microprocessor Systems

## Unit 7:
## Interrupts

# Topics/Reading

1. Interrupt sources (software/hardware)
2. μP response to interrupts
3. 8259A – Programmable Interrupt Controller
4. 8253 – Programmable Interval Timer
5. 8279 – Programmable Keyboard/Display Interface
6. Peripheral device interfacing for I/O

Reading: Chapter 12, sections 1-4

Chapter 11, section 4 (and 5 for 6th ed)

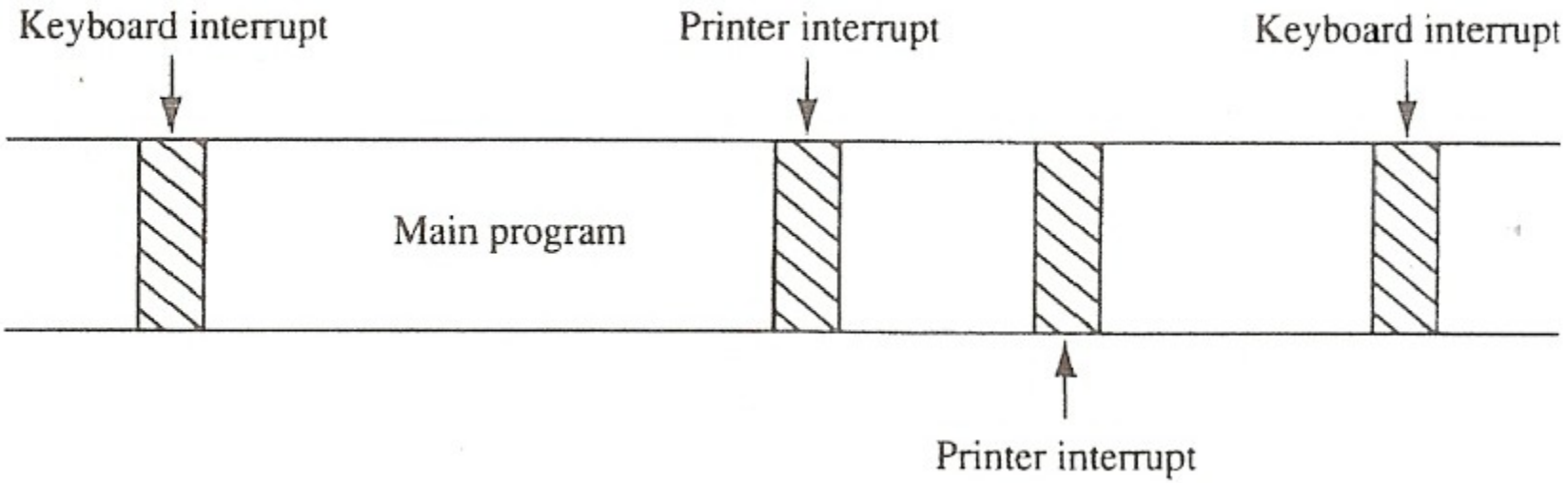
Orange Book: 8259A, 8253, 8279

# Interrupts

- What is an interrupt? A hardware initiated subroutine call. This gives the hardware access to software without any interaction until the hardware needs software to perform some function.
  - Software-initiated interrupts are also possible
- An interrupt service procedure (ISP) is the subroutine called by the hardware interrupt.

# Interrupts

- The hardware in a personal computer and most embedded systems use the interrupt structure extensively.



# Interrupt Sources

- Two types of interrupts:

1. Software interrupts – caused by instructions
   - `INT` *n* – specify interrupt type (32-255)
     - INT 3 is special 1-byte case (useful for debugging)
   - `INTO` – interrupt on overflow ('O' flag bit)
   - `BOUND` – specify upper & lower bounds
   - Divide by zero error

2. Hardware interrupts – caused by μP pins
   - `INTR` – interrupt pin
   - `NMI` – non-maskable interrupt

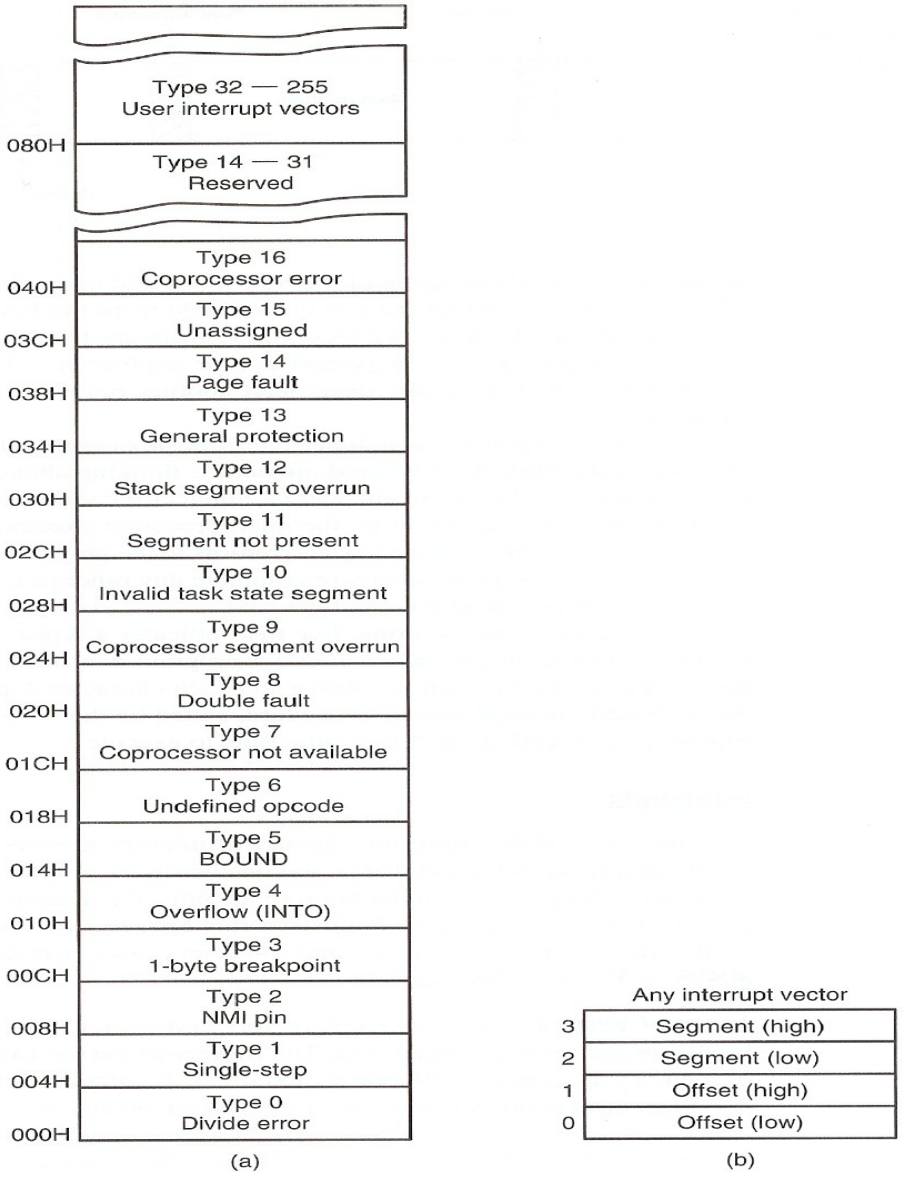
# Interrupt Response Sequence

- Each time the μP completes execution of an instruction, it will check the status of **`NMI`** and **`INTR`**.
- if either is active, or if the next instruction is **`INTO`**, **`INT`** ***`n`***, or **`BOUND`**, then:
    1. Push flag register onto stack.
    2. Clear `IF` and `TF` (interrupt enable and trap flags). Interrupts are now disabled.
    3. Push `CS` then `IP` on stack.
    4. Fetch the *interrupt vector* (discussed shortly)
    5. Proceed to the ISR; flush the instruction queue
- The final statement of an interrupt service routine (ISR) is **`IRET`** – it pops `IP`, `CS` and `Flags`.

# Interrupt Vector Table

- Located in first 1K of memory (00000-003FF).
- Contains 256, 4-byte interrupt vectors.
- Each interrupt vector contains the address (segment and offset) of the service routine.
- Each entry in the vector table is represented by an integer between 0 and 255, called the *interrupt type*.

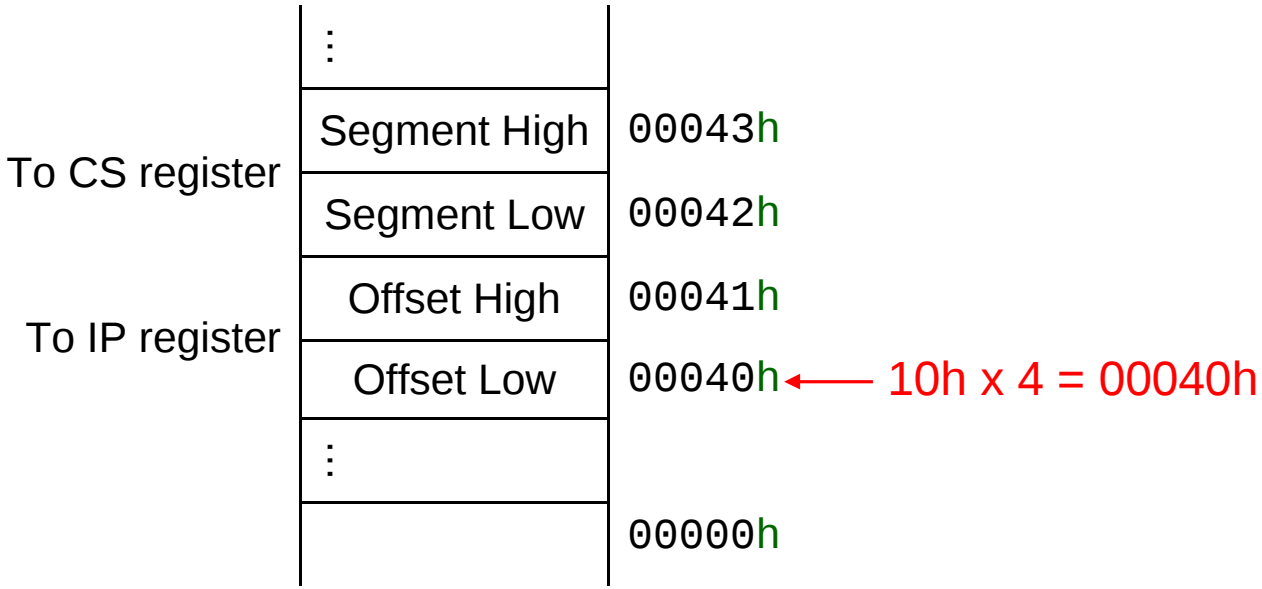
# Interrupt Vector Table



# Interrupt Example

`INT 10h`

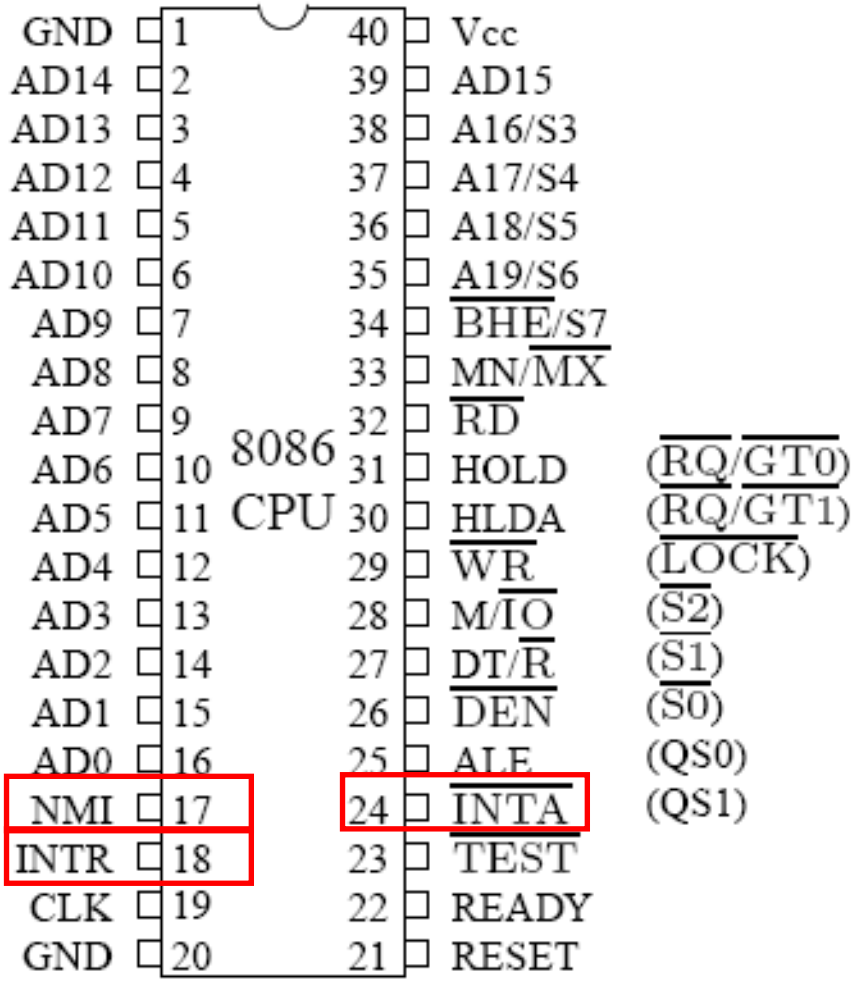
- Interrupt type is 10h(16 decimal)

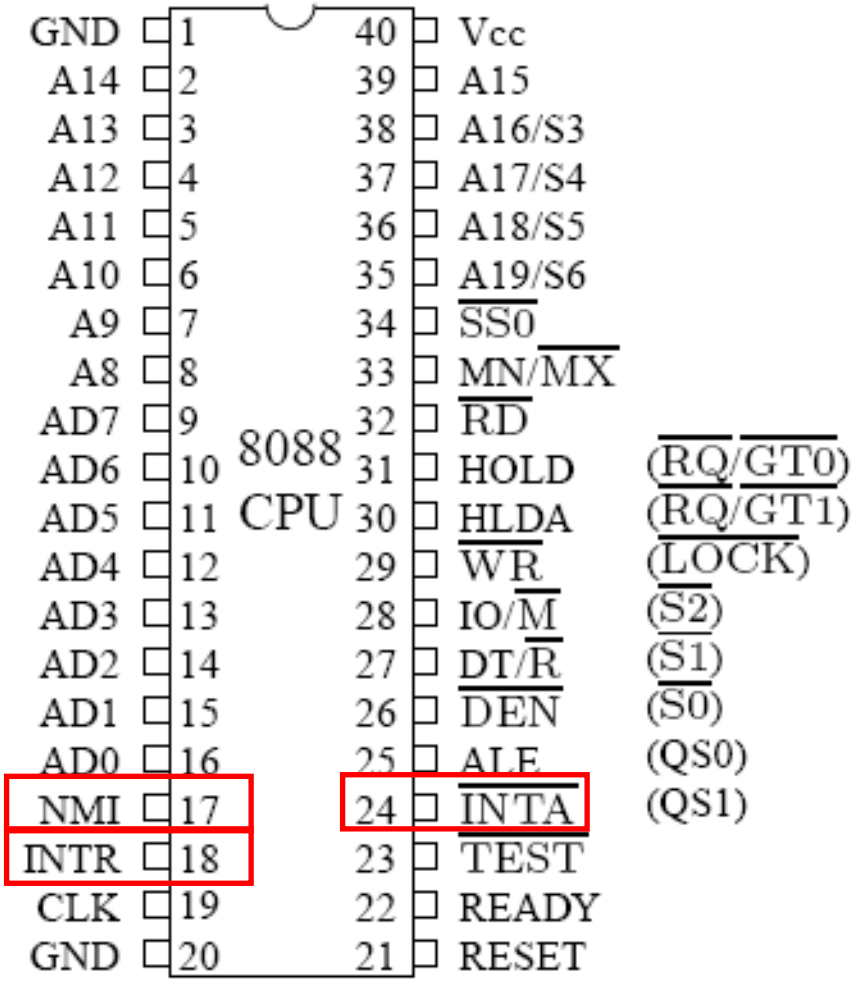



- Starting address for vector is type x 4 (or type << 2)
- First 32 interrupt types are reserved by Intel – the remaining 32-255 are available for the user.

# Hardware Interrupts

- 3 Hardware interrupt pins
  - Non-maskable interrupt (NMI) (input)
  - Interrupt request (INTR) (input)
  - Interrupt acknowledge (INTA) (output)



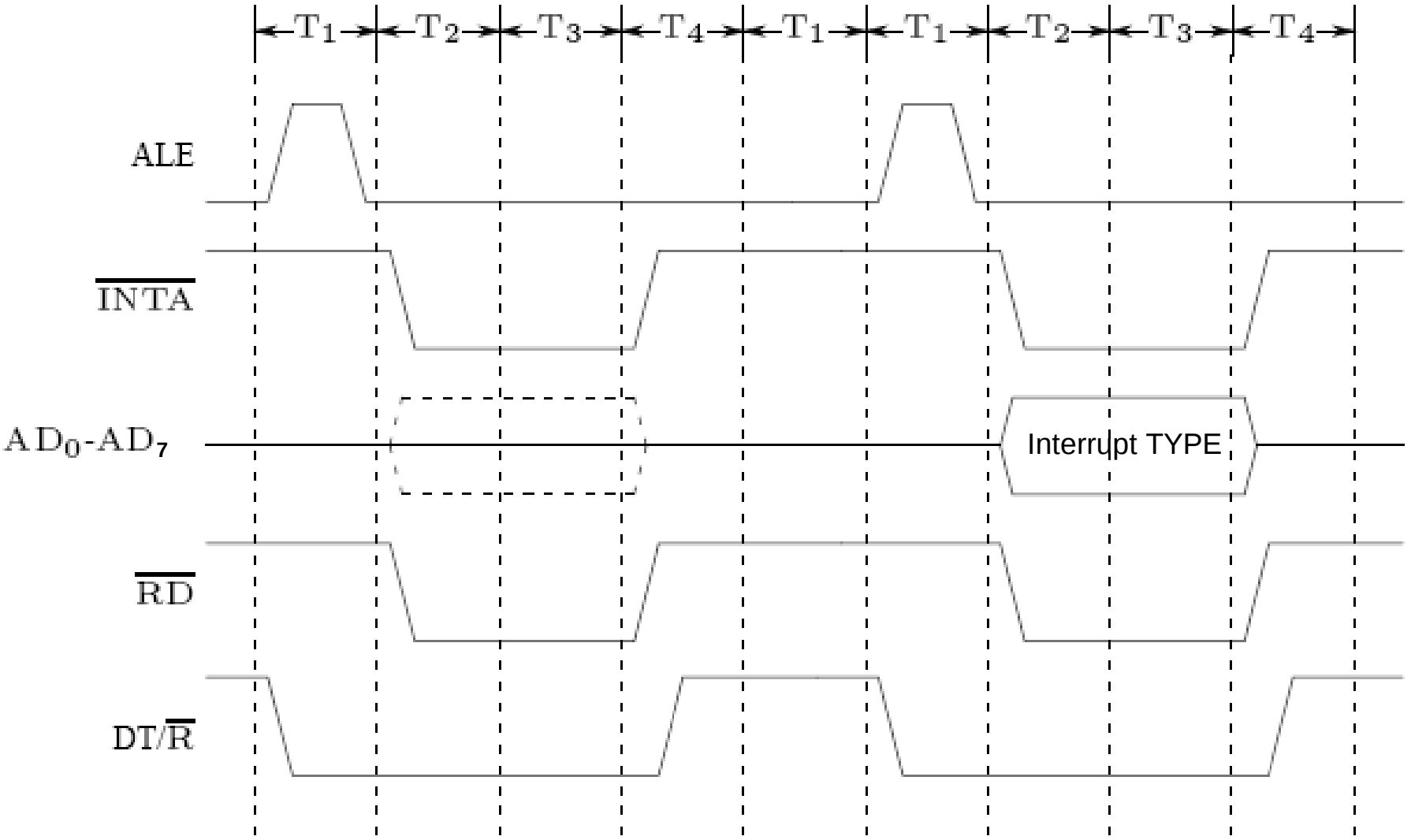
# Hardware Interrupts

- **NMI** Non Maskable Interrupt pin
  - edge triggered.
  - must be logic zero for two clock periods before positive edge.
  - must remain held by external device at logic 1 until $\overline{\textbf{INTA}}$ goes low.
  - Often used for parity errors, power failures and other major faults.

# Hardware Interrupts

- **`INTR`** Interrupt pin
  - level sensitive
  - must remain held by external device at logic 1 until $\overline{\text{INTA}}$ goes low
  - usually reset within the interrupt service routine (ISR)
    - eg: reading data from 82C55 resets `INTR` within 82C55
  - Notes:
    1. ISR is entered with `IF=0` (interrupt flag), i.e. interrupts disabled.
    2. further interrupts may be enabled within/during the ISR using **`STI`**.
    3. **MUST** ensure that INTR is brought low prior to execution of **`STI`**. Otherwise, infinite loop.

# µP response to hardware interrupts

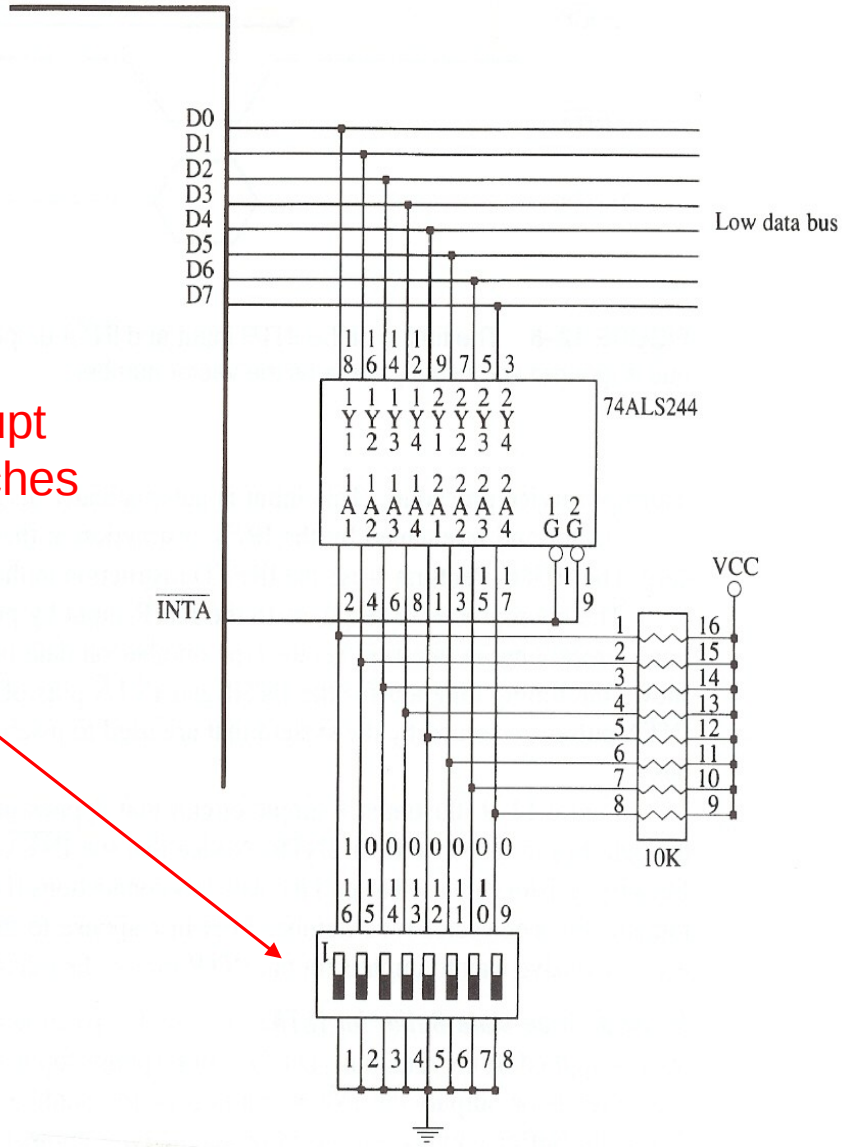


# µP response to hardware interrupts

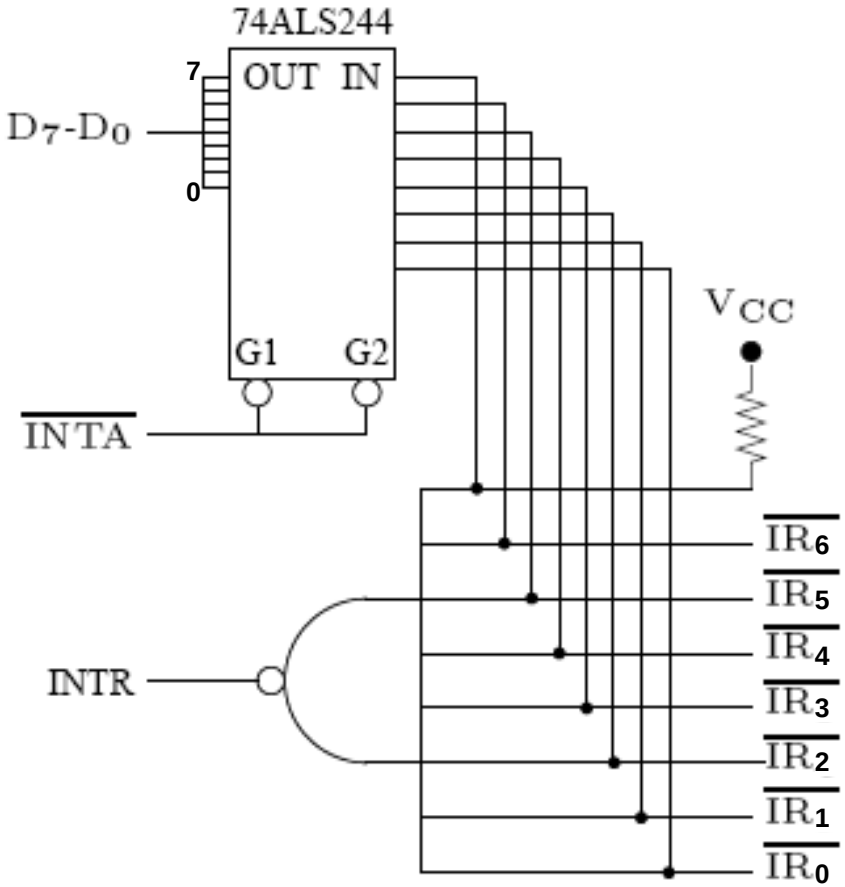
- The response to an INTR is *two* $\overline{\text{INTA}}$ bus cycles separated by two idle clock cycles.
- No address is provided by the 8086, but ALE is generated which will load the address latches with unknown data.
- First $\overline{\text{INTA}}$ cycle signals devices to prepare to present the **TYPE** number on the next $\overline{\text{INTA}}$ (CPU does not capture info on the first $\overline{\text{INTA}}$).
- During the second $\overline{\text{INTA}}$, the device causing the interrupt places a byte on D7-D0 which represents the interrupt TYPE.

# µP response to hardware interrupts

User can set interrupt type using dip switches here.



# µP response to hardware interrupts




In this case, the interrupt types are:

| | | |
|---|---|---|
| $IR_0$ = FEH | 1111 | 1110 |
| $IR_1$ = FDH | 1111 | 1101 |
| $IR_2$ = FBH | 1111 | 1011 |
| $IR_3$ = F7H | 1111 | 0111 |
| $IR_4$ = EFH | 1110 | 1111 |
| $IR_5$ = DFH | 1101 | 1111 |
| $IR_6$ = BFH | 1011 | 1111 |

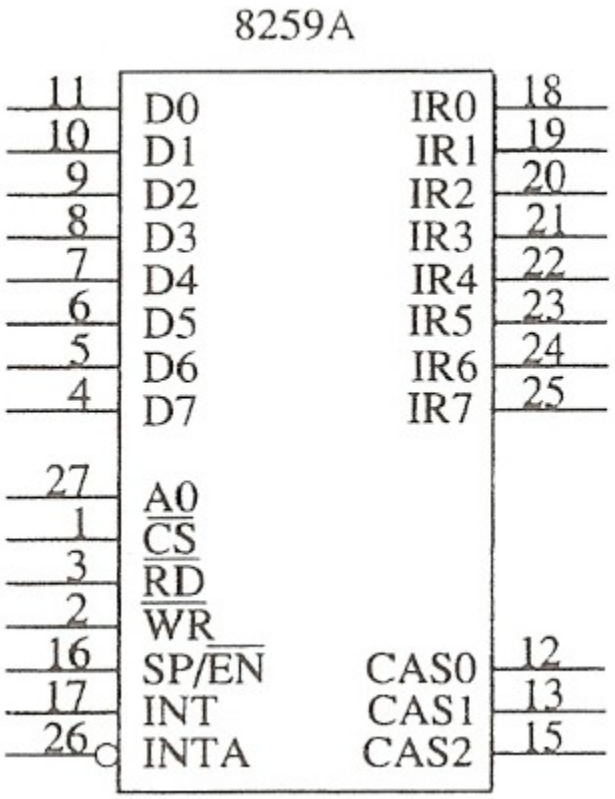
Figure 3: Servicing seven interrupt levels.

# µP response to hardware interrupts

- $D_7$ tied high to ensure interrupt type ≥ 128
- What happened if two IR lines go low?
- Ex: `IR0` and `IR1` both go low, type is `FCH`.
  - Priority is resolved in the vector table.
  - If `IR0` has the higher priority, then the vector for `IR0`'s ISR is stored at FCH.
  - The top half of the vector table (128) vectors must be used to resolve all combinations ($2^7$) of the seven interrupt request lines.
- 8259A resolves priorities for us… (next)

# 8259A Programmable Interrupt Controller

- 8 vectored priority encoded interrupts.
- Can be expanded using one 8259A master and up to eight 8259A slaves to give 64 interrupt levels.
- The 8259A:
  1. accepts requests from peripheral devices
  2. determines which request has the highest priority
  3. issues an interrupt request to the µP
  4. responds with an interrupt type



# 8259A Programmable Interrupt Controller

- Major functional blocks:
  1. Interrupt request register (IRR): stores interrupt levels requesting service.
  2. Priority resolver: selects highest priority from IRR.
  3. In service requestor (ISR): store interrupt level currently being serviced.
  4. Interrupt mask register (IMR): operates on IRR to "mask out" those levels which are disabled.
  5. Cascade buffer: used for master-slave.
     - master notifies slave through CAS0-CAS2 lines.
     - Slave will output its interrupt vector on data bus during the $\overline{\texttt{INTA}}$ pulse.

# 8259A Programmable Interrupt Controller

- Interrupt sequence
  1. One or more IRQ lines goes high → sets IRR bits.
  2. 8259 evaluates and sends `INTR` to μP.
  3. μP sends first $\overline{\texttt{INTA}}$ pulse to 8259 to acknowledge.
  4. 8259 sets highest priority ISR bit and resets corresponding IRR bit.
  5. μP sends second $\overline{\texttt{INTA}}$ pulse, 8259 sends an 8-bit pointer (the interrupt type) onto the data bus.

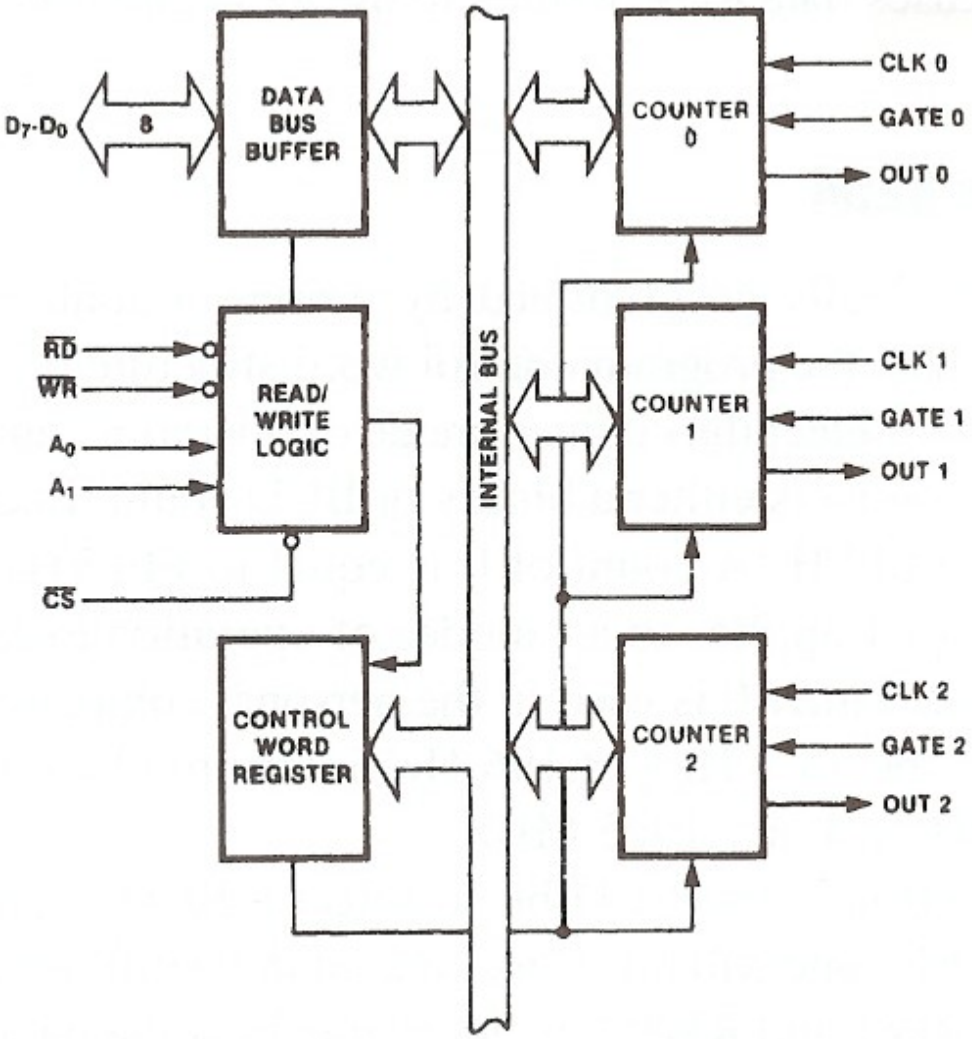
# 8259A Programmable Interrupt Controller

- The interrupt type for the eight IRQ levels are:

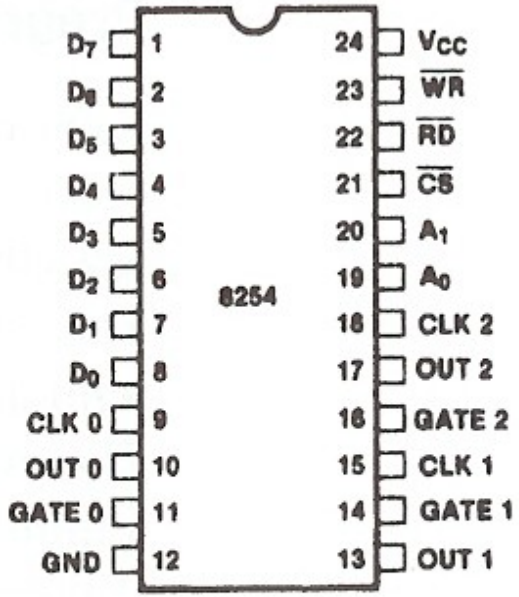
| $T_7$ | $T_6$ | $T_5$ | $T_4$ | $T_3$ | X | X | X |
|---|---|---|---|---|---|---|---|
| Set in ICW2 | | | | | IRQ | | |

- Programming:
  - there are a number of steps needed.
  - Initialization Commands words (from 3 to 4)
  - Operation command words (3 types).
  - Will see examples in lab 3.
    - Students are responsible for assigned readings in pre-lab.

# 8253 Programmable Interval Timer




(a)




(b)

# 8253 Programmable Interval Timer

- Generates accurate time delays under software control.
- Instead of software timing loops, simply configure the 8253 and initialize a counter
  - the 8253 will count out the delay and interrupt the CPU
- Three counters are present:

| $A_1$ | $A_0$ | Function |
|---|---|---|
| 0 | 0 | Counter 0 |
| 0 | 1 | Counter 1 |
| 1 | 0 | Counter 2 |
| 1 | 1 | Control word register |

  - can read or write counter, control word (mode) is write-only.

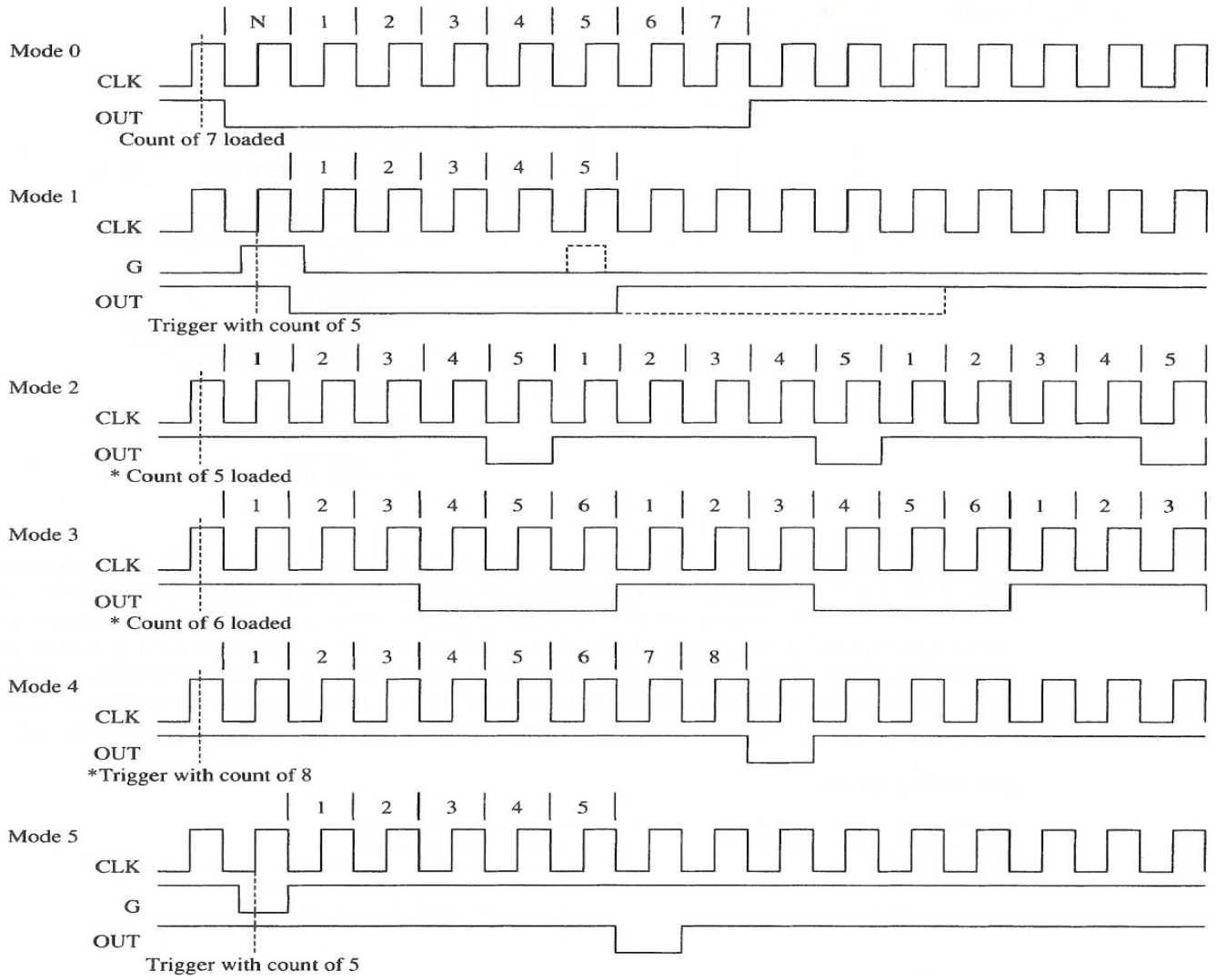
# 8253 Programmable Interval Timer

- Separate control words can be written (all to $A_1A_0$=11) to initialize each counter.
- Six modes are available for each counter:
  - **Mode 0:** Interrupt on terminal count.
    - load counter, count down, INTR μP, stop.
  - **Mode 1:** Programmable one shot.
    - Output goes low on rising edge of gate input.
    - Output goes high on terminal count.
  - **Mode 2:** Rate Generator
    - One low pulse, repeated at each terminal count.

# 8253 Programmable Interval Timer

- **Mode 3:** Square Wave generator.
  - Output is high for ½ count then low for the second half.
  - Repeated.
- **Mode 4:** Software triggered strobe.
  - Load count, output is high.
  - on terminal count, output goes low for one clock, then high again.
- **Mode 5** Hardware-triggered strobe.
  - Output is high, counter starts on rising edge of gate input.
  - Output goes low on terminal count.

# 8253 Programmable Interval Timer



# 8253 Programmable Interval Timer

- Programming is simple:
  - Control word for each counter to be used.
  - Load (write) or read counters as needed.
  - Servicing of ISR's for any INTRs generated by system

# 8279 Programmable Keyboard/Display Interface

- Two sections:
  1. Keyboard
     - scanned interface to 64-contact key matrix.
     - Keyboard entries are debounced and strobed into an 8-character FIFO.
     - Key entries send INTR to the μP.
  2. Display
     - scanned display interface.
     - number and alphanumeric segment (LED) displays.
     - 16 x 8 display RAM which can be arranged as dual 16 x 4.

# 8279 Programmable Keyboard/Display Interface

- Functional Blocks:
  1. I/O control and Data Buffers:
     - $\overline{CS}$, $A_0$, $\overline{RD}$, $\overline{WR}$
     - $A_0$ = 1: Command ($\overline{WR}$) or status ($\overline{RD}$)
     - $A_0$ = 0: Data ($\overline{WR}$ or $\overline{RD}$)
  2. Control and timing registers
     - Store keyboard and display modes.
     - Control programmed with $A_0$ = 1, WR = 0, and data.
     - Timing information is used for keyboard scan and display refresh.
  3. Scan Counter
     - Provides additional timing information for keyboard scan.

# 8279 Programmable Keyboard/Display Interface

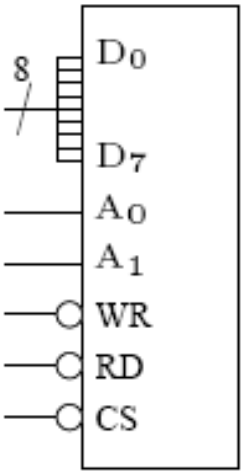
4. Keyboard debounce and control
   - 8 returns lines buffered and latched.
   - These lines are scanned to detect key closure in that row.
   - if detect a switch closure, wait for 10mS see if the switch remains closed (debounce)
5. FIFO RAM and status
   - 8 x 8 RAM.
   - Each new key press is written to FIFO queue.
   - Status keeps track of number of elements in the queue.
   - Status read with $A_0 = 1$.
6. Display address registers and display RAM.
   - address registers hold the internal address of the word currently
   - being written or read by the CPU.
   - these address locations correspond to display positions.

# Peripheral Device Interfacing for I/O

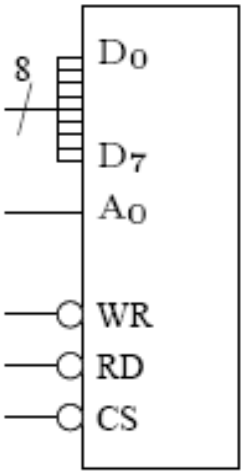
- All peripherals we have seen have similar interfacing requirements.
- Some of these have additional requirements (RESET, CLK) and often require interfacing to provide interrupts (usually to a PIC).
- Each device has a number of internal “ports” or “registers” which may be written to or read from.
  - The internal address for these ports is determined by the address pins.
  - The port addresses used by the μP to transfer commands, status, or data depends on address decoding and the interface.
  - Address decoding is used to select the device through the $\overline{CS}$ pin, and to enable the correct port/register through the $A_n$ lines.

# Peripheral Device Interfacing for I/O

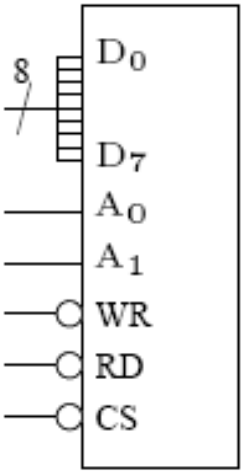
8255 (PPI)




8259 (PIC)




8253 (PIT)




8279