I/O INTERFACING

I/O

- A computer’s job is to process data
  - Computation (CPU, cache, and memory)
  - Move data into and out of a system (between I/O devices and memory)

Register transfers

- Recall that the CPU is able to output to and input from memory using:
  - an address bus and decoder to select a particular register in memory,
  - a data bus to transfer the register's contents in or out of the CPU, and
  - a control bus to carry signals such at Read, Write, and Output Enable.

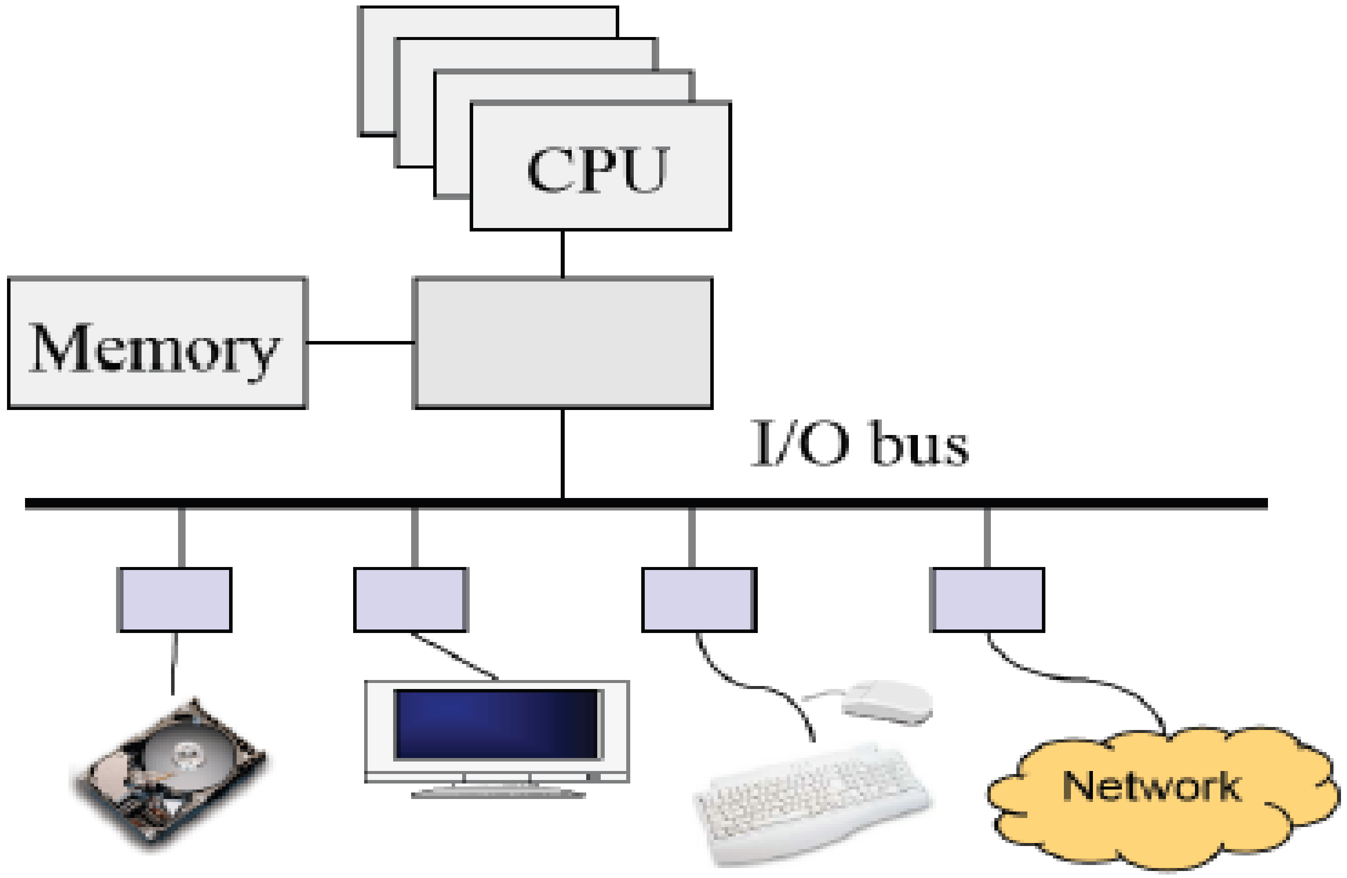
How does the CPU talk to devices?

- Device controller: Hardware that enables devices to talk to the
- peripheral bus
- Host adapter: Hardware that enables the computer to talk to the
- peripheral bus
- Bus: Wires that transfer data between components inside computer

Review: Computer Architecture

- Compute hardware
  - CPU and caches
  - Chipset
  - Memory
- I/O Hardware
  - I/O bus or interconnect
  - I/O controller or adaptor
  - I/O device

Review: Computer Architecture



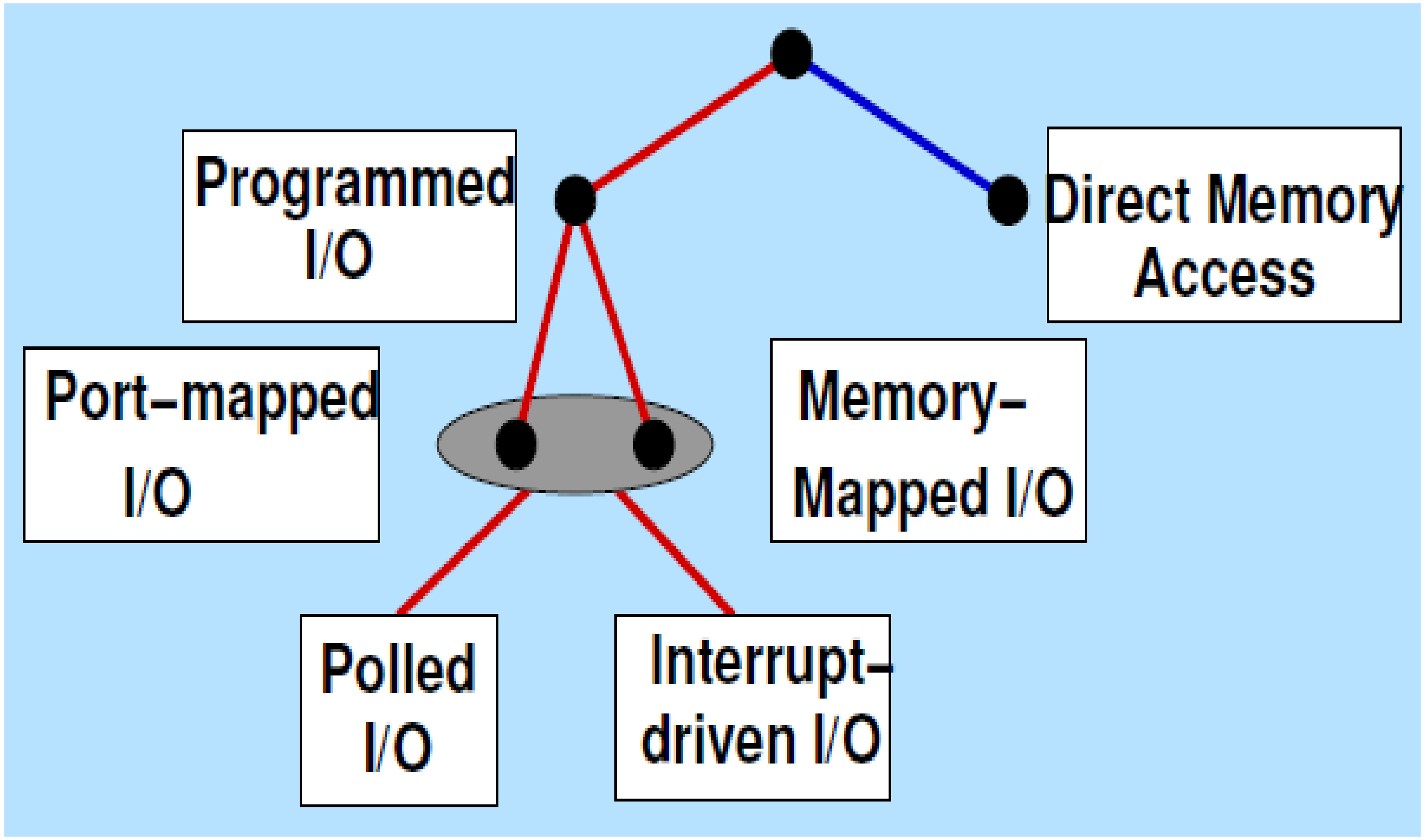
Types of I/O

- Two types of I/O:
  - Programmed I/O (PIO)

    CPU does the work of moving data
- Direct Memory Access (DMA)

  CPU offloads the work of moving data to DMA controller

Types of I/O



Programmed I/O
Direct Memory Access
Port-mapped I/O
Memory-Mapped I/O
Polled I/O
Interrupt-driven I/O

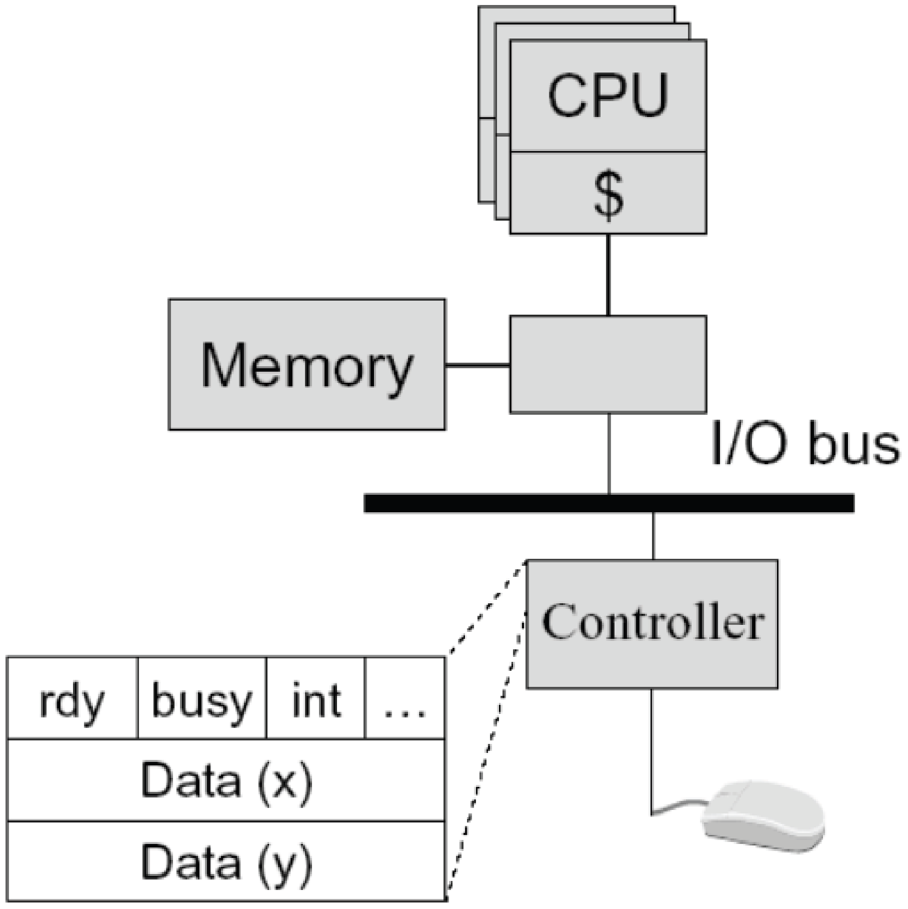
Programmed I/O

- Input:
  - Device controller
    - Status registers
      - ready: tells if the device is done with previous instruction.
      - busy: tells if the device is busy performing an instruction
    - Data registers

Programmed I/O

- Perform an Input:
  - Device tests the status register
  - If it is ready then data is taken in data register
  - And busy flag is set

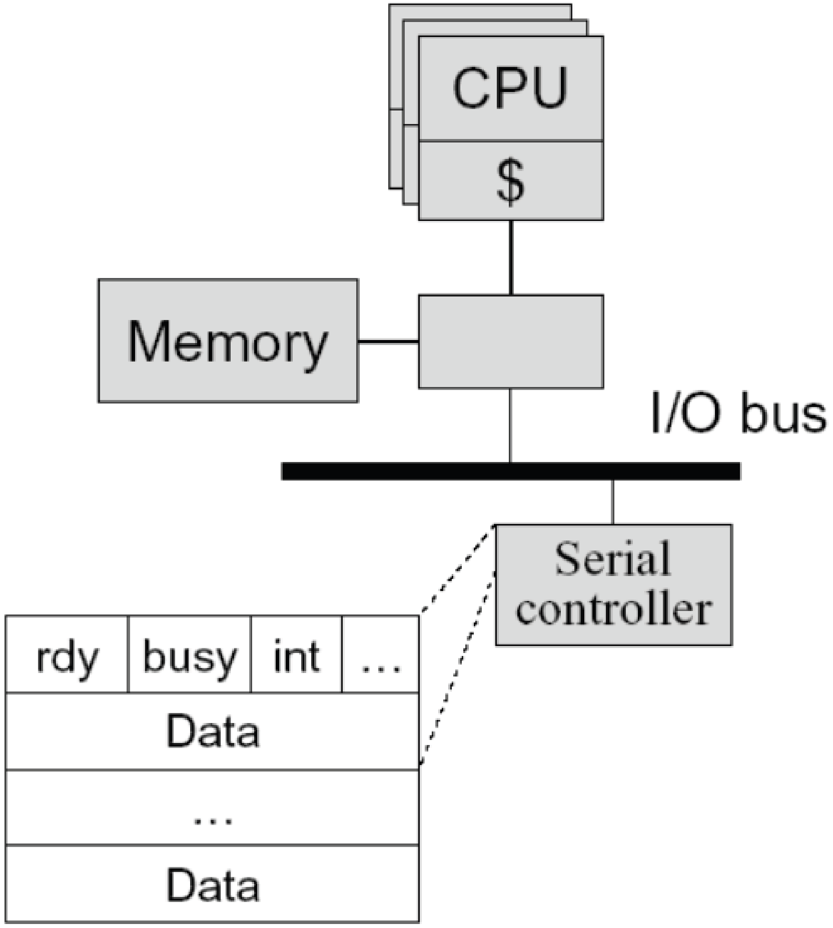
Programmed I/O



Programmed I/O

- Perform an output
  - CPU: address the PD and test the ready bit
  - If ready bit is set
  - Writes the data to data register(s)
  - Controller sets busy bit and transfers data
  - Controller clears the busy and ready bit

Programmed I/O



- If there are more than one device using programmed I/O, it is necessary to poll the ready bits of all the devices. The technique of testing a number of peripherals in turn is known as **software polling.**

Let us suppose that there are three devices . **STAT1,STAT2,STAT3 are the** addresses of the status registers of these devices. **PROC1 , PROC2 ,PROC3 are the procedures to perform input operations.**

- Then the following program sequence tests the three devices:

```
INPUT: IN AL,STAT1
TEST AL , 1B
JZ DEV2
Call PROC1
DEV2: IN AL,STAT2
TEST AL , 1B
JZ DEV3
Call PROC2;
DEV3: IN AL,STAT3
TEST AL , 1B
JZ N0_INPUT
Call PROC3;
NO_INPUT: JMP INPUT
```

Interrupt

When a process is executed by the CPU and when a user request for another process then this will create a disturbance for running/current process, this is called as interrupt.

Types of Interrupt?

What happens when MP is interrupted?

- When the Microprocessor receives an interrupt signal, it suspends the currently executing program and jumps to an Interrupt Service Routine (ISR) to respond to the incoming interrupt.
- Each interrupt will have its own ISR.
- After finishing the second program/interrupt, automatically return to the first program and start execution from where it was left.
- Interrupt service routine is a set of program/instruction which tells the MP how to handle the interrupt.

Interrupt service routine and Interrupt Vector Table

- Every starting addresses of interrupt service routines are kept on Interrupt Vector Table.
- The first 1KB (Kilo Byte) of memory of 8086 is set aside as an interrupt vector table for storing the starting address of Interrupt Service Procedure (ISP). Since 4-bytes are required for storing starting address of ISPs, the table can hold 256 Interrupt starting address procedure.
- Interrupt Table total size 1KB = 1000B, Size of Every interrupt starting address is 4B. So no. of address can be hold = (1024/4) = 256.

Interrupt I/O

Interrupts allows the peripherals to signal its readiness by interrupting the processor .Thus it improves on programmed I/O by not requiring the program to sit idle while waiting for a peripheral to become ready.This allows more effective use of the processor, including running programs at the same time as I/O is being performed.But still it requires action from the CPU for each data to be transferred.

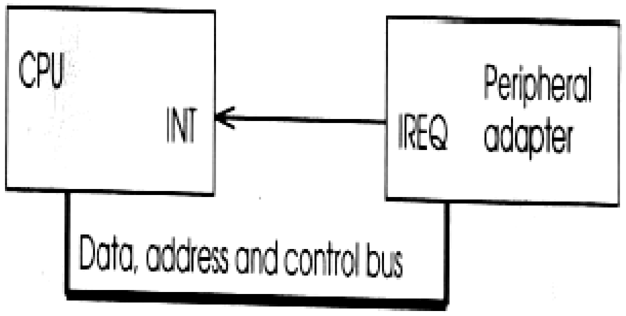


Figure 2.5 *Interrupt connection*

Interrupts

- An interrupt is an event that causes the CPU to initiate a fixed sequence, known as an interrupt sequence.
- When a peripheral is able to transfer data, it sets its ready flag in its status register and also asserts a control line (IREQ) connected to the CPU.

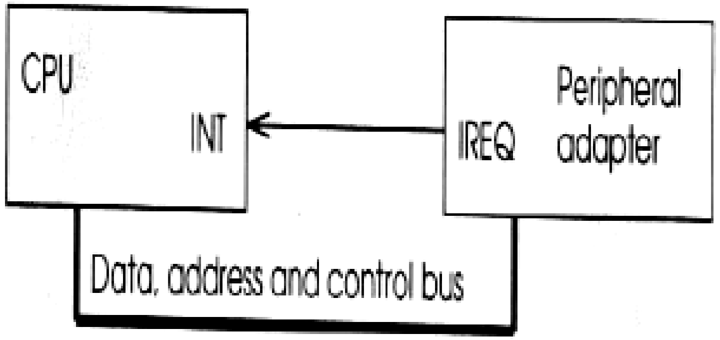


Figure 2.5 *Interrupt connection*

▪As soon as it is able (often at the end of the current instruction) the CPU then stops what it is doing,stores enough information to be able to resume later and starts executing an interrupt routine.

▪This routine deals with the device and then at the end uses the information stored at the beginning of the interrupt to return the CPU to executing the interrupted program.

▪Many devices may be able to interrupt.There is a list of addresses for the interrupt routines and one is chosen by using the interrupt identifier as an index into the table.
This table is known as ***Interrupt Vector Table*** (IVT).

▪Whenever a peripheral is ready to transfer data it is necessary to service its within a reasonable amount of time or else subsequent data may be lost.

▪ Faster peripherals require faster servicing. If two or more peripherals are ready at the same time, it is better to service the faster one first since the slow one can be made to wait a little while.

Priority Interrupts

Multiple interrupt requests can be resolved by using a priority interrupt scheme in which a hardware priority encoder arbitrates between requests and sends a single value that represents the highest priority device to the CPU. The interrupt request is formed by performing a logical OR of the peripheral's IREQ lines.

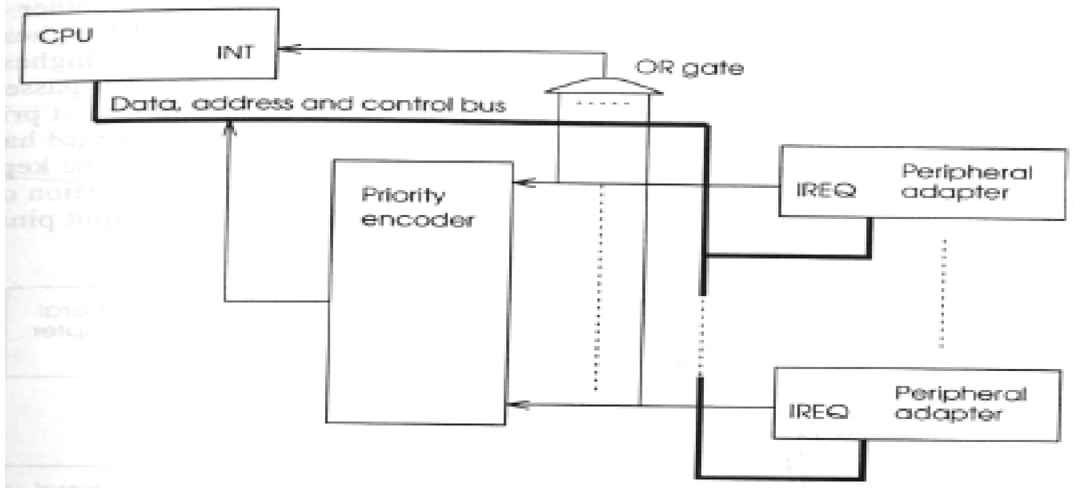


Figure 2.6 *Priority interrupts using a priority encoder*

Priority Interrupt or Parallel Interrupt

11.5 Priority Interrupt - Parallel Priority Interrupt

- Fig 11-14
- Priority according to the position of the bits in the register.
- Masking registers be set by program
- IST (interrupt Status F-F) (useful to make VAD), IEN
- Priority Encoder. Truth table 11-2

TABLE 11-2 Priority Encoder Truth Table

Inputs				Outputs			
I_0	I_1	I_2	I_3	x	y	IST	Boolean functions
1	×	×	×	0	0	1	
0	1	×	×	0	1	1	$x = I_0'I_1'$
0	0	1	×	1	0	1	$y = I_0'I_1 + I_0'I_2'$
0	0	0	1	1	1	1	$(IST) = I_0 + I_1 + I_2 + I_3$
0	0	0	0	×	×	0	

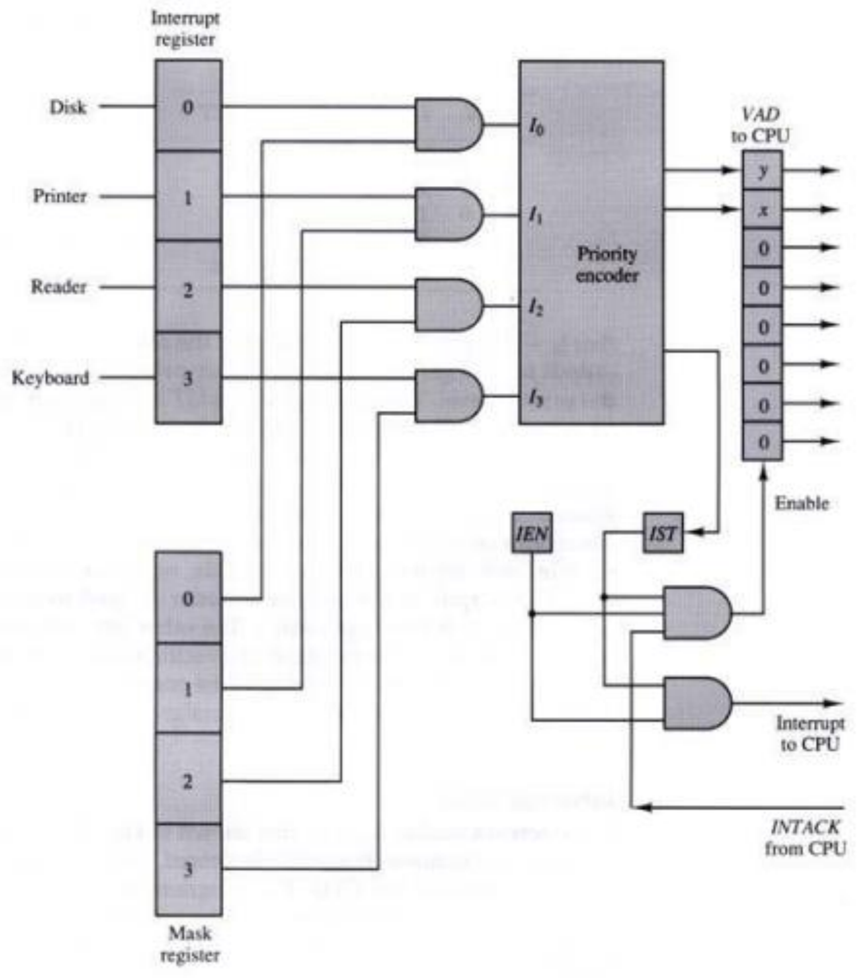


Figure 11-14 Priority interrupt hardware.

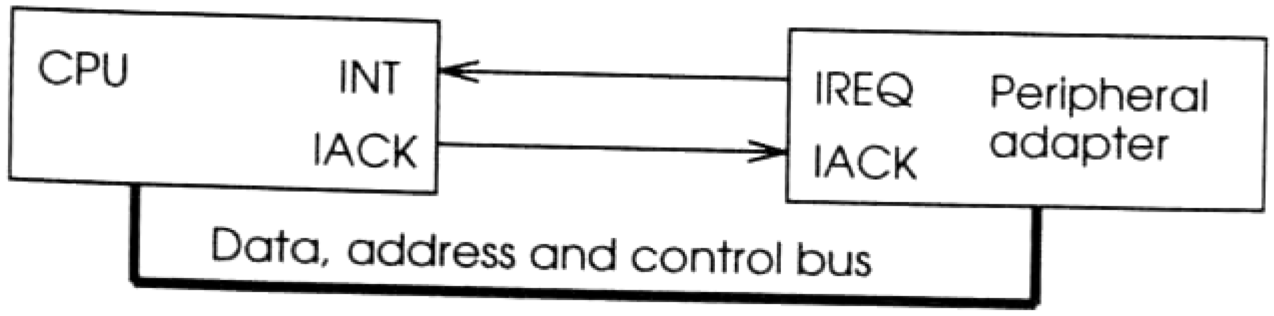


Figure 2.7 *Interrupts with acknowledge*

Priority Interrupts Using Daisy Chain

Using an interrupt acknowledgement it is possible to construct a simpler priority scheme.

• In this scheme the CPU is able to determine *priority* not from the interrupt request but by which device the acknowledgement is sent to.

•In daisy chain fashion all the interrupt request lines are OR'ed together.

•The CPU IACK is connected directly to the highest priority device.

So if more than one request has been made the highest priority device sees it first. If it has not made a request , it passes the IACK along to the next device.

• This continues down to the lowest priority device which will receive an acknowledgement only if no other device has made a request.

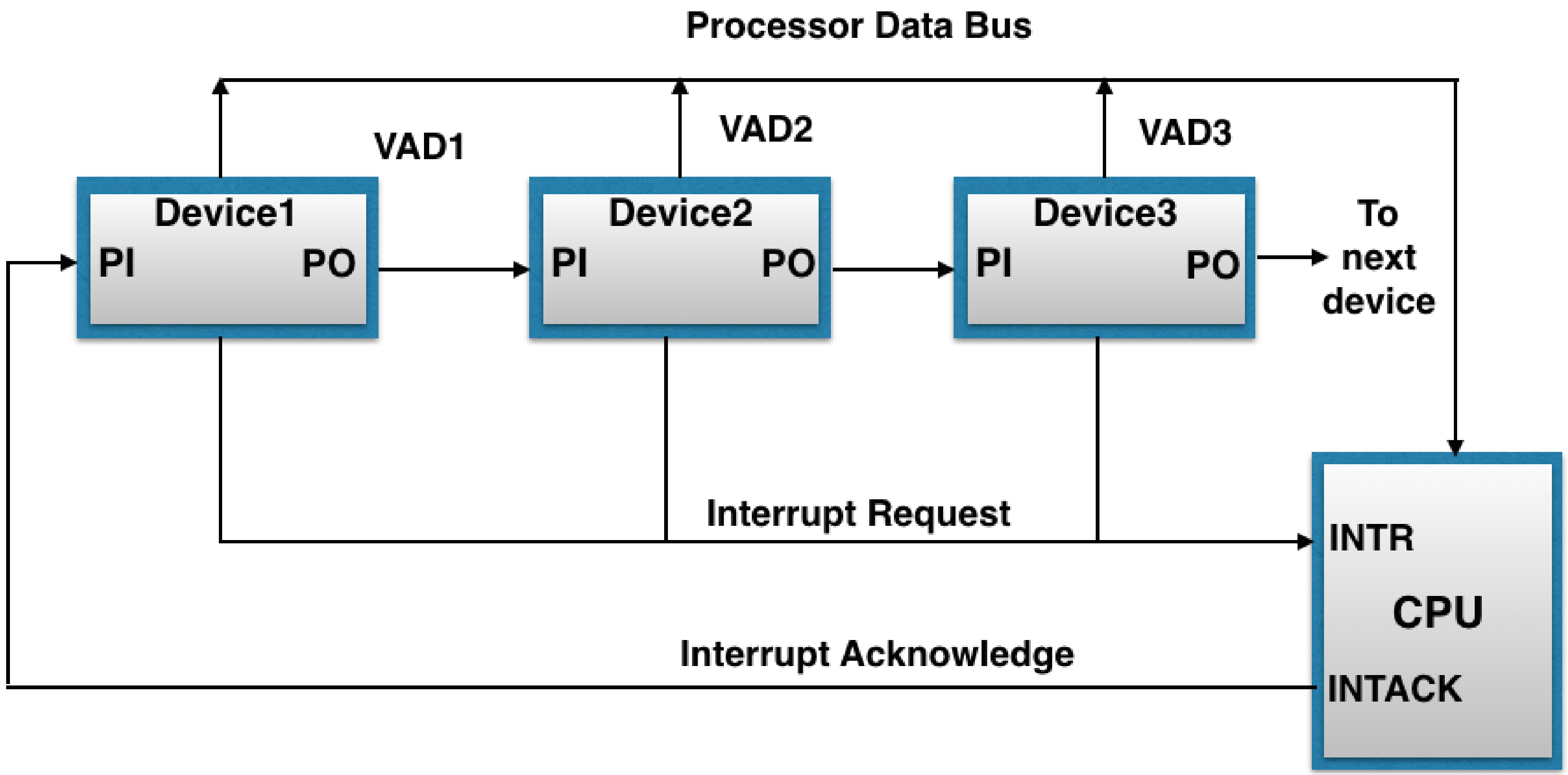


Figure: Priority interrupt using daisy chaining

Polling- vs. Interrupt-driven I/O

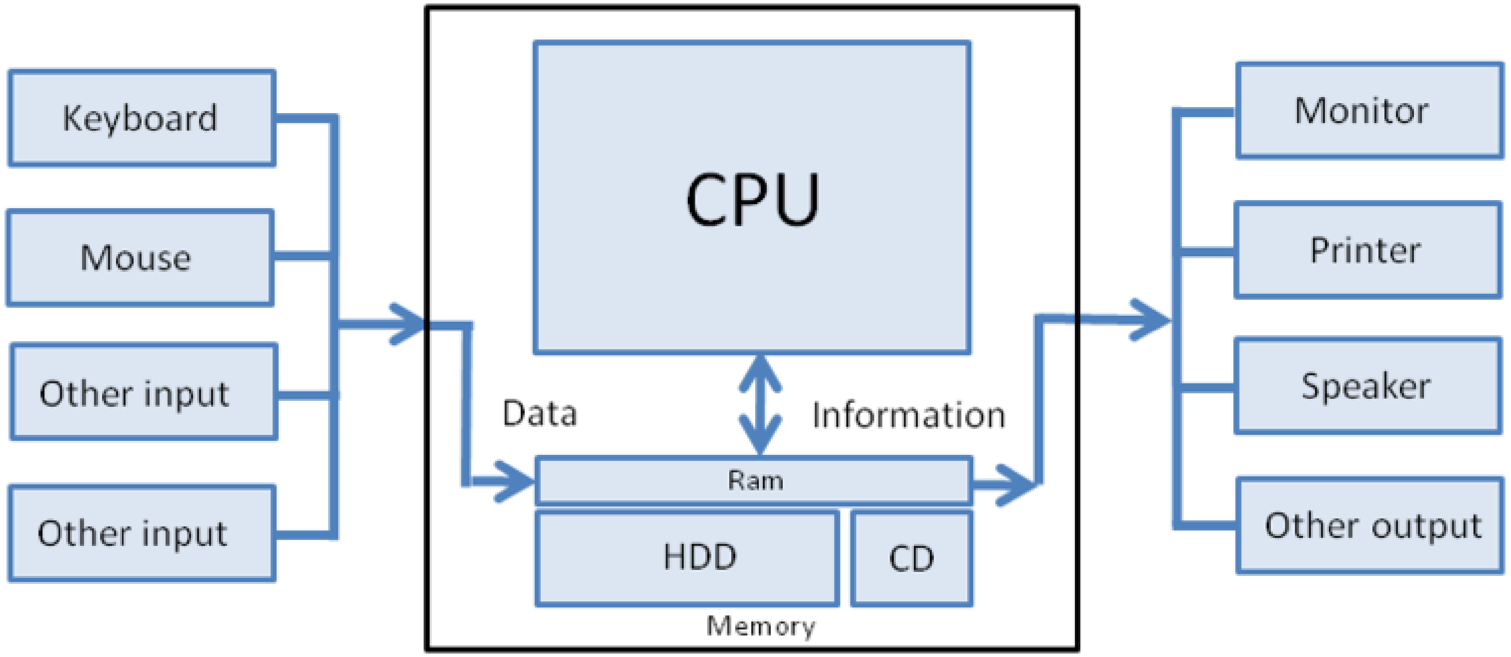
- Polling
  - CPU issues I/O command
  - In this method, Though CPU busy with any execution and completion of instruction, it continuously check the devices for any pending service request and if request found it will serve it.
- Interrupt-driven I/O
  - CPU issues I/O command
  - CPU directly writes instructions into device's registers
  - CPU continues operation until get any interrupt from any devices.

Polling- vs. Interrupt-driven I/O

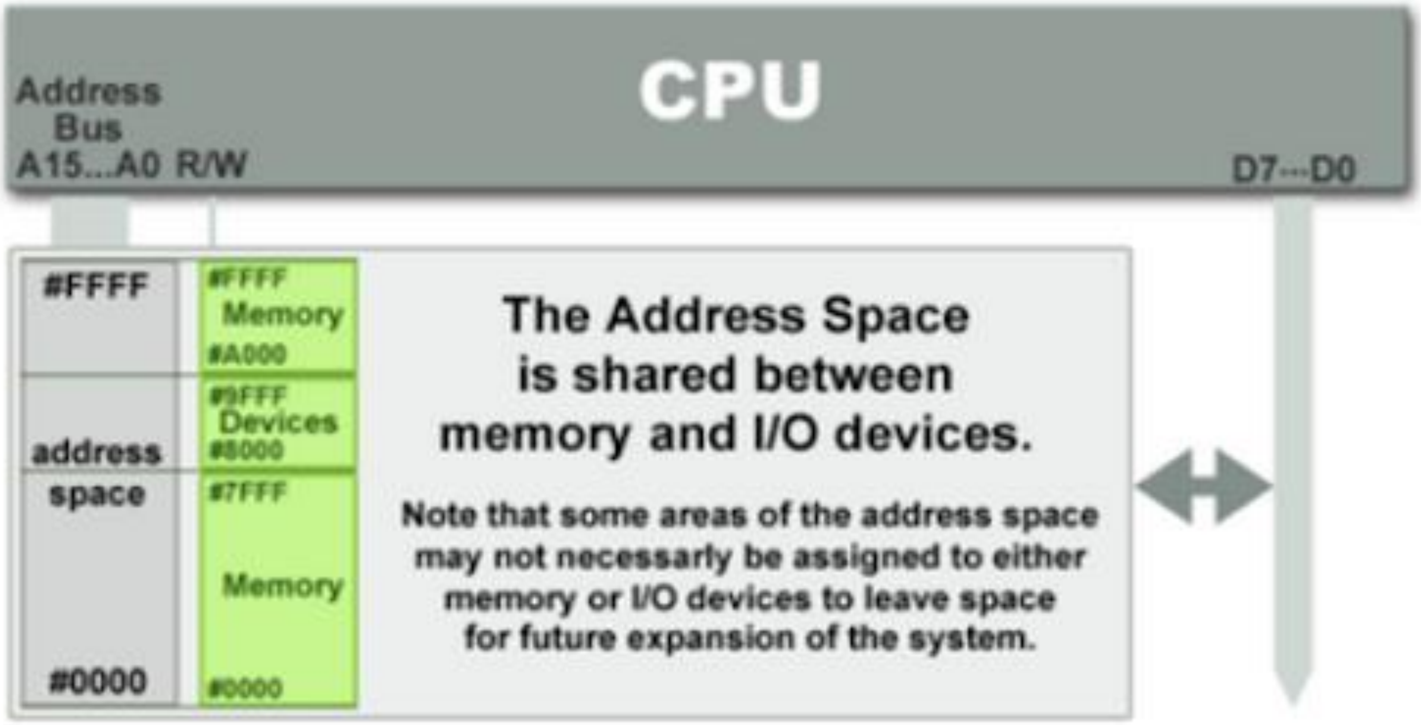
- Polling
  - Expensive for large transfers
  - Better for small, dedicated systems with infrequent I/O
- Interrupt-driven
  - No time waste in checking as CPU should not check manually like polling.
  - I/O module interrupts when ready: event driven

Memory mapped I/O

Memory-mapped I/O uses the same address space to address both **memory** and I/O devices.



Memory-mapped IO (MMIO)



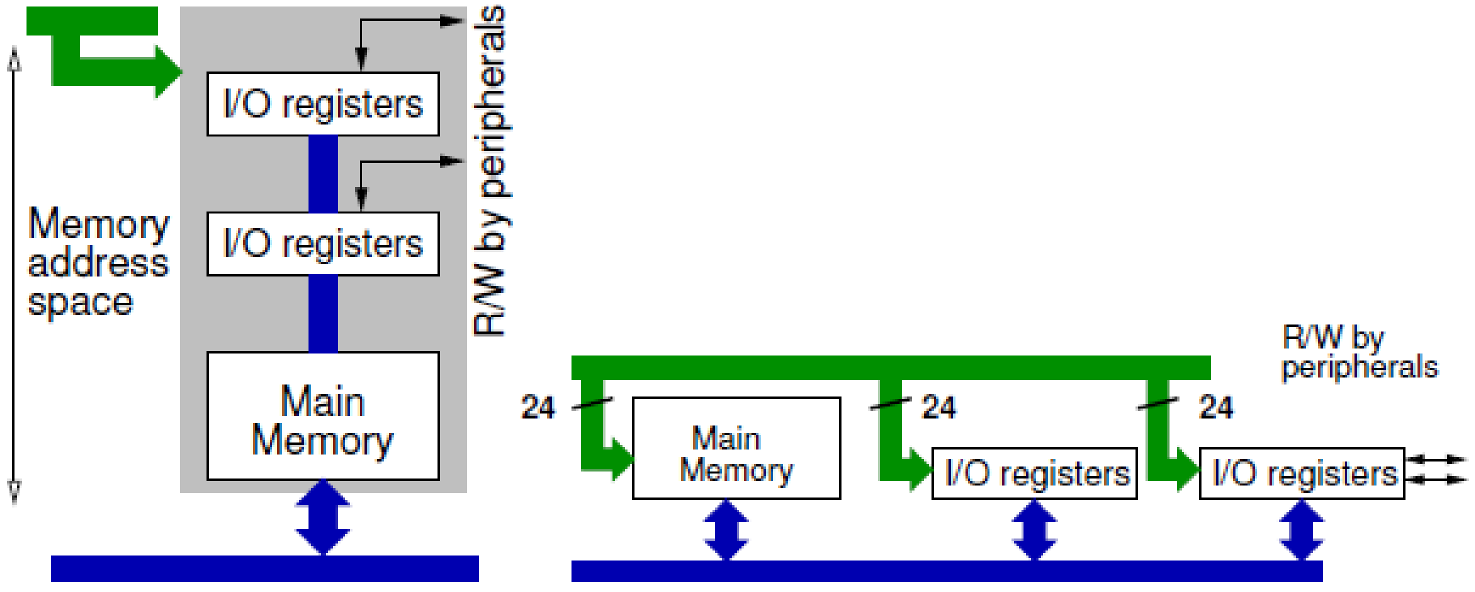
The Address Space is shared between memory and I/O devices.

Note that some areas of the address space may not necessarly be assigned to either memory or I/O devices to leave space for future expansion of the system.

Memory-mapped I/O

Use the same address bus to address both memory and I/O devices

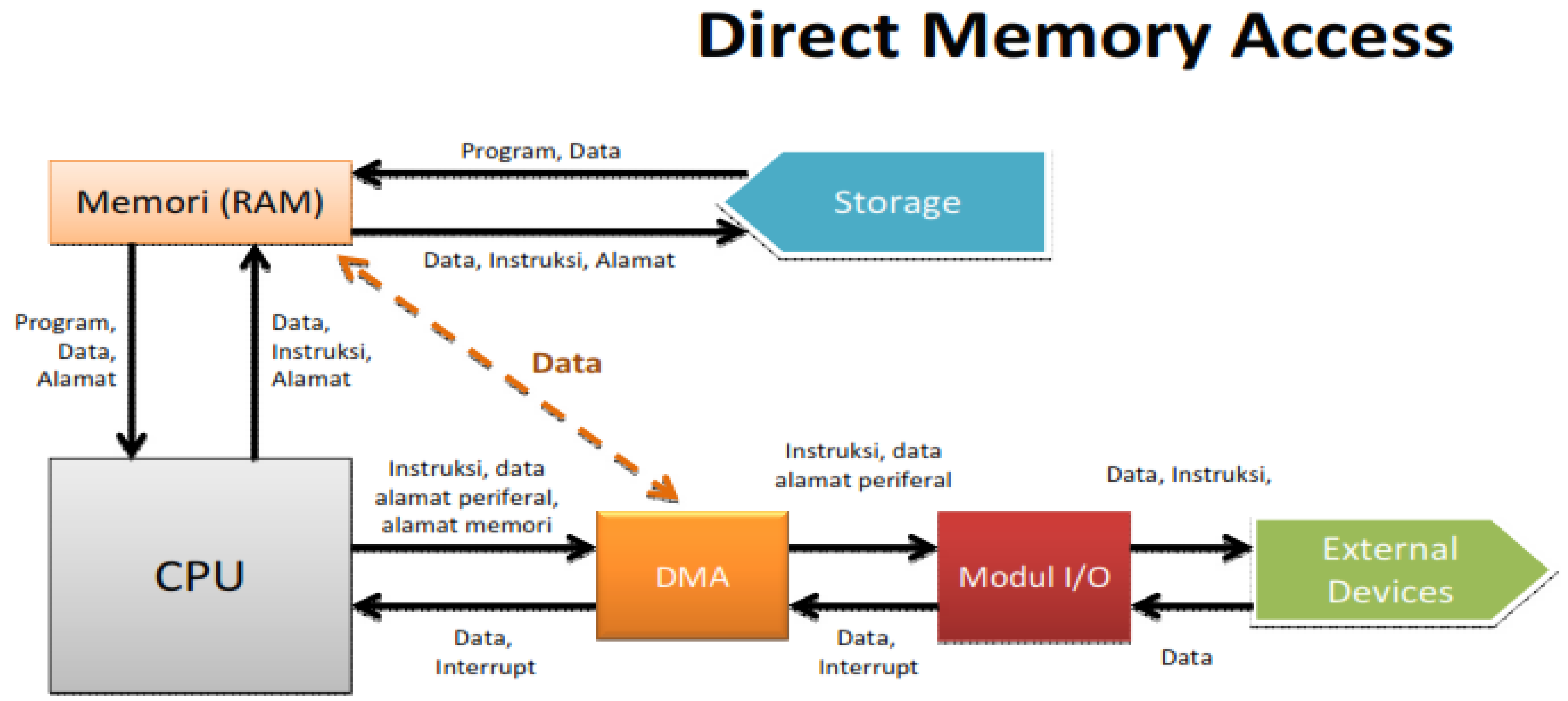
- The memory and registers of I/O devices are mapped to address values
- Allows same CPU instructions to be used with regular memory and devices



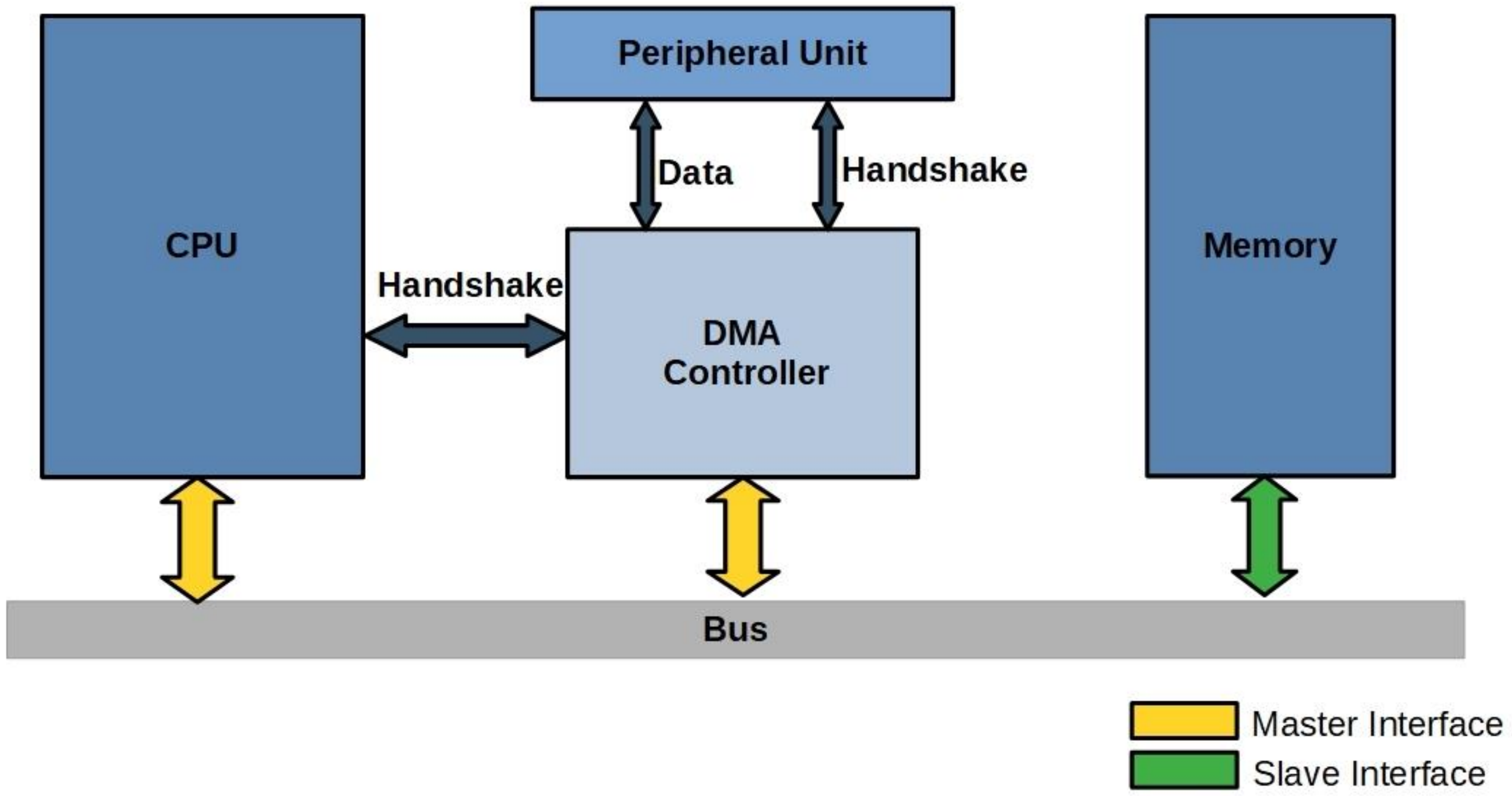
(a) (b)

Direct Memory Access (DMA)

Direct memory access (DMA) is a feature of computer systems that allows certain hardware and peripherals devices to **access** main **memory** (RAM: random-**access memory**) without intervention of the central processing unit (CPU). In normal system, peripherals devices first access in CPU , then they access in Memory. But, in DMA system, Peripherals devices can access in memory without accessing in CPU. But how? For this we need a direct memory access controller.



Direct Memory Access (DMA)



Direct Memory Access (DMA)

DMA controller or adaptor

- Status register (ready, busy, interrupt, …)
- DMA command register
- DMA register (address, size)
- DMA buffer

Host CPU initiates DMA

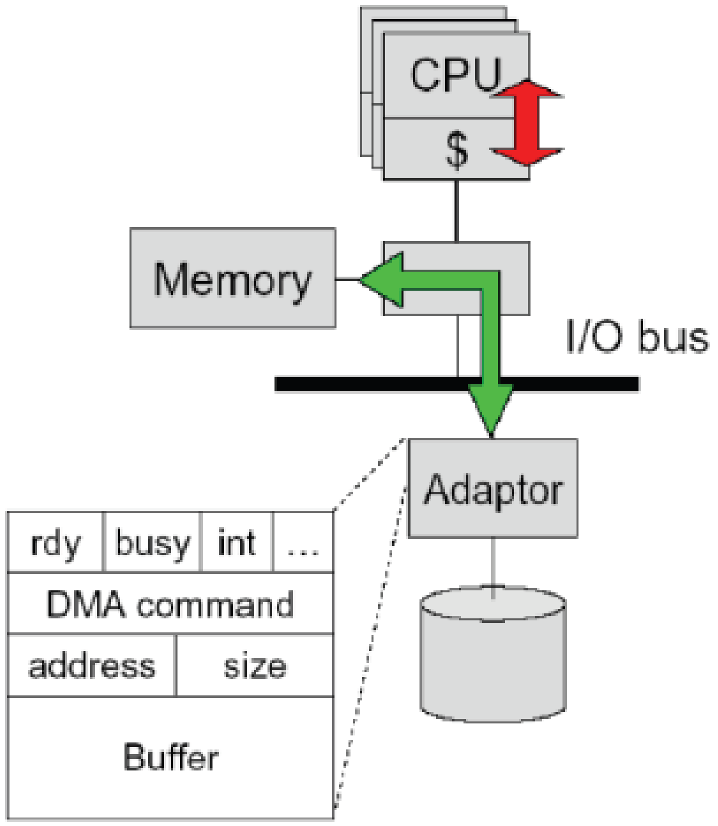
- Device driver call (kernel mode)
- Wait until DMA device is free
- Initiate a DMA transaction (command, memory address, size)
- Block

Controller performs DMA

- DMA data to device (size--; address++)
- Issue interrupt on completion (size == 0)

CPU's interrupt handler

- Wakeup the blocked process



- When a peripheral indicates that it is ready for a transfer, the DMA unit gains the control of the bus, places appropriate address and control signals on it to make the transfer and then releases the bus.
- This action of taking over the bus for a period and executing a memory access cycle instead of the CPU doing so is known as Cycle Stealing.