Introduction to Computing Systems
from bits & gates to C & beyond

# Chapter 8

## Input/Output

- *I/O basics*
- *Keyboard input*
- *Monitor output*
- *Interrupt driven I/O*
- *DMA*

# I/O Basics

- **Definitions**
  - Input
    - *transfer data from the outside world to the computer: keyboard, mouse, scanner, bar-code reader, etc.*
  - Output
    - *transfer data from the computer to the outside: monitor, printer, LED display, etc.*
  - Peripheral: any I/O device, including disks.
- **LC-3 supports only a keyboard and a monitor**





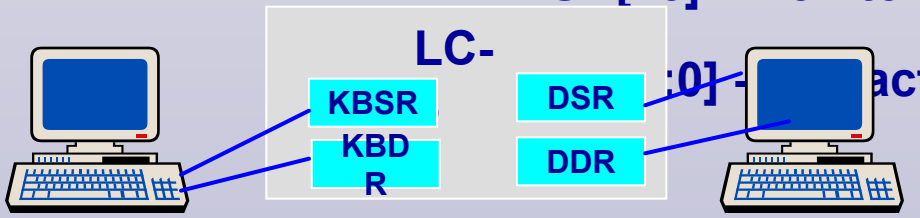
Slides prepared by Walid A. Najjar & Brian J. Linard, University of California, Riverside

# Device Registers

- **I/O Interface**
  - **Through a set of Device Registers:**
    - ***Status register (device is busy/idle/error)***
    - ***Data register (data to be moved to/from device)***
  - **The device registers have to be read/written by the CPU.**

- **LC-3**
  - ***KBDR: keyboard data register***
  - ***KBSR: keyboard status register***
  - ***DDR: display data register***
  - ***DSR: display status register***

- **KBSR[15] - keyboard ready (new character available)**
- **KBDR[7:0] - character typed (ASCII)**
- **DSR[15] - monitor ready**
- **DDR[7:0] - character to be displayed**








Slides prepared by Walid A. Najjar & Brian J. Linard, University of California, Riverside

# Addressing Device Registers

- **Special I/O Instructions**
  - Read or write to device registers using specialized I/O instructions.
- **Memory Mapped I/O**
  - Use existing data movement instructions (Load & Store).
  - Map each device register to a memory address (fixed).
  - CPU communicates with the device registers as if they were memory locations.
- **LC-3**
  - Uses memory mapped I/O:

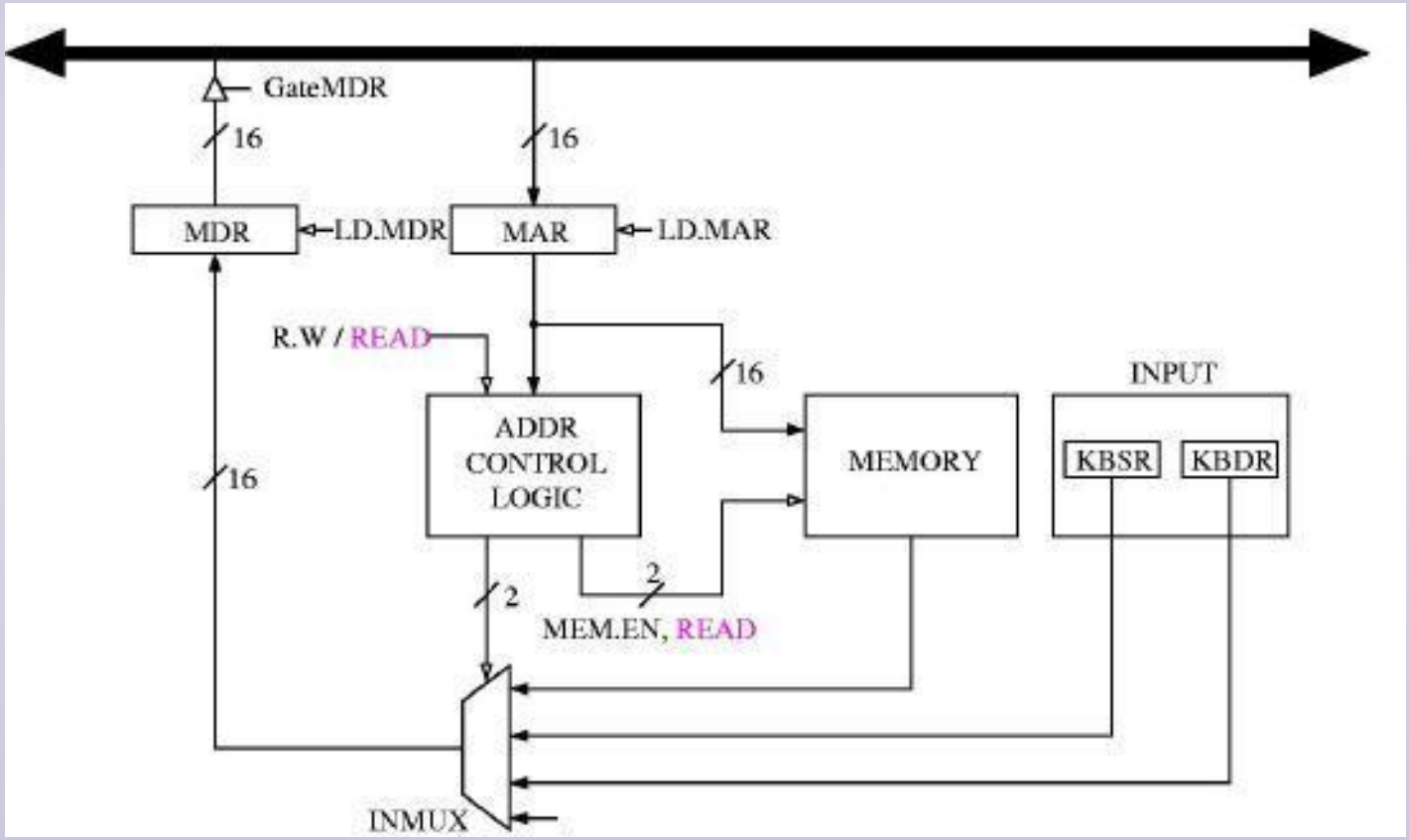
| | | |
|---|---|---|
| xFE00 | KBSR | Keyboard Status Register |
| xFE02 | KBDR | Keyboard Data Register |
| XFE04 | DSR | Display Status Register |
| XFE06 | DDR | Display Data Register |
| XFFFE | MCR | Machine Control Register |





Slides prepared by Walid A. Najjar & Brian J. Linard, University of California, Riverside

# Memory-mapped Input

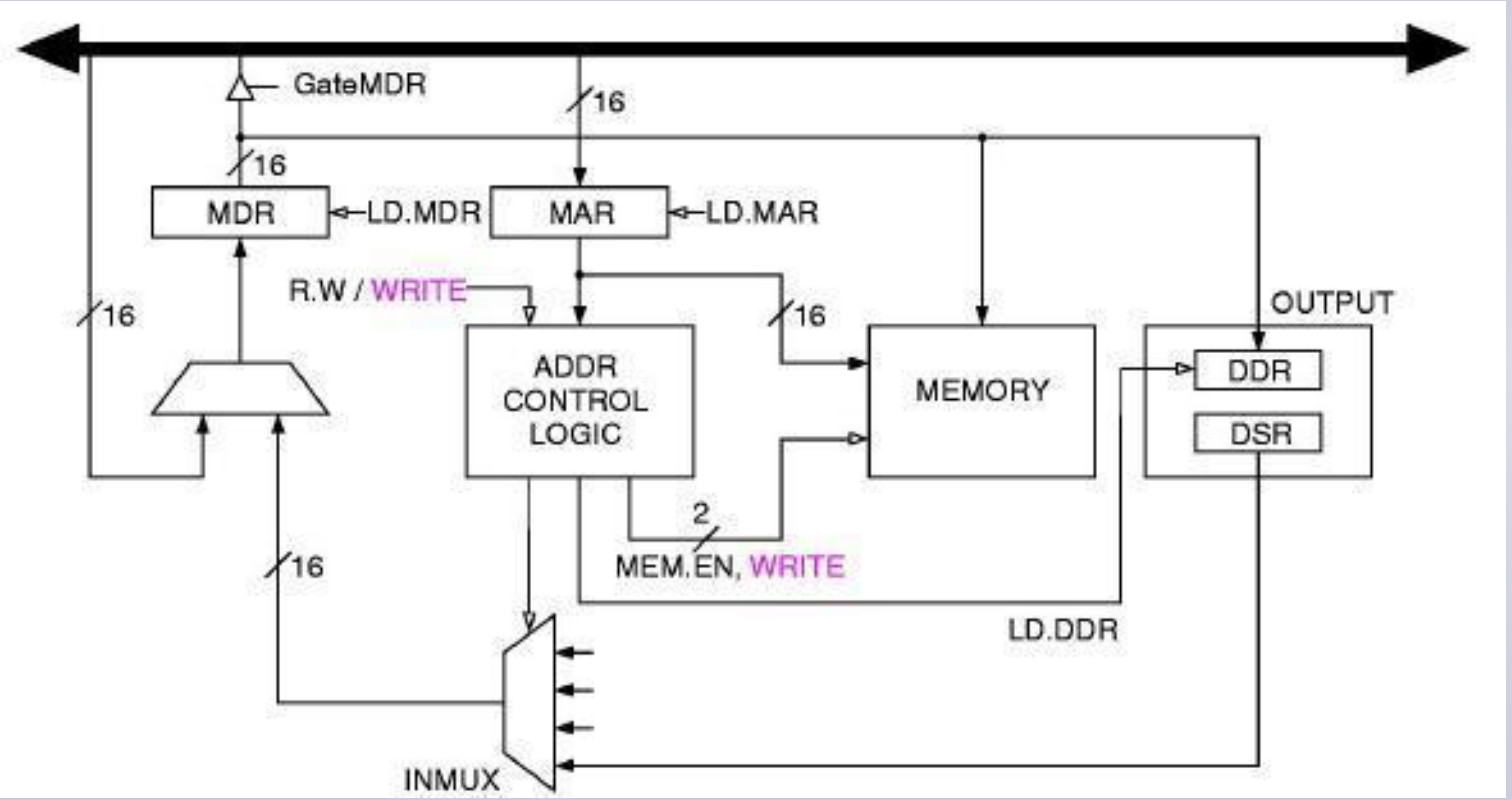







Slides prepared by Walid A. Najjar & Brian J. Linard, University of California, Riverside

# Memory-mapped Output

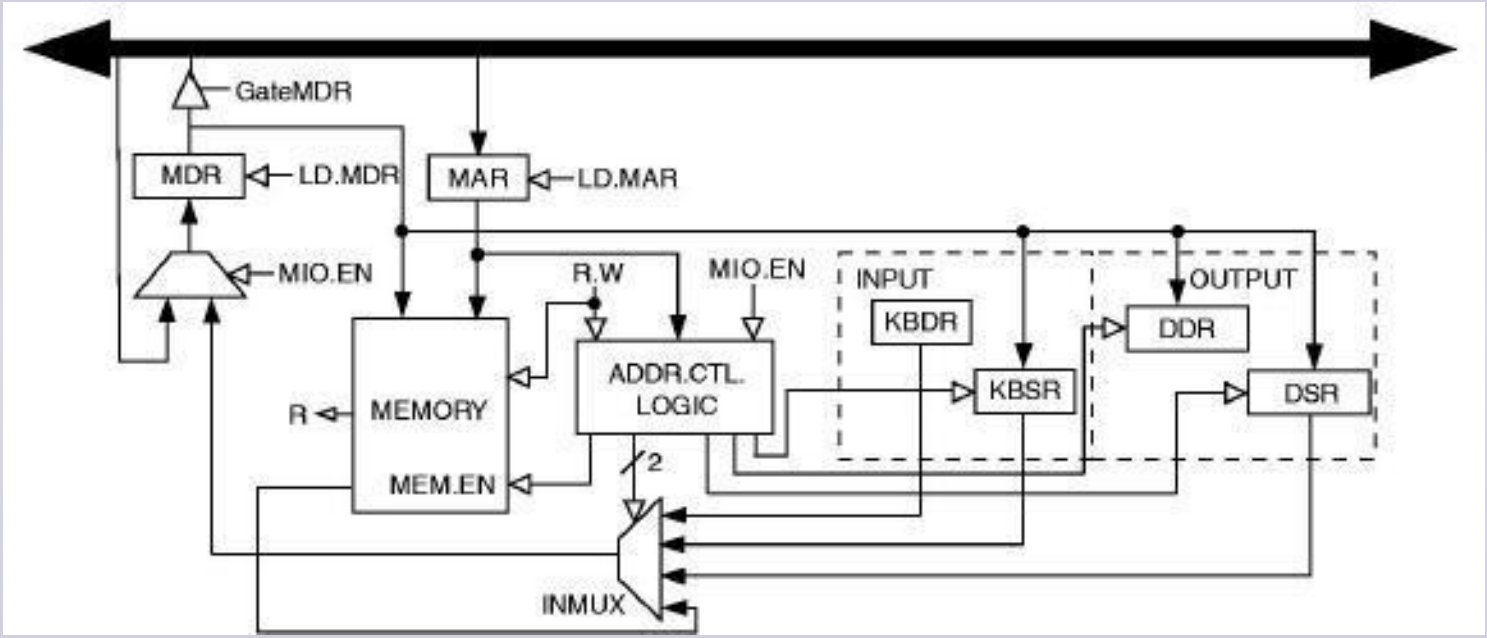







Slides prepared by Walid A. Najjar & Brian J. Linard, University of California, Riverside

# Memory-mapped I/O complete








Slides prepared by Walid A. Najjar & Brian J. Linard, University of California, Riverside

# Synchronizing CPU and I/O

- **Problem**
  - Speed mismatch between CPU and I/O:
    - *CPU runs at up to 2 GHz, while all I/O is much slower.*
  - Example : Keyboard input is both slow, and irregular.
  - We need a protocol to keep CPU & KBD synchronized.
- **Two possible solutions:**
  - Polling (handshake synchronization)
  - Interrupt-driven I/O

# Synchronizing CPU and I/O - 2

- **Polling, or handshake synchronization**
  - CPU checks the KBD Ready status bit.
  - If set, CPU reads the data register and resets the Ready bit.
  - Start over.
  - Makes CPU-I/O interaction *seem* to be synchronous.
- **Interrupt-driven I/O**
  - An external device is allowed to *interrupt* the CPU and demand attention
  - The CPU attends to the device in an orderly fashion (more later)

# Polling v/s Interrupts (Who's driving?)

- **Polling: CPU in charge**
  - CPU checks the ready bit of status register (as per program instructions).
    - *If (KBSR[15] == 1) then load KBDR[7:0] to a register.*
  - If the I/O device is very slow, CPU is kept busy waiting.
- **Interrupt: peripheral in charge**
  - Event triggered - when the I/O device is ready, it sets a flag called an interrupt.
  - When an interrupt is set, the CPU is forced to an *interrupt service routine* (ISR) which services the interrupting device.
  - There can be different priority levels of interrupt.
  - Specialized instructions can *mask* an interrupt level.





Slides prepared by Walid A. Najjar & Brian J. Linard, University of California, Riverside

# Polling Algorithm

- **Input (keyboard)**
  - *The CPU loops checking the Ready bit*
  - *When bit is set, a character is available*
  - *CPU loads the character waiting in the keyboard data register*
- **Output (monitor)**
  - *CPU loops checking the Ready bit*
  - *When bit is set, display is ready for next character*
  - *CPU stores a character in display data register*





Slides prepared by Walid A. Najjar & Brian J. Linard, University of California, Riverside

# Polling details

- **Keyboard**
  - When key is struck
    - *ASCII code of character is written to KBDR[7:0] (least significant byte of data register).*
    - *KBSR[15] (Ready Bit) is set to 1.*
    - *Keyboard is locked until CPU reads KBDR.*
    - *The CPU sees Ready Bit, reads KBDR, and clears the Ready Bit, unlocking the keyboard.*
- **Monitor**
  - When CPU is ready to output a character
    - *CPU checks DSR[15] (Ready Bit) until it is set to 1*
    - *CPU writes character to DDR[7:0]*
    - *Monitor sets DSR[15] to 0 while it is busy displaying the character, then sets it back to 1 to indicate readiness for next character.*





Slides prepared by Walid A. Najjar & Brian J. Linard, University of California, Riverside

# Simple Polling Routines

```
START   LDI R1, KBSR     ;Loop if Ready not set
        BRzp    START
        LDI R0, KBDR     ;If set, load char to R0
        BRnzp   NEXT_TASK
 KBSR   .FILL    xFE00   ;Address of KBSR
 KBDR   .FILL    xFE02   ;Address of KBDR

Input a character from keyboard
```

```
START   LDI R1, DSR;Loop if Ready not set
        BRzp    START
        STI R0, DDR      ;If set, send char to DDR
        BRnzp   NEXT_TASK
 DSR    .FILL    xFE04   ;Address of DSR
 DDR    .FILL    xFE06   ;Address of DDR

Output a character to the monitor
```





Slides prepared by Walid A. Najjar & Brian J. Linard, University of California, Riverside

# Keyboard Echo: combine the above

```
START  LDI     R1, KBSR    ;Loop if KB not ready
       BRzp    START
       LDI     R0, KBDR    ;Get character
ECHO   LDI     R1, DSR     ;Loop if monitor not ready
       BRzp    ECHO
       STI     R0, DDR     ;Send character
       BRnzp   NEXT_TASK

KBSR   .FILL   xFE00       ;Address of KBSR
KBDR       .FILL   xFE02       ;Address of KBDR
DSR    .FILL   xFE04       ;Address of DSR
DDR    .FILL   xFE06       ;Address of DDR
```





Slides prepared by Walid A. Najjar & Brian J. Linard, University of California, Riverside

# Example: Print a string

```
      LEA     R1, STR     ;Load address of string
LOOP  LDR     R0, R1, #0  ;get next char to R0
      BRz     DONE        ;string ends with x0000
LP2   LDI     R3, DSR     ;Loop until MON is ready
      BRzp    LP2
      STI     R0, DDR     ;Write next character
      ADD     R1, R1, #1  ;Set address to next char
      BRnzp       LOOP
DONE  BRnzp   NEXT_TASK

STR   .STRINGZ   "Char String"
```





Slides prepared by Walid A. Najjar & Brian J. Linard, University of California, Riverside

# Interrupt-driven I/O

- **Generating the interrupt signal**
  - The I/O device must want to request service.
  - The device must have the right to request service,
  - This request must be more urgent than the processor's current task.
- **Handling the interrupt signal**
  - We will wait untile we understand stacks before getting to this.

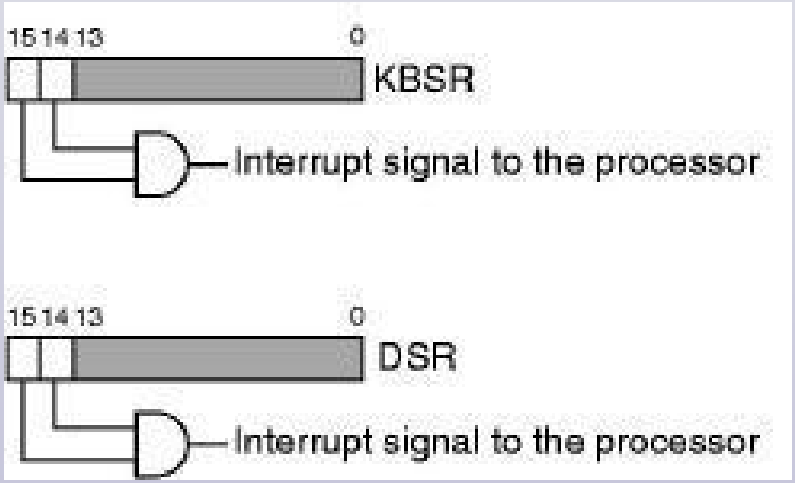




Slides prepared by Walid A. Najjar & Brian J. Linard, University of California, Riverside

# Generating the Interrupt

- **Using the Status Register**
  - The peripheral sets a Ready bit in SR[15] (as with polling)
  - The CPU sets an Interrupt Enable bit in SR[14]
  - These two bits are anded to set the Interrupt.
    - ***In this way, the CPU has the final say in who gets to interrupt it!***





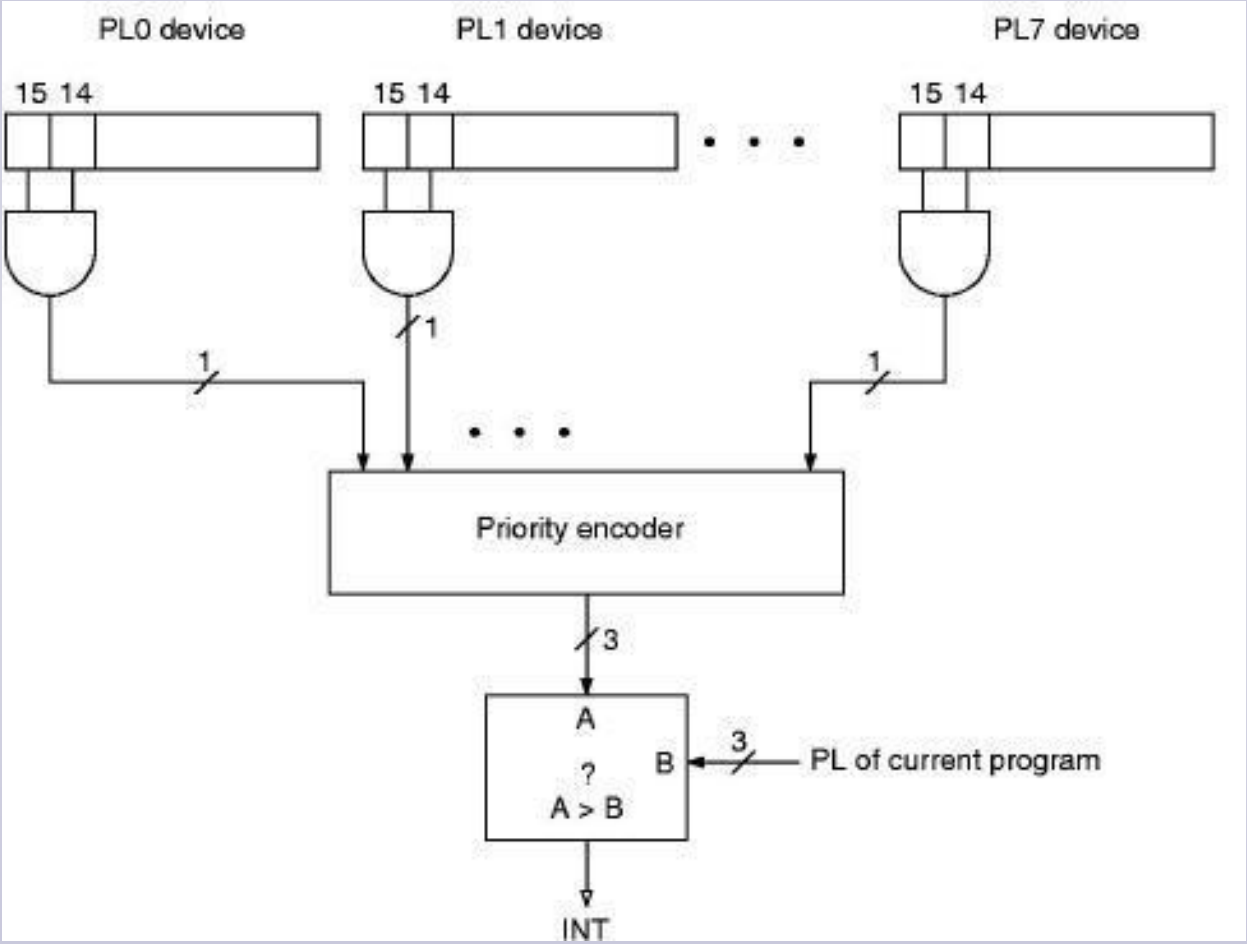



Slides prepared by Walid A. Najjar & Brian J. Linard, University of California, Riverside

# Priority

- Each task has an assigned priority level
  - *LC-3 has levels PL0 (lowest) to PL7 (highest).*
  - *If a higher priority task requests access, a lower priority task will be suspended.*
- Likewise, each device has an assigned priority
  - *The highest priority interrupt is passed on to the CPU only if it has higher priority than the currently executing task.*
- If an INT is present at the start of the instruction cycle, an extra step is inserted:
  - *The CPU saves its state information so that it can later return to the current task.*
  - *The PC is loaded with the starting address of the Interrupt Service Routine*
  - *The FETCH phase of the cycle continues as normal.*

# Device Priority



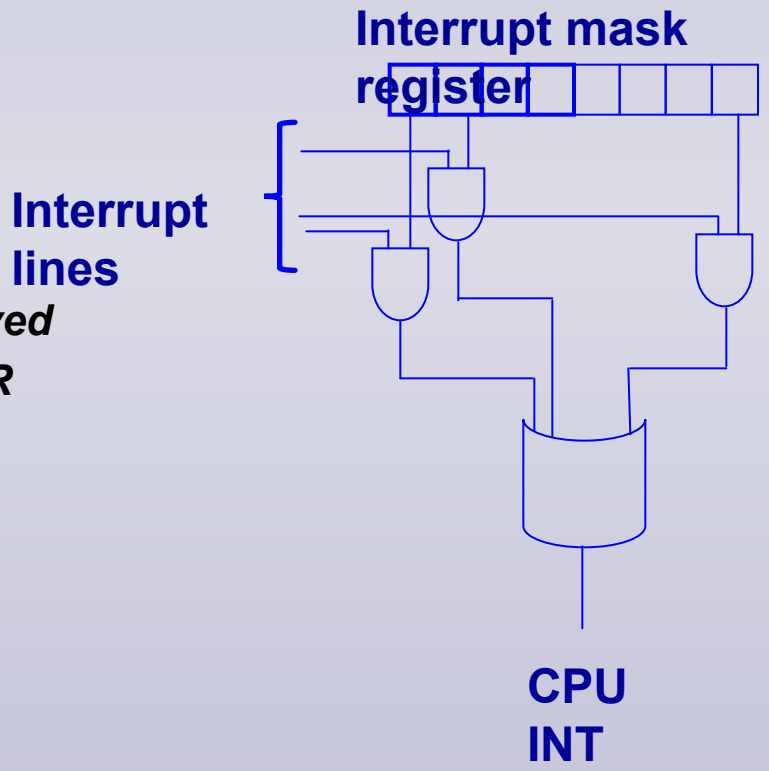





Slides prepared by Walid A. Najjar & Brian J. Linard, University of California, Riverside

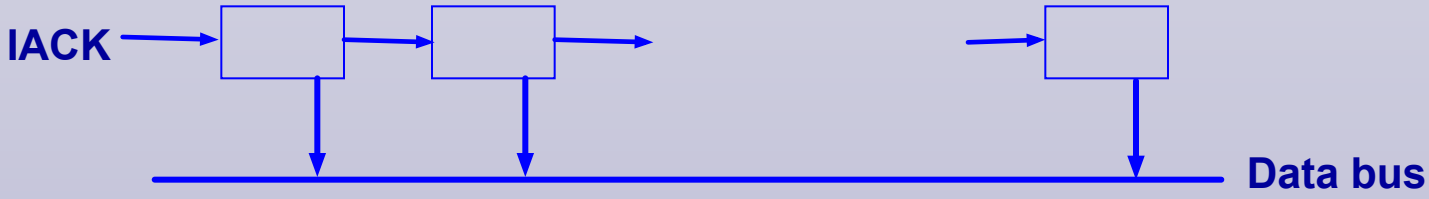
# Interrupts - More

- Another method of enabling devices to interrupt is the Interrupt mask register: each bit specifies whether the corresponding device is allowed to interrupt.
- In the CPU, the control logic checks the INT bit before each instruction fetch stage.
- If INT is set:
  - *CPU State information & PC are saved*
  - *PC ← address of* corresponding *ISR*
  - *Change mask settings*
    *(allow nested interrupts)*
  - *ISR is executed*
  - *Reset mask settings*
  - *Saved CPU state & PC are restored*
- How can CPU tell who interrupted?



# Interrupt – Who called?

- **How can CPU tell who interrupted?**
  - **Polling**
  - **Vectored Interrupts**
    - ***Multiple CPU INT lines, each dedicated to a device or group of devices (Intel: INT0, INT1 …)***
    - ***Each INT line sends the CPU to a specific ISR (Interrupt Service Routine).***
    - ***The ISR must figure out who called if more than one device is associated with that INT line.***
  - **Daisy Chain**
    - ***CPU sends an interrupt acknowledge (IACK) that is passed from one device to another.***
    - ***Interrupting device puts the address of its ISR on the Data Bus.***
    - ***Order of devices in the chain is the priority.***
    - ***Example: SCSII, USB***



# DMA – Direct Memory Access

- **DMA**
  - A device specialized in transferring data between memory and an I/O device (disk).
  - CPU writes the starting address and size of the region of memory to be copied, both source and destination addresses.
  - DMA does the transfer in the background.
  - It accesses the memory only when the CPU is not accessing it (*cycle stealing*).