Unit 4-Basic I/O

I/O Overview

- One of the basic features of a computer is its ability to exchange data with other devices.
- They are an integral part of home appliances, manufacturing equipment, transportation systems, banking, and point-of-sale terminals.
- In such applications, input to a computer may come from a sensor switch, a digital camera, a microphone, or a fire alarm.
- Output may be a sound signal sent to a speaker, or a digitally coded command that changes the speed of a motor, opens a valve, or causes a robot to move in a specified manner.

I/O module

- In addition to the processor and a set of memory modules, the third key element of a computer system is a set of I/O modules.
- Each module interfaces to the system bus or central switch and controls one or more peripheral devices.
- An I/O module is not simply a set of mechanical connectors that wire a device into the system bus.
- Rather, the I/O module contains logic for performing a communication function between the peripheral and the bus.

I/O module..

Why one does not connect peripherals directly to the system bus?

The reasons are as follows:

- There are a wide variety of peripherals with various methods of operation. It would be impractical to incorporate the necessary logic within the processor to control a range of devices.
- The data transfer rate of peripherals is often much slower than that of the memory or processor. Thus, it is impractical to use the high-speed system bus to communicate directly with a peripheral.
- On the other hand, the data transfer rate of some peripherals is faster than that of the memory or processor. Again, the mismatch would lead to inefficiencies if not managed properly
- Peripherals often use different data formats and word lengths than the computer to which they are attached.

I/O module..

Two major functions of this module:

- Interface to the processor and memory via the system bus or central switch.
- Interface to one or more peripheral devices by tailored data links.

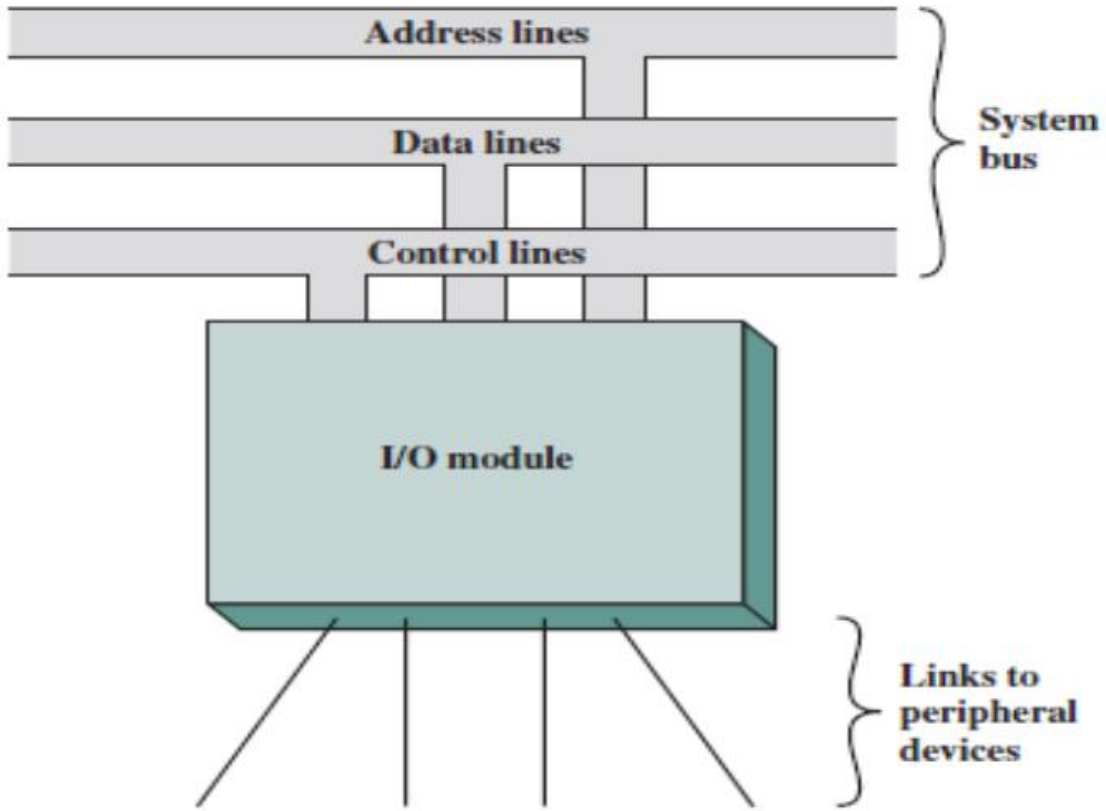


Figure 4.1 Generic Model of I/O Module

External devices

- Computer has ability to exchange data with other devices.
- An external device attaches to the computer by a link to an I/O module
- The link is used to exchange control, status, and data between the I/O module and the external device.
- An external device connected to an I/O module is often referred to as a peripheral device or, simply, a peripheral.

Classification external devices:

- Human-computer communication
- Computer-computer communication
- Computer-device communication

Block Diagram of an External Device

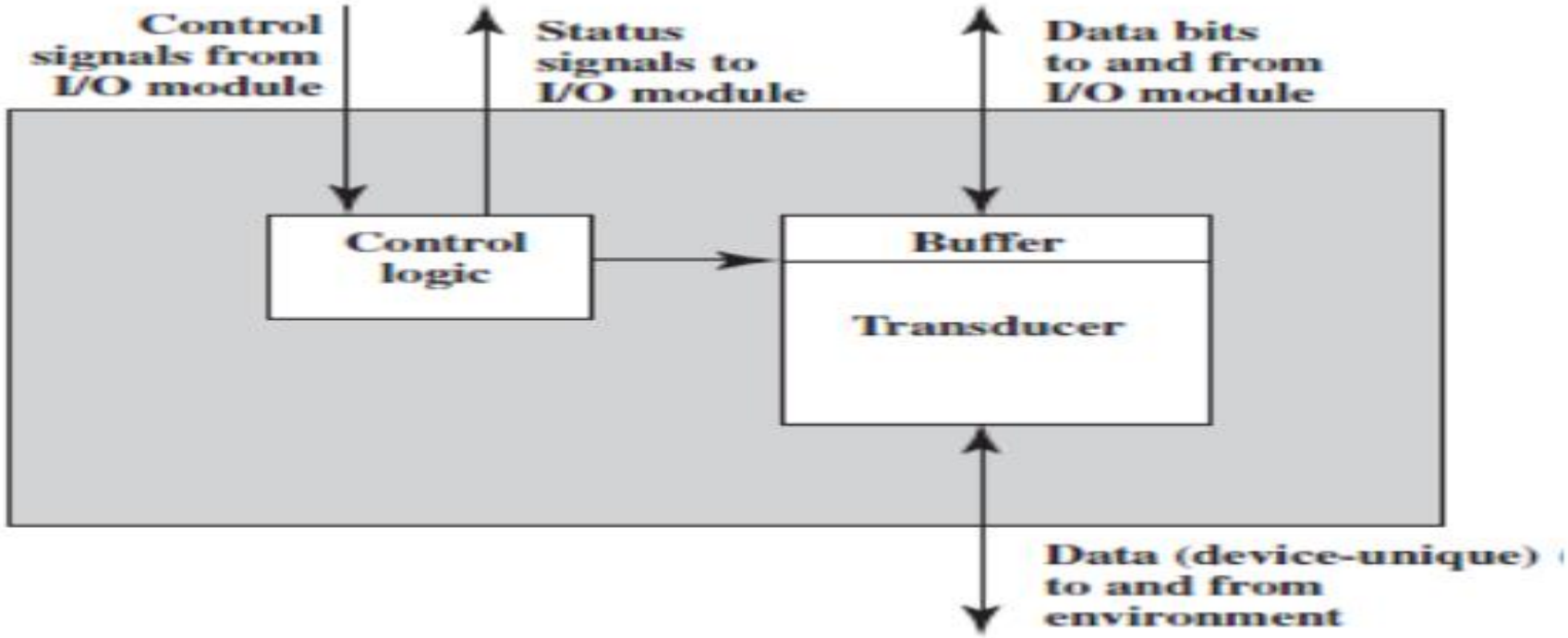


Figure 4.2 Block Diagram of External Device

Block Diagram of an External Device..

- Control signals determine the function that the device will perform, such as send data to the I/O module (INPUT or READ), accept data from the I/O module (OUTPUT or WRITE), report status, or perform some control function particular to the device
  - e.g., position a disk head.
- Data are in the form of a set of bits to be sent to or received from the I/O module.
- Status signals indicate the state of the device.
  - Examples are READY/ NOT-READY to show whether the device is ready for data transfer.

Block Diagram of an External Device..

- Control logic associated with the device controls the device's operation in response to direction from the I/O module.
- The transducer converts data from electrical to other forms of energy during output and from other forms to electrical during input. Typically, a buffer is associated with the transducer to temporarily hold data being transferred between the I/O module and the external environment.
  - A buffer size of 8 to 16 bits is common for serial devices, whereas block- oriented devices such as disk drive controllers may have much larger buffers.

I/O Modules functions

The major functions or requirements for an I/O module fall into the following categories:

- Control and timing
- Processor communication
- Device communication
- Data buffering
- Error detection

I/O module functions..

- The internal resources, such as main memory and the system bus, must be shared among a number of activities, including data I/O.
- Thus, the I/O function includes a control and timing requirement, to coordinate the flow of traffic between internal resources and external devices.

I/O module functions..

The control of the transfer of data from an external device to the processor might involve the following sequence of steps:

1. The processor interrogates the I/O module to check the status of the attached device.
2. The I/O module returns the device status.
3. If the device is operational and ready to transmit, the processor requests the transfer of data, by means of a command to the I/O module.
4. The I/O module obtains a unit of data (e.g., 8 or 16 bits) from the external device.
5. The data are transferred from the I/O module to the processor.

I/O module functions..

Processor Communication involves the following:

- **Command decoding:** The I/O module accepts commands from the processor, typically sent as signals on the control bus.
  - For example, an I/O module for a disk drive might accept the following commands: READ SECTOR, WRITE SECTOR, SEEK track number, and SCAN record ID. The latter two commands each include a parameter that is sent on the data bus.
- **Data**:Data are exchanged between the processor and the I/O module over the data bus.
- **Status reporting**: Because peripherals are so slow, it is important to know the status of the I/O module.
  - Common status signals are BUSY and READY. There may also be signals to report various error conditions.
- **Address recognition**: I/O module must recognize one unique address for each peripheral it controls.

I/O module functions..

- An essential task of an I/O module is **data buffering**.
- The transfer rate into and out of main memory or the processor is quite high, the rate is orders of magnitude lower for many peripheral devices and covers a wide range.
- Data coming from main memory are sent to an I/O module in a rapid burst.
- The data are buffered in the I/O module and then sent to the peripheral device at its data rate.
- I/O module is often responsible for **error detection** and for subsequently reporting errors to the processor.
  - One class of errors includes mechanical and electrical malfunctions reported by the device (e.g., paper jam, bad disk track)

Computer system

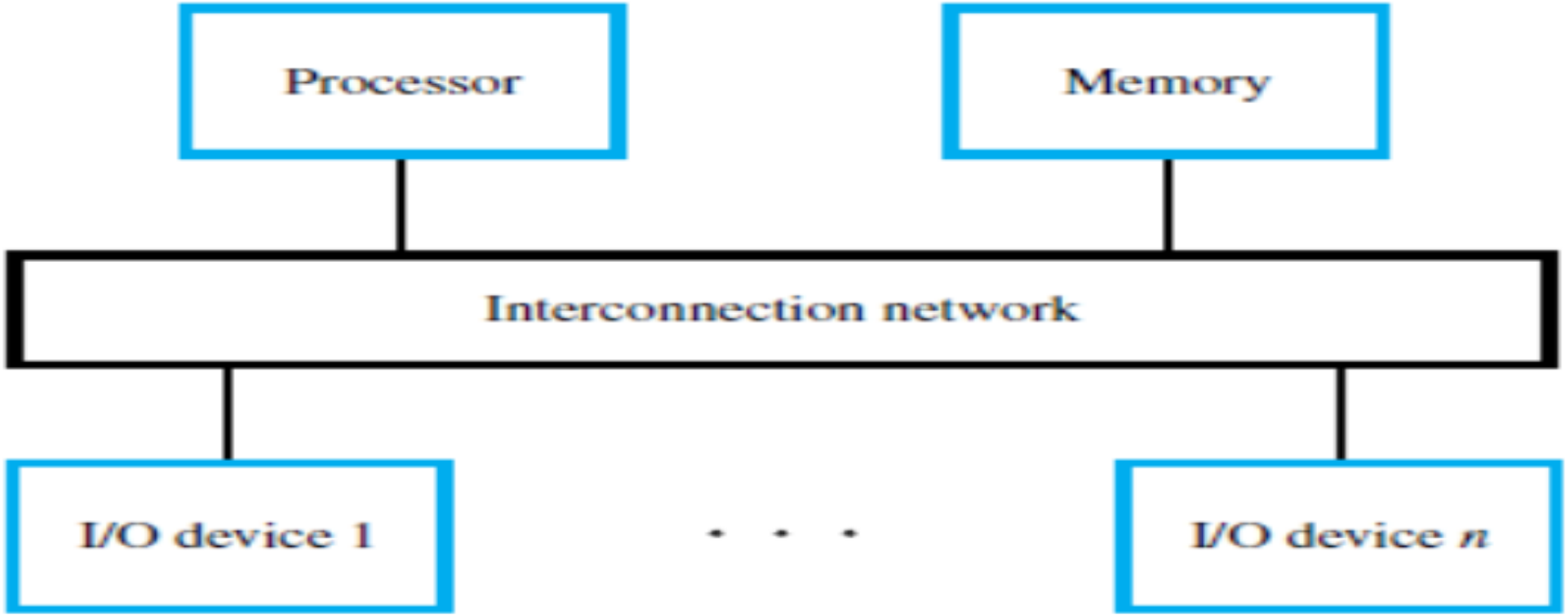


Figure 4.3 Computer System

I/O Module Structure

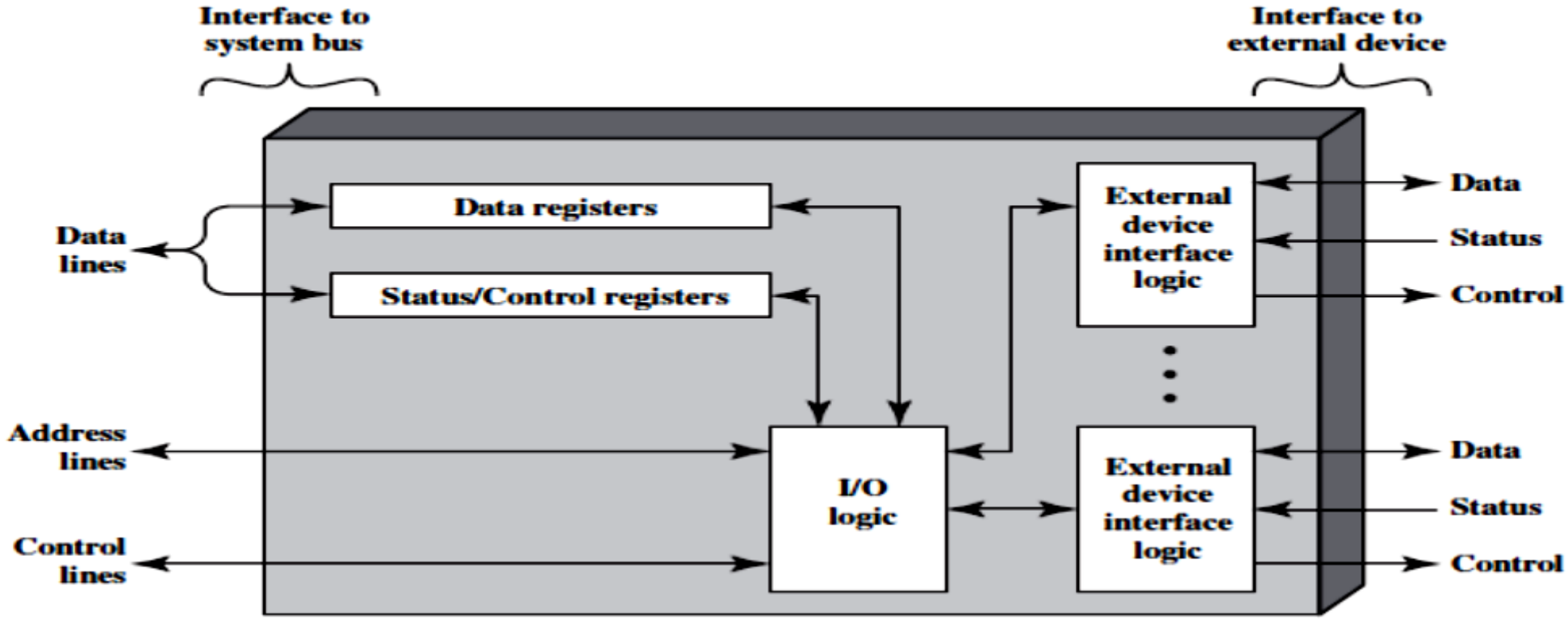


Figure 4.4 Block Diagram of IO Module

I/O Operations

Three techniques are possible for I/O operations:

- **Programmed I/O**, data are exchanged between the processor and the I/O module.
  - The processor executes a program that gives it direct control of the I/O operation, including sensing device status, sending a read or write command, and transferring the data.
  - When the processor issues a command to the I/O module, it must wait until the I/O operation is complete.
  - If the processor is faster than the I/O module, this is waste of processor time.
- **Interrupt-driven I/O**, the processor issues an I/O command, continues to execute other instructions, and is interrupted by the I/O module when the latter has completed its work.
- **Direct memory access (DMA),** In this mode, the I/O module and main memory exchange data directly, without processor involvement.

Programmed I/O

- When the processor is executing a program and encounters an instruction relating to I/O, it executes that instruction by issuing a command to the appropriate I/O module.
- With programmed I/O, the I/O module will perform the requested action and then set the appropriate bits in the I/O status register.
- The I/O module takes no further action to alert the processor.
- In particular, it does not interrupt the processor. Thus, it is the responsibility of the processor to periodically check the status of the I/O module until it finds that the operation is complete.

I/O Commands

To execute an I/O-related instruction, the processor issues an address, specifying the particular I/O module and external device, and an I/O command

Control: Used to activate a peripheral and tell it what to do.

- For example, a magnetic-tape unit may be instructed to rewind or to move forward one record

Test: Used to test various status conditions associated with an I/O module and its peripherals.

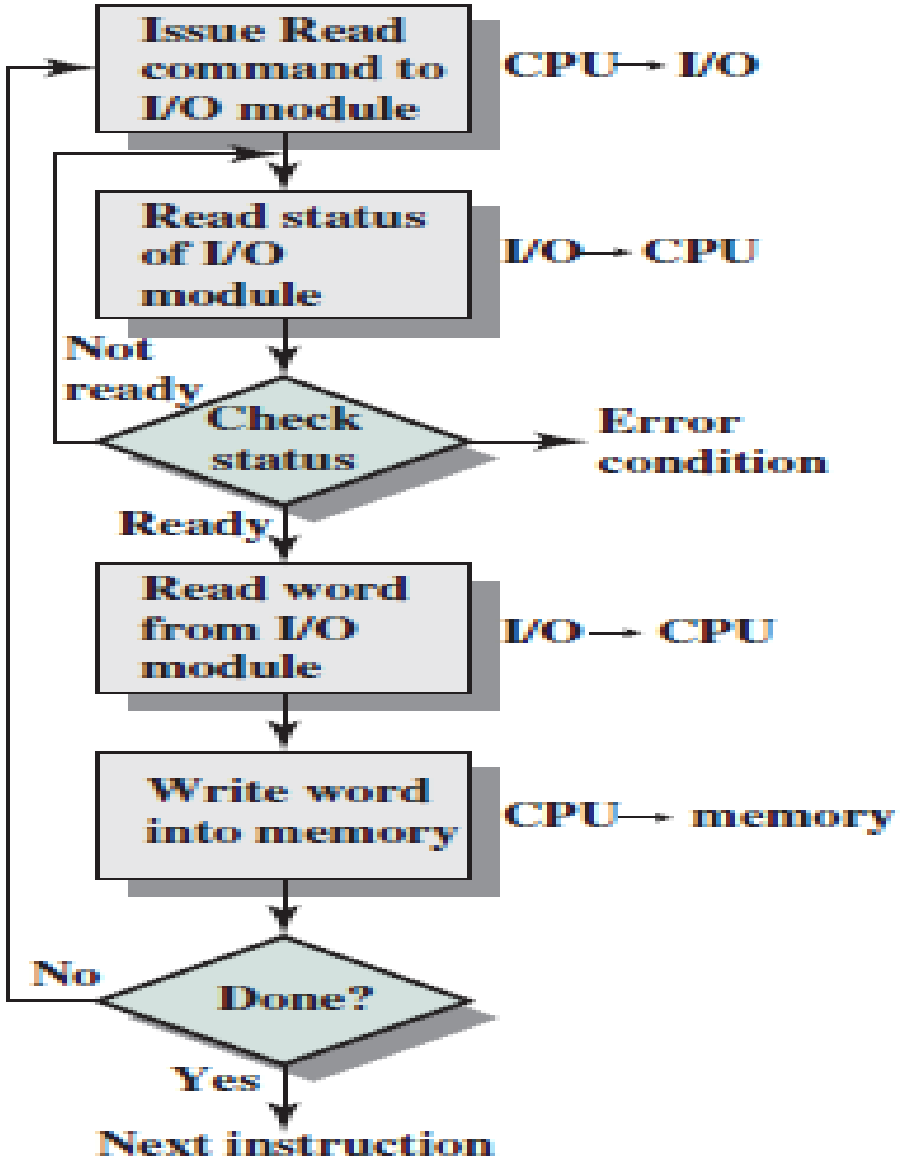
- The processor will want to know that the peripheral of interest is powered on and available for use.
- It will also want to know if the most recent I/O operation is completed and if any errors occurred.

I/O Commands..

Read: Causes the I/O module to obtain an item of data from the peripheral and place it in an internal buffer The processor can then obtain the data item by requesting that the I/O module place it on the data bus.

Write: Causes the I/O module to take an item of data (byte or word) from the data bus and subsequently transmit that data item to the peripheral.

Figure 4.5
Programmed I/O



I/O Instructions

- There will be many I/O devices connected through I/O modules to the system.
- Each device is given a unique identifier or address.
- When the processor issues an I/O command, the command contains the address of the desired device.
- Thus, each I/O module must interpret the address lines to determine if the command is for itself.

I/O Instructions..

- When the processor, main memory, and I/O share a common bus, two modes of addressing are possible: **memory mapped and isolated.**

I/O Instructions..

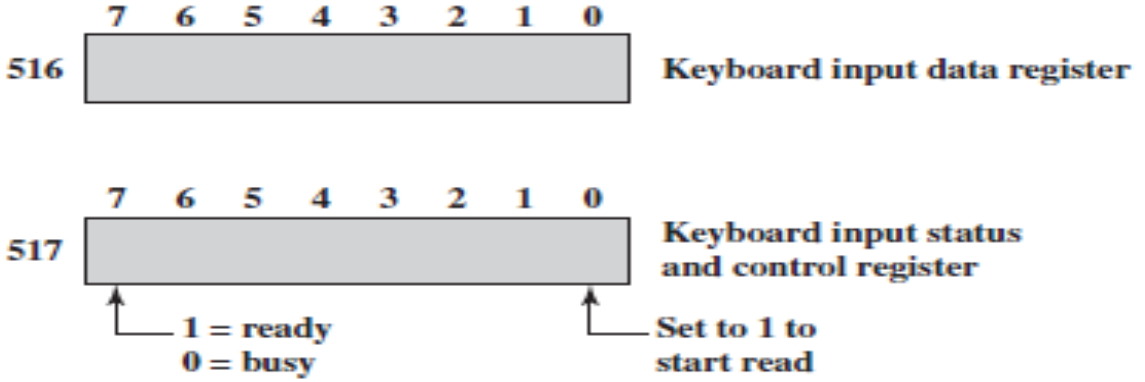
Memory-Mapped I/O

- There is a single address space for memory locations and I/O devices.
- The processor treats the status and data registers of I/O modules as memory locations and uses the same machine instructions to access both memory and I/O devices.
  - So, for example, with 10 address lines, a combined total of 2^{10} = 1024 memory locations and I/O addresses can be supported, in any combination.
- A single read line and a single write line are needed on the bus

I/O Instructions...

Isolated I/O

- The bus may be equipped with memory read and write plus input and output command lines
- The command line specifies whether the address refers to a memory location or an I/O device.
- The full range of addresses may be available for both
- With 10 address lines, the system may now support both 1024 memory locations and 1024 I/O addresses
- Because the address space for I/O is isolated from that for memory, this is referred to as **isolated I/O**



ADDRESS	INSTRUCTION	OPERAND	COMMENT
200	Load AC	"1"	Load accumulator
	Store AC	517	Initiate keyboard read
202	Load AC	517	Get status byte
	Branch if Sign = 0	202	Loop until ready
	Load AC	516	Load data byte

(a) Memory-mapped I/O

ADDRESS	INSTRUCTION	OPERAND	COMMENT
200	Load I/O	5	Initiate keyboard read
201	Test I/O	5	Check for completion
	Branch Not Ready	201	Loop until complete
	In	5	Load data byte

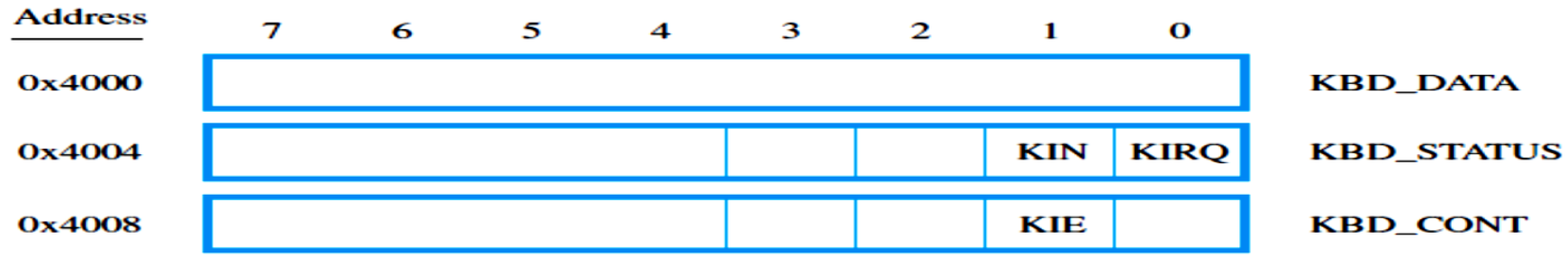
(b) Isolated I/O

Figure 4.6 Memory-Mapped and Isolated I/O

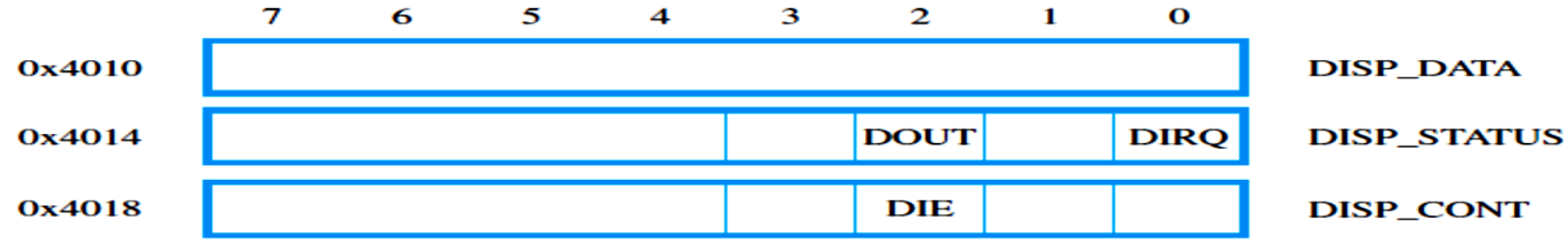
Program-Controlled I/O Example

- I/O devices operate at speeds that are very much different from that of the processor
- Keyboard, for example, is very slow
- It needs to make sure that only after a character is available in the input buffer of the keyboard interface; also, this character must be read only once

Program-Controlled I/O Example..



(a) Keyboard interface



(b) Display interface

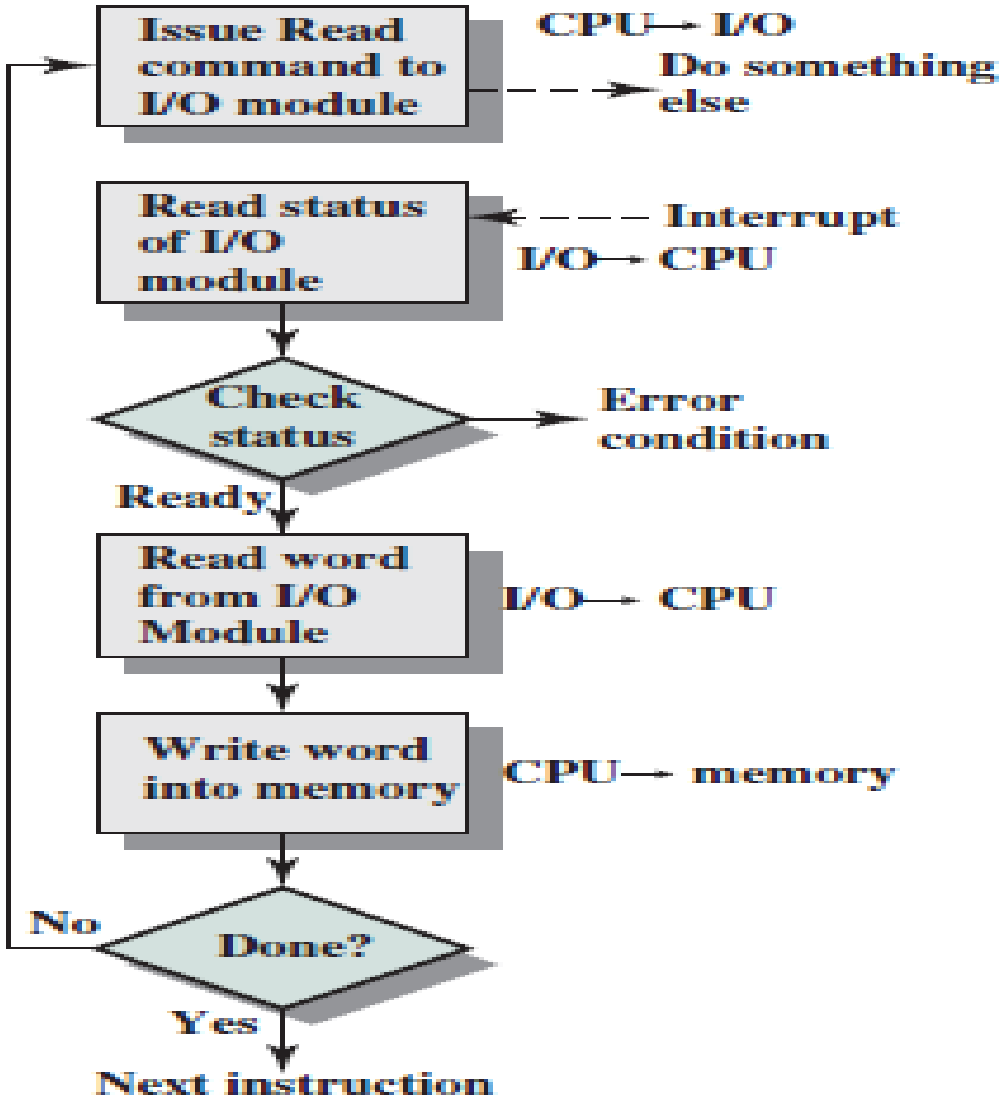
Figure 3.3 Registers in the keyboard and display interfaces.

```
        Move        R2, #LOC          Initialize pointer register R2 to point to the
                                        address of the first location in main memory
                                        where the characters are to be stored.
READ:   TestBit     KBD_STATUS, #1    Wait for a character to be entered
        Branch=0    READ                in the keyboard buffer KBD_DATA.
        MoveByte    (R2), KBD_DATA    Transfer the character from KBD_DATA into
                                        the main memory (this clears KIN to 0).
ECHO:   TestBit     DISP_STATUS, #2   Wait for the display to become ready.
        Branch=0    ECHO
        MoveByte    DISP_DATA, (R2)   Move the character just read to the display
                                        buffer register (this clears DOUT to 0).
        CompareByte (R2)+, #CR        Check if the character just read is CR
                                        (carriage return). If it is not CR, then
        Branch≠0    READ                branch back and read another character.
                                        Also, increment the pointer to store the
                                        next character.
```

Interrupt-Driven I/O

- The problem with the programmed I/O is the processor enters a wait loop in which it repeatedly tests the device status
- During this period, the processor is not performing any useful computation. There are many situations where other tasks can be performed while waiting for an I/O device to become ready.
- To allow this to happen, we can arrange for the I/O device to alert the processor when it becomes ready.
- It can do so by sending a hardware signal called an interrupt request to the processor. Since the processor is no longer required to continuously poll the status of I/O devices, it can use the waiting period to perform other useful tasks.

Interrupt-Driven I/O..



Interrupt Service Routine

ISR can get invoked from anywhere in the program that was executing –

- This anywhere means it depends on exactly where the interrupt signal arrived.
- So, potentially all the registers that are used in the service routine needs to be saved and restored.

Challenges in Interrupts

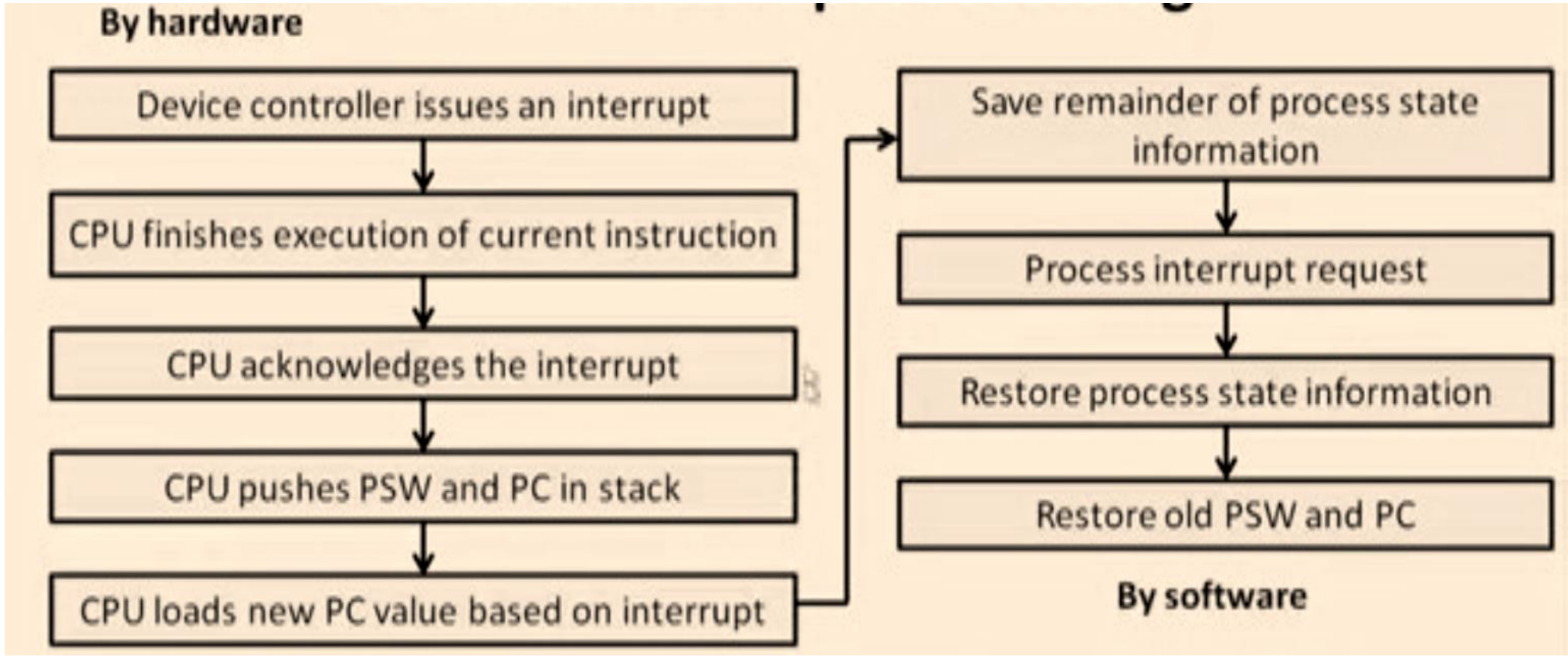
- For multiple sources of the interrupts, how to know the address of the ISR?
- How to handle multiple interrupts?
  - While an interrupt request is being processed, another interrupt request might come.
  - Enabling, disabling and masking of the interrupts
- How to handle simultaneously arriving interrupts?
- Sources of the interrupts other than I/O devices.
  - Exceptions

Interrupt handling

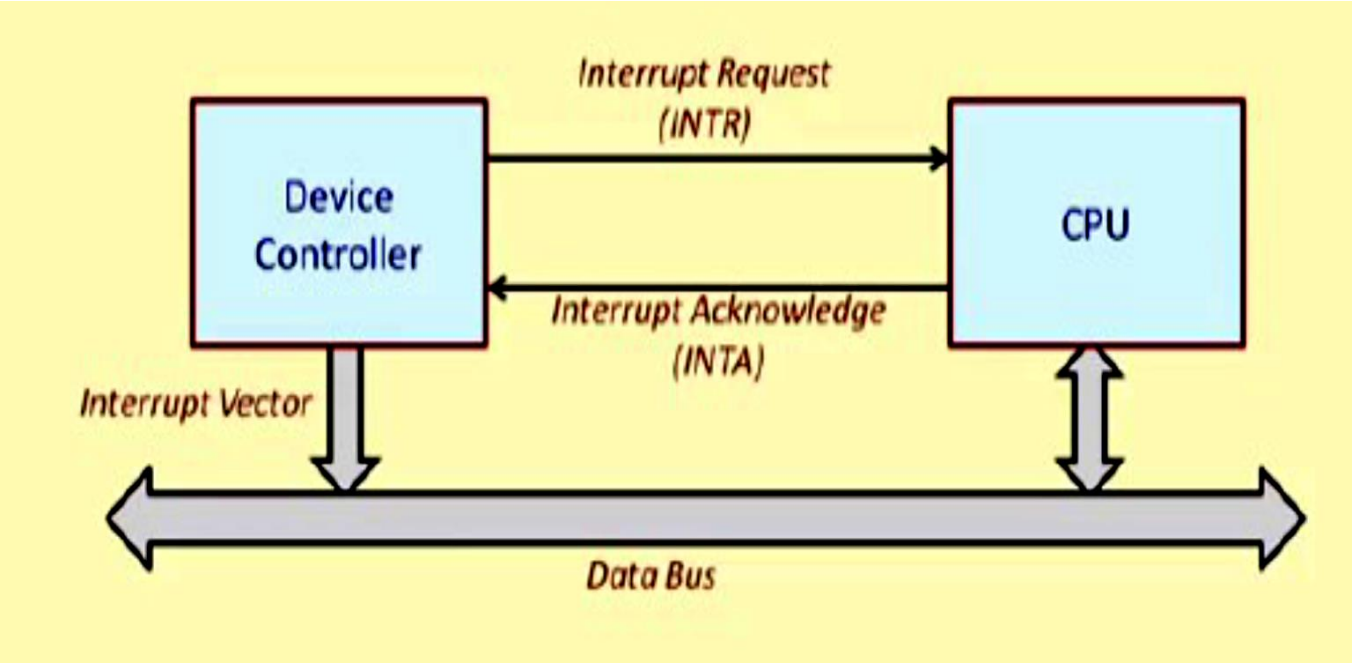
What happens when an interrupt request arrives?

- At the end of the current instruction execution, the PC and program status word(PSW) are saved in the stack automatically.
  - PSW contains status flags and other processor status information.
- The interrupt is acknowledged, the interrupt vector obtained , based on which control transfers to the appropriate ISR.
  - Different interrupting devices may have different ISR'S.
- After handling the interrupt, the ISR executes a special Return From Interrupt (RTI) instruction.
  - Restores the PSW and returns control to the saved PC address
  - Unlike normal RETURN where PSW is not restored .

General Interrupt Processing



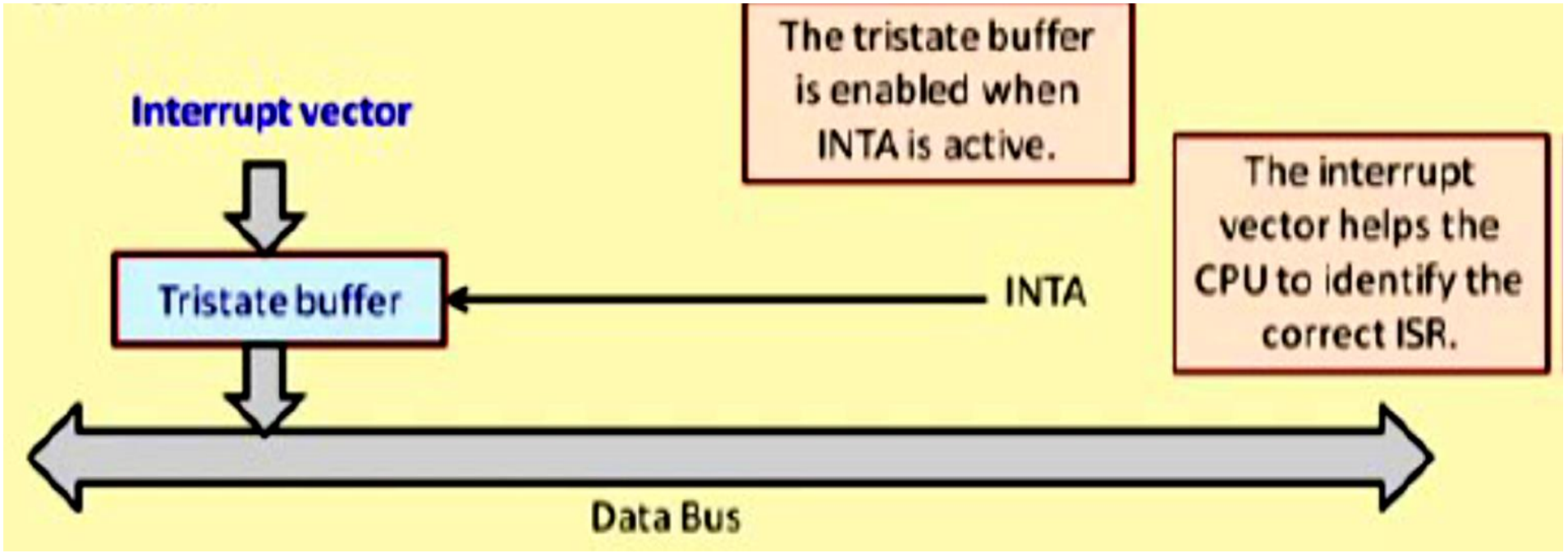
General Interrupt Processing



Steps

- Device controller sends INTR to the CPU.
- CPU finishes the current instruction and sends back INTA.
- Device controller sends interrupt vector over data bus
- CPU reads the interrupt vector, and identifies the device

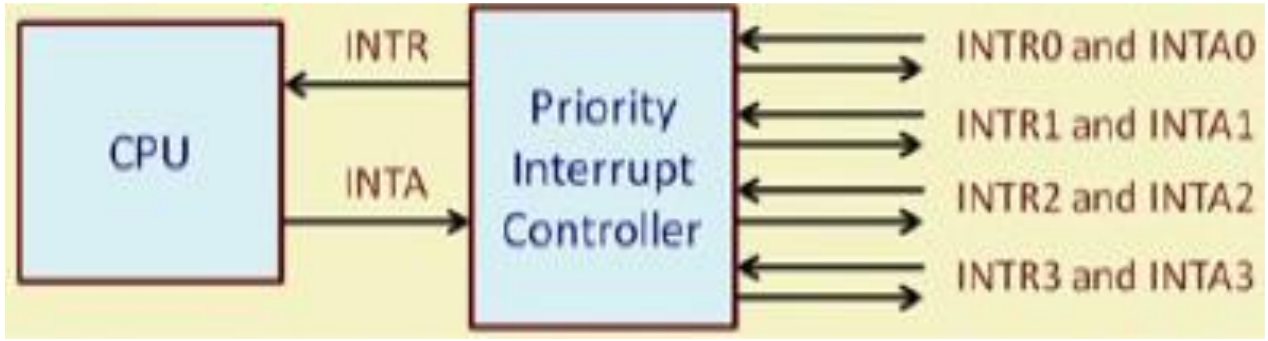
How is the interrupt vector sent on the data bus in response to INTA?



Multiple Devices Interrupting the CPU

A common solution is to use a priority interrupt controller.

- The interrupt controller interacts with CPU on the one side and multiple devices on the other side.
- For simultaneous interrupt requests, interrupt priority is defined.
- The interrupt controller is responsible for sending the interrupt vector to CPU.



How it works?

The INTR line is made active when some of the device(s) activate their interrupt request line.

INTR = INTRO + INTR1 + INTR2 + INTR3

- When the CPU sends back INTR, the interrupt controller sends back the corresponding acknowledge to the interrupting device, and puts the interrupt vector on the data bus.
- The interrupt controller is programmable, where the interrupt vectors for the various interrupts can be programmed (specified).
- For more than one interrupt request simultaneously active, a priority mechanism is used
  - E.g. INTRO is highest priority, followed by INTR1, etc.

How is interrupt nesting handled?

Consider the scenario:

- A device D0 has interrupted and the CPU is executing the ISR for D0
- In the mean time, another device D1 has interrupted.

Two possible scenarios here:

- D1 will interrupt the ISR for D0, get processed first, and then the ISP for D0 will be resumed
- Disable the interrupt system automatically whenever an interrupt is acknowledged so that handling of the nested interrupt is not required.

Typical instruction set architectures have the following instructions:

- EI : Enable interrupt
- DI : Disable interrupt

For the second scenario as discussed, the ISR will give an EI instruction just before RTI.

- Some ISA combine EI and RTI in a single instruction.

The DI instruction is sometimes used by the operating system to execute atomic code (e.g. semaphore wait and signal operations).

- Nobody should interrupt the code while it is being executed.

Cases that make interrupt handling difficult

For some interrupts, it is not possible to finish the execution of the current instruction.

- A special RETURN instruction is required that would return and restart the interrupted instructions.

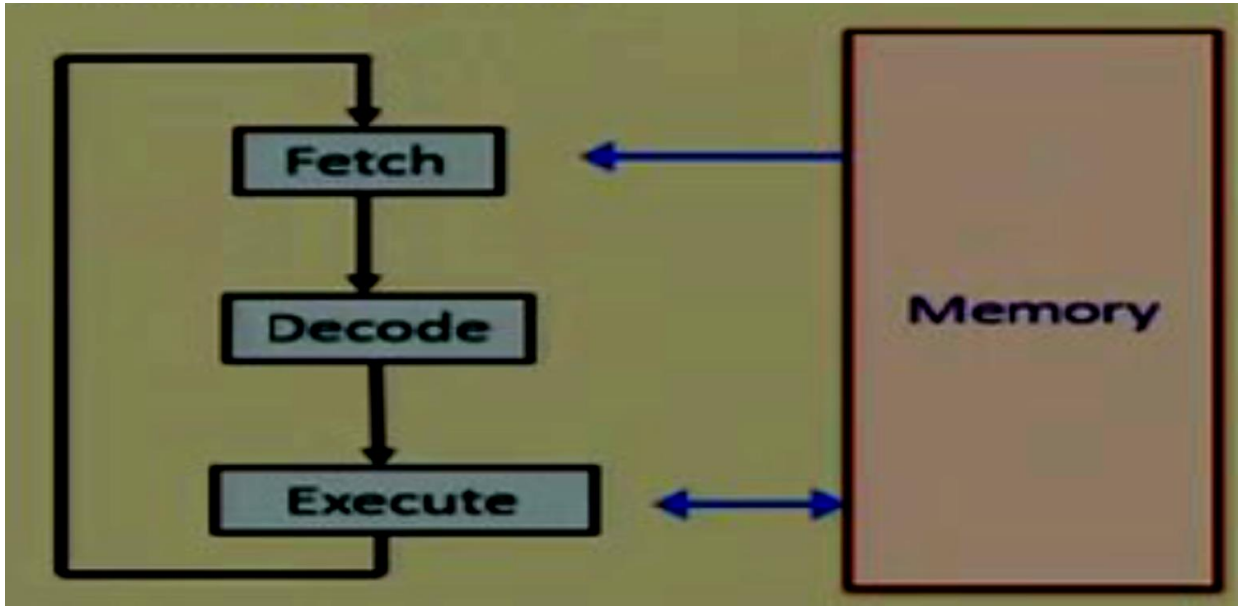
Some examples:

- Page fault interrupt: A memory location is being accessed that is not presently available in main memory.
- Arithmetic exception: Some error has occurred during some arithmetic operations (e.g. division by zero).

Bus Organization

How an instruction gets executed?

Fetch – Execute Cycle



Example

Example: Add R1, R2

Address	Instruction
1000	ADD R1, R2
1004	MUL R3, R4

a) PC = 1000
b) MAR = 1000
c) PC = PC + 4 = 1004
d) MDR = "ADD R1, R2"
e) IR = "ADD R1, R2"
(Decode and finally execute)
f) R1 = R1 + R2

Requirement for Instruction execution

- The necessary registers must be present.
- Internal organization of the registers must be known.
- The data path must be known
- For instruction execution a number of micro operations are carried out on the data path.
  - May involve movement of data

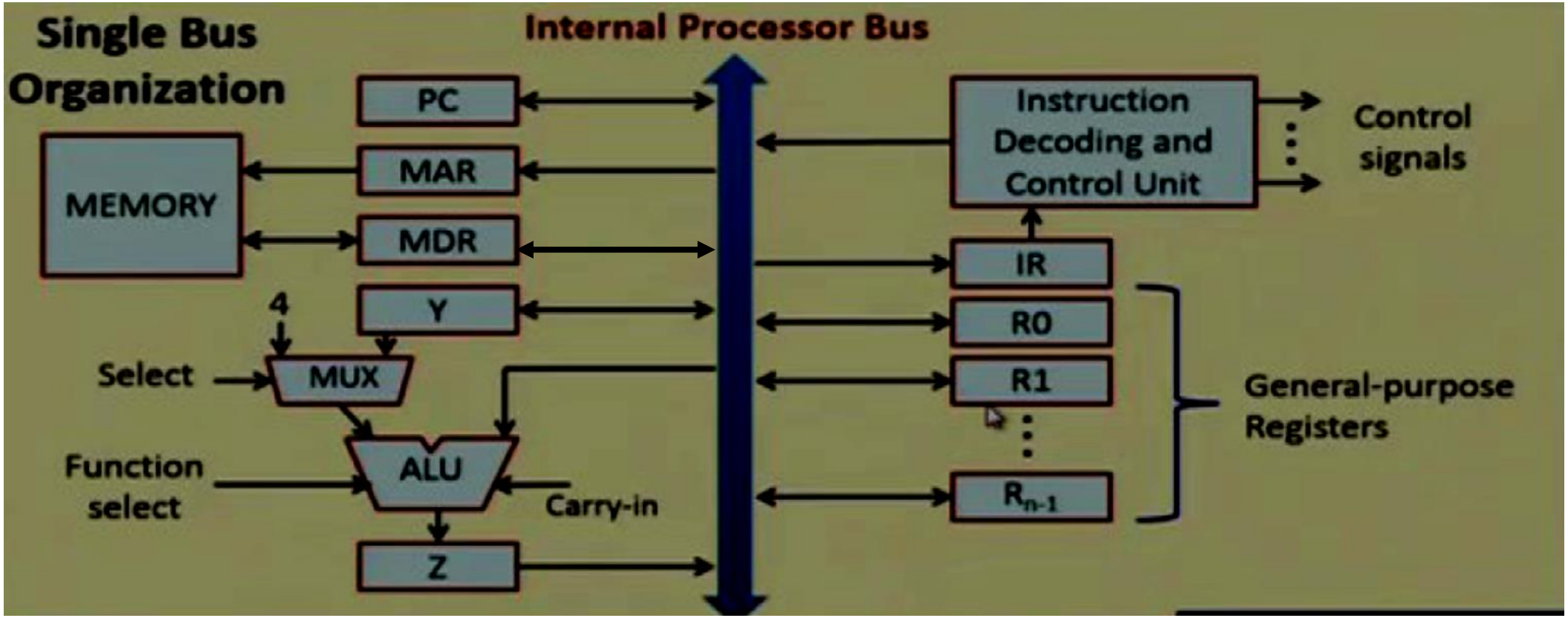
Kinds of data movement

Broadly it can be register to register, it can be register to ALU, or ALU to register.

Data movement are supported by the data path.

- The data path contains what the registers.
- The bus through which the data will move.
- The ALU, and of course, some of the temporary registers.

Single Bus Organization



Single Internal bus organization

- All the registers and various units are connected using a single internal bus.
- Registers are R0 to Rn-1. So, we have n general-purpose registers used for various purposes.
- Y and Z are used for storing intermediate results.
- The MUX selects either a constant 4 or the output of register Y.
- When PC is incremented a constant 4 has to be added.

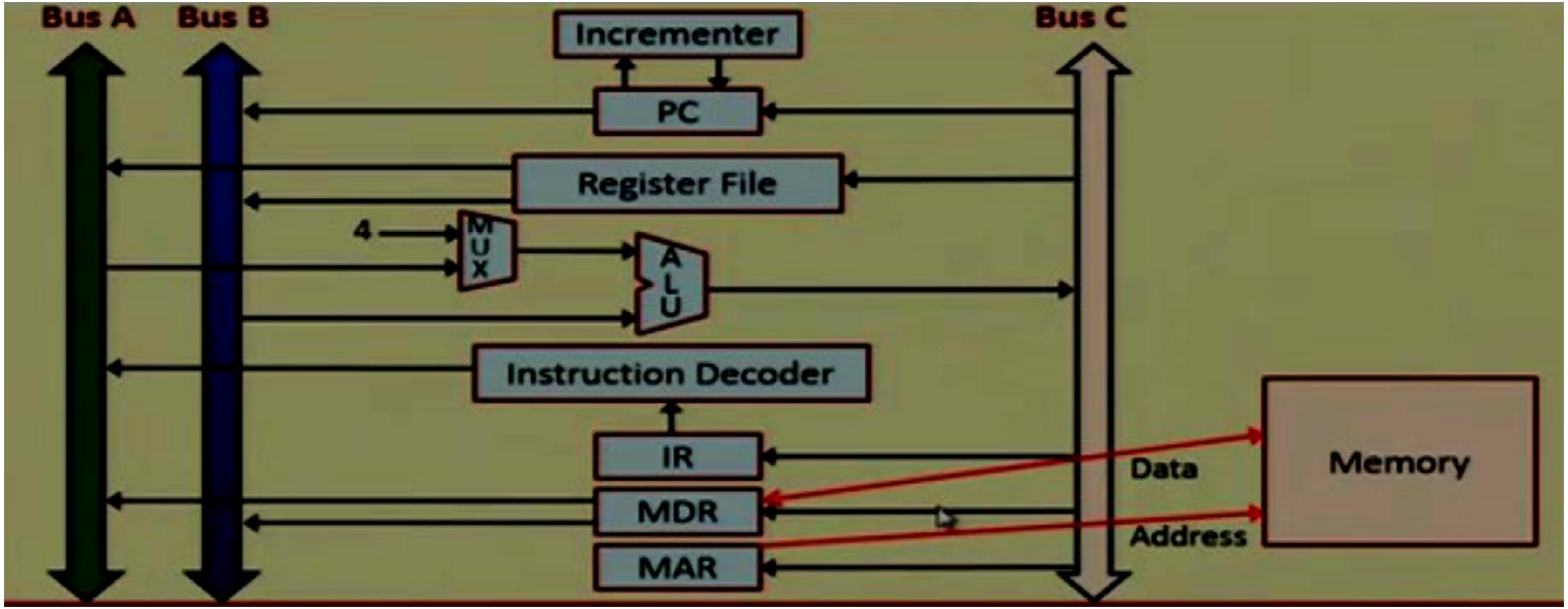
Single Internal bus organization

- The instruction decoder and control unit is responsible for performing the action specified by the instruction loaded into IR.
- Now once the instruction is fetched from the memory, it is it comes through MDR, and then it goes to IR using that single bus.
- Once it is loaded in IR, it is the responsibility of the decoder unit to decode that particular instruction.
  - And then it generates whatever needs to be done; if it has to bring the data from memory again it will do the required operation, if the data is already present in the processor register, then it has to add it or multiply it with whatever action is specified it needs to be done.

Single Internal bus organization

- The decoder generates all the control signals in proper sequence required to execute the instruction specified by IR.
- The registers, the ALU, and the interconnecting bus are collectively referred to as the **data path**

Three-bus organizations



Three-bus organizations..

A 3-bus organization is internal to the CPU

The 3 buses allow 3 parallel data transfer operations to be carried out.

- Less number of cycles in turn will be required to execute an instruction, compared to single bus organization.

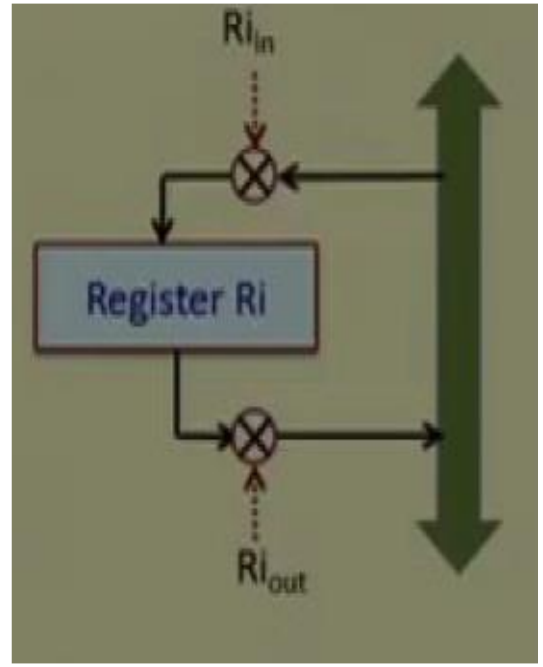
Organization of a Register

A register is used for temporary storage of data.

Ri typically has two control signals

- Riin: Used to load the register with data from the bus
- RiOut: Used to place the data stored in the register on the bus.

Input and output lines of the register Ri are connected to the bus via control switches.

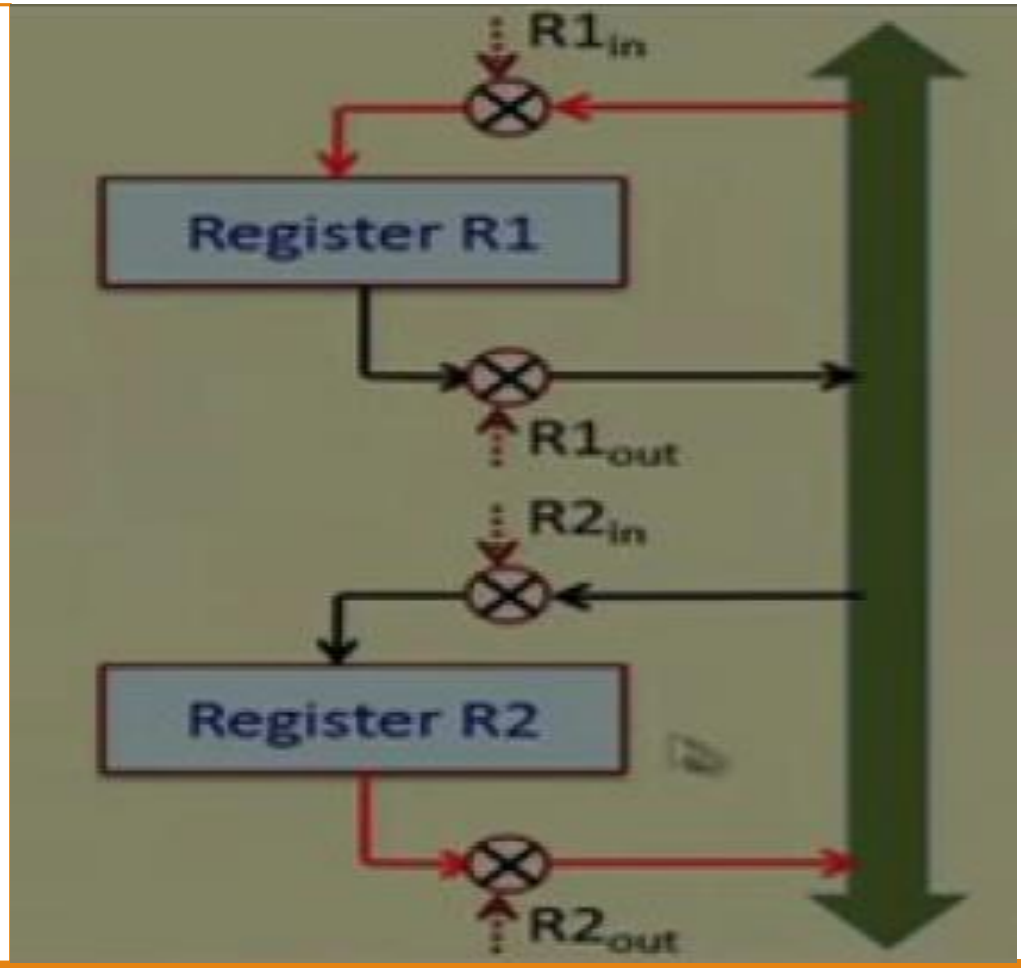


Register Transfer

MOVE R1,R2

- Enable the output of R2 by setting $R2_{out}$ =1
- Enable the input of the register R1 by setting $R1_{in}$=1
- All the operations are performed in synchronism with the processor clock.
  - The control signals are asserted at the start of the clock cycle.
  - After data transfer the control signals will return to 0
- We write as T1:$R2_{out}$, $R1_{in}$

Time Step Control Signal



ALU operation

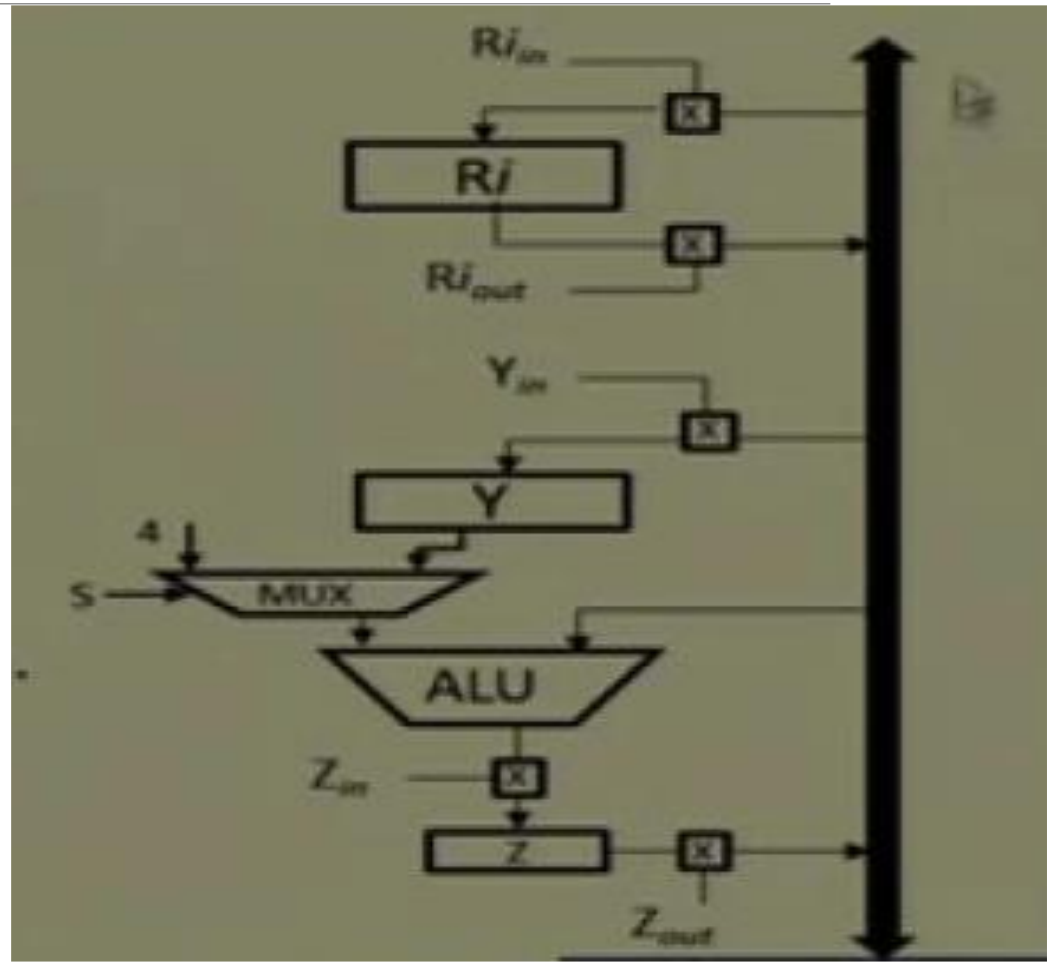
ADD R1, R2

- Bring the two operands (R1 and R2) to the two inputs of the ALU.
- One through Y (R1) and another (R2) directly from the internal bus.
- Result is stored in Z and finally transferred to R1.

T1: R1out, Yin

T2: R2out, SelectY, ADD, Zin

T3: Zout, R1in



Fetch a word from the memory

These are the steps that are involved to fetch a word from memory.

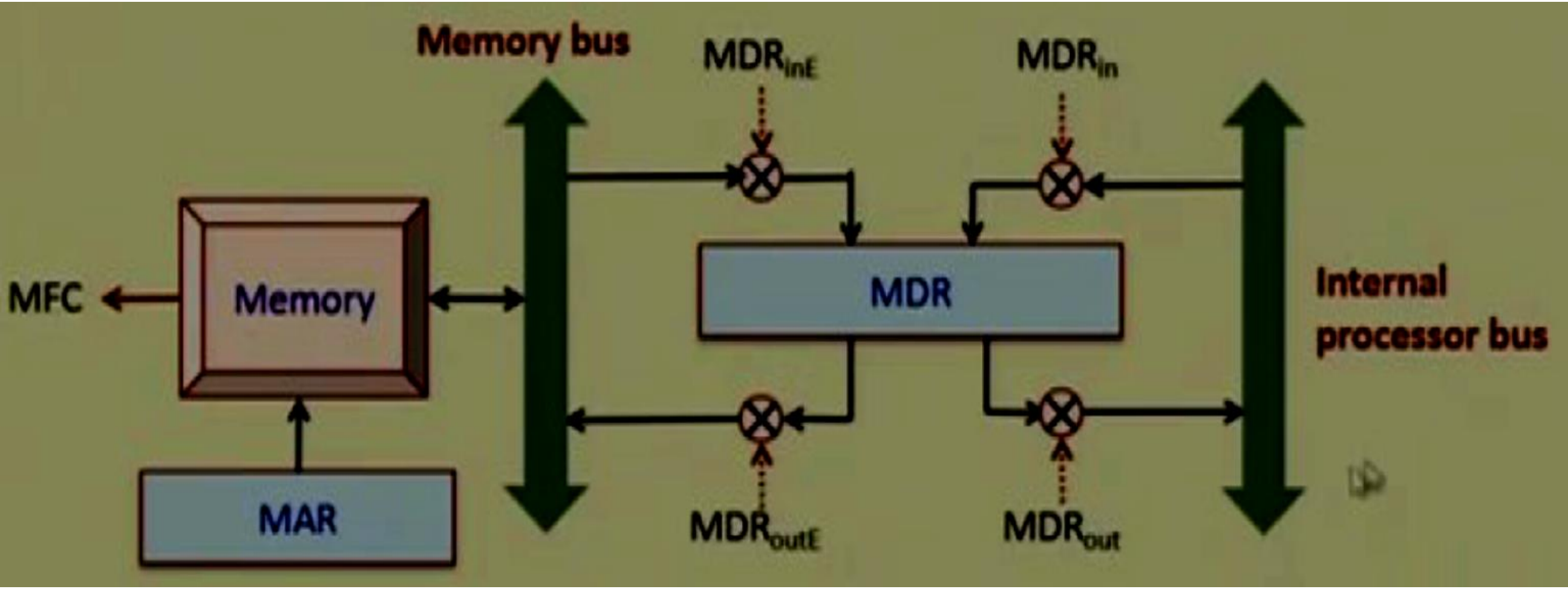
- The processor specifies the address of the memory location where the data or instruction is stored.
- The processor request for a read operation.
- The information to be fetched can be either an instruction or it can be a data, accordingly it is fetched.
- The data read is brought from memory to MDR.
- Then it can be transfer it to the required register or ALU.

Storing a word into memory

The steps involved to store a word into the memory:

- The processor specifies the address of the memory location where the data is to be written.
- The data to be written in loaded into MDR.
- The processor requests a write operation.
- The content of MDR will be written to the specified memory location.

Connecting MDR to Memory Bus and Internal Bus



Memory Read Write Operation

- The address of the memory location is transferred to MAR
- At the same time the read or write control signal is provided to indicate the operation
- For read the data, from memory data bus comes to MDR by activating MDRinE
- For write the data from MDR goes to data bus by activating the signal MDRoutE

Memory Read Write Operation

- When the processor sends a read request it has to wait until the data is read from the memory and return into MDR
- To accommodate this variability in response time as there is no fixed response time that how much time it will be required, the processor has to wait until it receives an indication from memory that the read operation has been completed.
- A control signals known as Memory Function Complete (MFC) is used
  - when this signal is 1, it indicates that the content of specified location is read and are available on the data lines of the memory bus
  - Then the data can be made available to MDR because it is directly connected to MDR from the data lines of memory.

Fetch a word: MOVE R1, (R2)

1. MAR ← R2
2. Start a Read operation on the memory bus
3. Wait for the MFC response from the memory
4. Load MDR from the memory
5. R1 ← MDR

Control steps:

a) $R2_{out}$, MAR_{in}, Read

b) MDR_{inE}, WMFC

c) MDR_{out}, $R1_{in}$

Store a word: MOVE (R1), R2

1. MAR ← R1
2. MDR ← R2
3. Start a Write operation on the memory bus
4. Wait for the MFC response from the memory

Control steps:

a) $R1_{out}$, MAR_{in}

b) $R2_{out}$, MDR_{in}, Write

c) MDR_{outE}, WMFC

Execution of a Complete Instruction

ADD R1, R2

T1: Pcout, MARin, Read, Select4, ADD,Zin

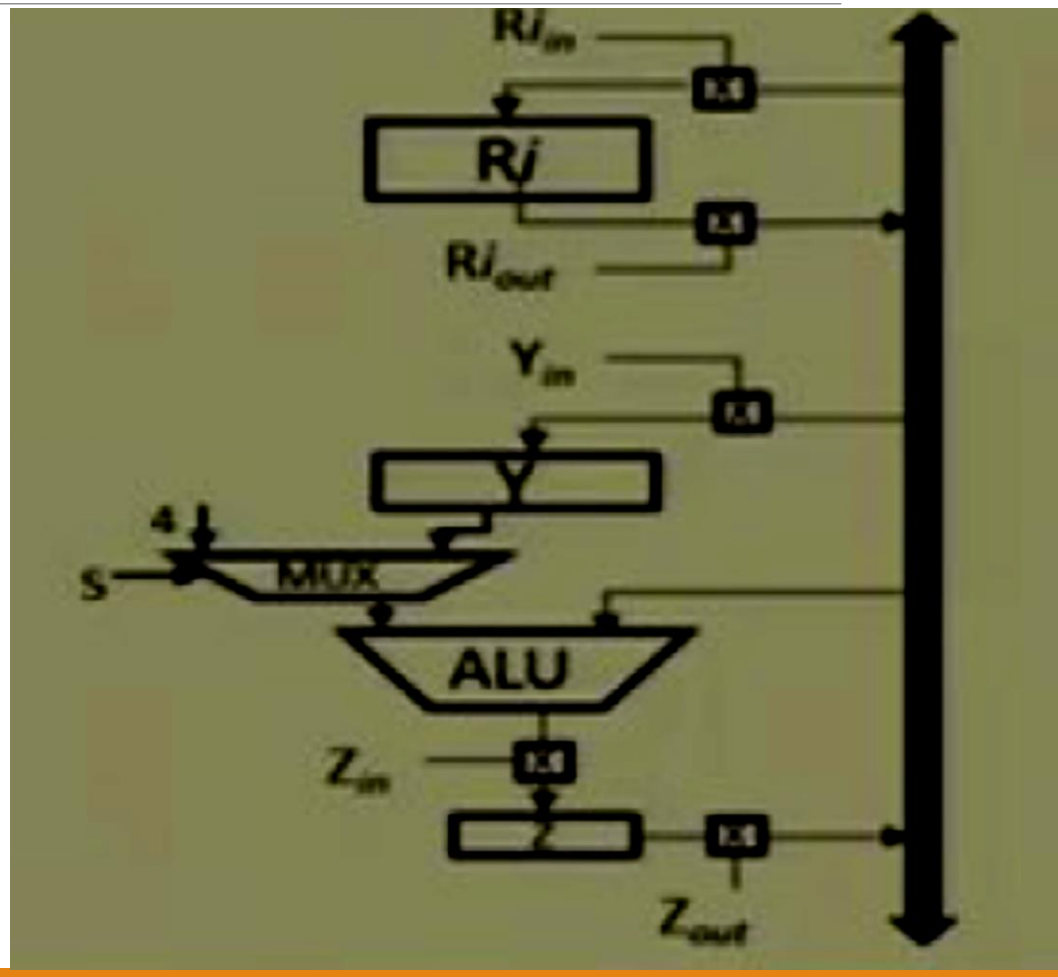
T2: Zout, Pcin, Yin, WMFC

T3: MDRout, Irin

T4: R1out, Yin, SelectY

T5: R2out, ADD, Zin

T6: Zout, R1in



MOVE R1,R2

Steps	Action
1	PC_{out}, MAR_{in}, Read, Select4, Add, Z_{in}
2	Z_{out}, PC_{in}, Y_{in}, WMFC
3	MDR_{out}, IR_{in}
4	$R2_{out}$, $R1_{in}$, END

Load R1,LOCA

Steps	Action
1	PC_{out}, MAR_{in}, Read, Select4, Add, Z_{in}
2	Z_{out}, PC_{in}, Y_{in}, WMFC
3	MDR_{out}, IR_{in}
4	Address field of IRout, MAR_{in}, Read
5	WMFC
6	MDR_{out}, $R1_{in}$, END

Example for a Three Bus Organization

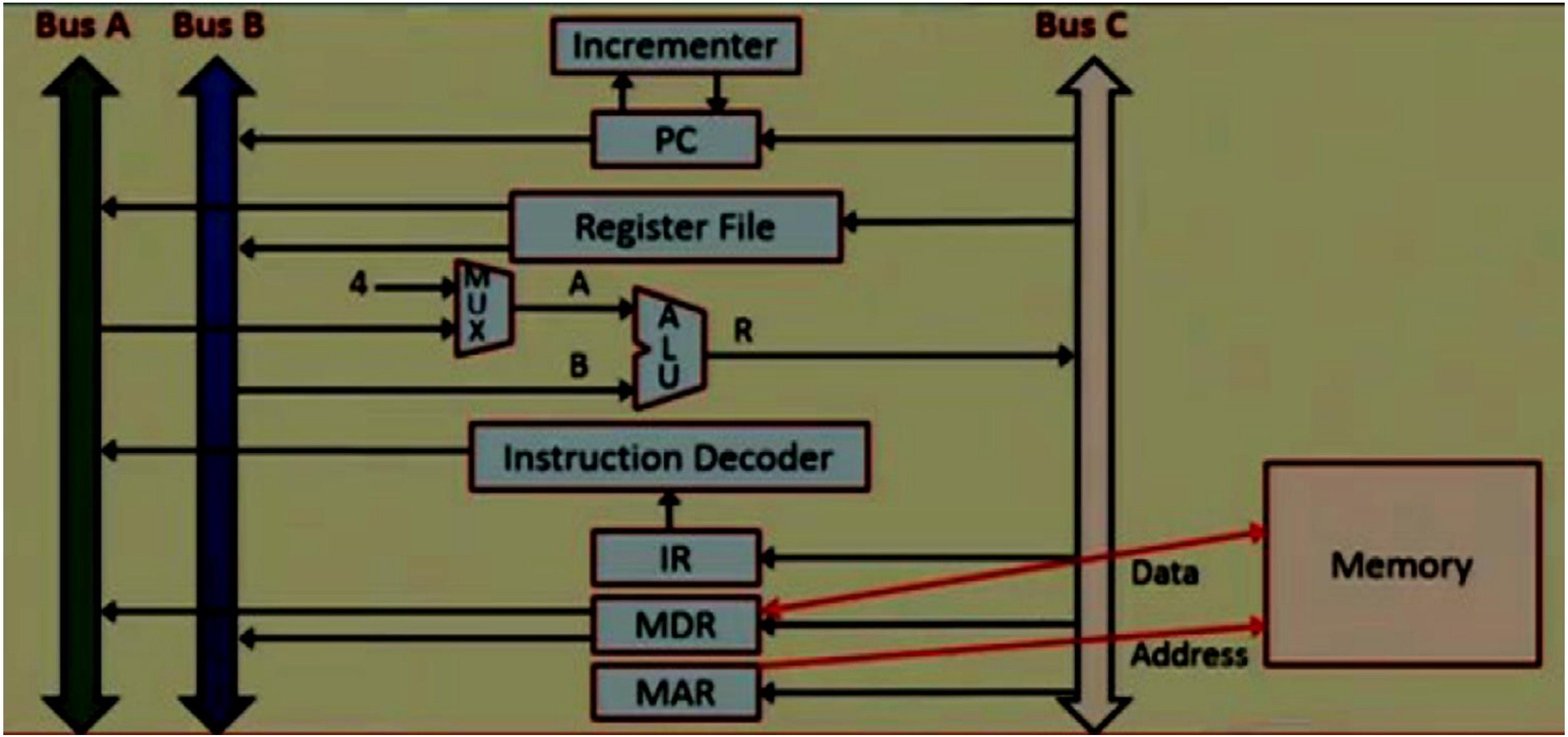
SUB R1, R2, R3

T1: Pcout, R=B, MARin, READ, IncPC

T2: WMFC

T3: MDRoutB, R=B, Irin

T4: R2outA, R3outB, SelectA, SUB, R1in, End


Bus A
Bus B
Bus C
Incrementer
PC
Register File
4
MUX
A
B
ALU
R
Instruction Decoder
IR
MDR
MAR
Data
Address
Memory

Example

ADD R1, LOCA (R1 = R1 + Mem[LOCA])

Steps	Action
1	PC_{out}, MAR_{in}, Read, Select4, Add, Z_{in}
2	Z_{out}, PC_{in}, Y_{in}, WMFC
3	MDR_{out}, IR_{in}
4	Address field of IRout, MAR_{in}, Read
5	$R1_{out}$, Y_{in}, WMFC
6	MDR_{out}, SelectY, Add, Z_{in}
7	Z_{out}, $R1_{in}$, End

Control Unit Design approaches

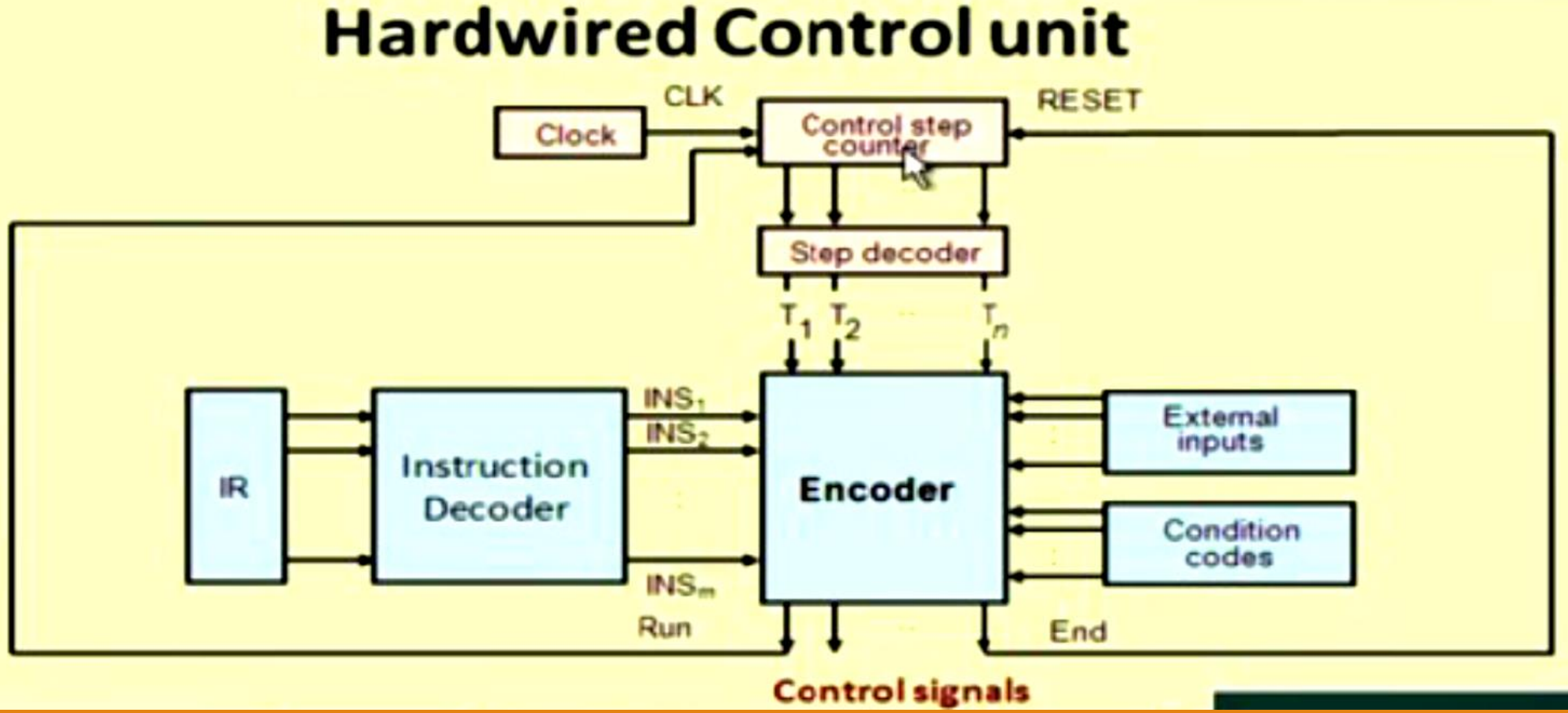
To execute an instruction, the processor must generate control signals for the data path in proper sequence.

- Example: ADD R1, R2
    1. R1out, Yin, SelectY
    2. R2out, ADD, Zin
    3. Zout, R1in

Two Alternate approaches

1. Hardwired control unit design
2. Microprogrammed control unit design

Hardwired Control Unit



Hardwired Control Unit

- Assumption
  - Each step in the sequence is completed in one clock cycle
    - But remember one thing that when we are reading something from the memory it may not be completed in one clock cycle.
- A counter is used to keep track of the time step.
- The control signals are determined by the following information:
  - Content of the control step
  - Counter content of the instruction register
  - Content of conditional code flags and external inputs such as MFC (Memory Function Complete)

Hardwired Control Unit…

- The encoder-decoder circuit is a combinational circuit that generates the control signals depending on the inputs provided.
- The step decoder generates separate signal line for each step in the control sequence (T1, T2, T3 etc.)
  - Depending on the maximum steps required for an instruction, the step decoder is designed
  - If the maximum step is 10 in that case step decoder size will be 4 x 16
- Among the total set of instructions, the instruction decoder is used to select one of them. That particular line will be 1 and the rest will be 0
  - If a maximum of 100 instructions are present in the ISA then a 7 x 128 instruction decoder is used.

Hardwired Control Unit…

- At every clock the RUN signal is used to increment the counter by one.
  - When RUN is 0 the counter stops counting.
  - This signal is needed when WMFC is issued.
- END signal starts a new instruction.
  - It resets the control step counter to its starting value.
- The sequence of the operations carried out by the control unit is determined by the wiring of the logic elements and hence it is named **hardwired**
- This approach of control unit design is fast but limited to the complexity of the instruction set that is implemented.

Generations of Control Signals

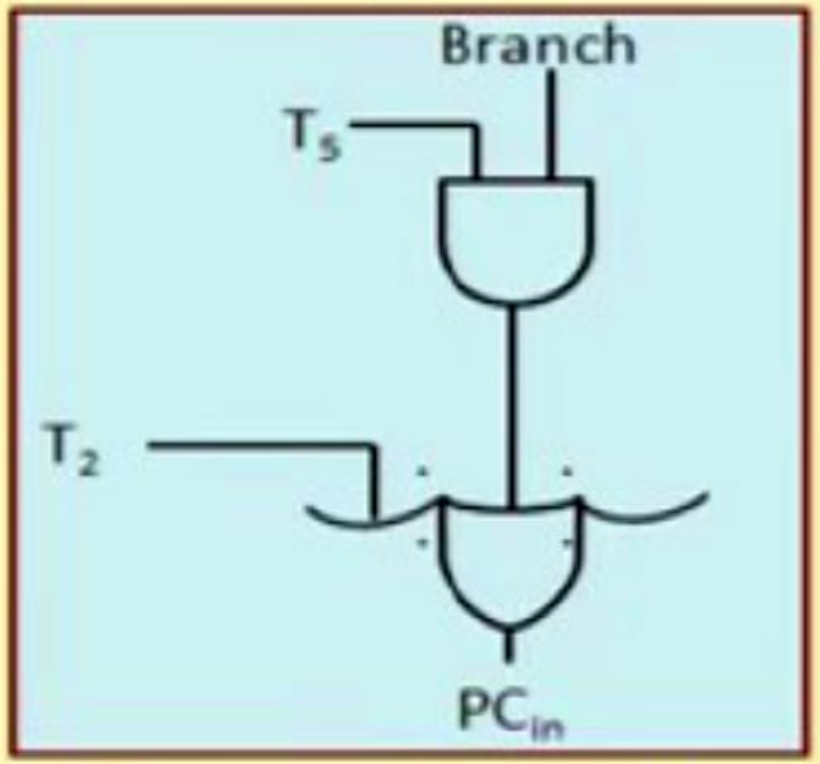
	ADD R1, R2
1	PC_{out}, MAR_{in}, Read, Select4, Add, Z_{in}
2	Z_{out}, PC_{in}, Y_{in}, WMFC
3	MDR_{out}, IR_{in}
4	$R1_{out}$, Y_{in}
5	$R2_{out}$, SelectY, Add, Z_{in}
6	Z_{out}, $R1_{in}$, End

	ADD R1, LOCA
1	PC_{out}, MAR_{in}, Read, Select4, Add, Z_{in}
2	Z_{out}, PC_{in}, Y_{in}, WMFC
3	MDR_{out}, IR_{in}
4	Address field of IRout, MAR_{in}, Read
5	$R1_{out}$, Y_{in}, WMFC
6	MDR_{out}, SelectY, Add, Z_{in}
7	Z_{out}, $R1_{in}$, End

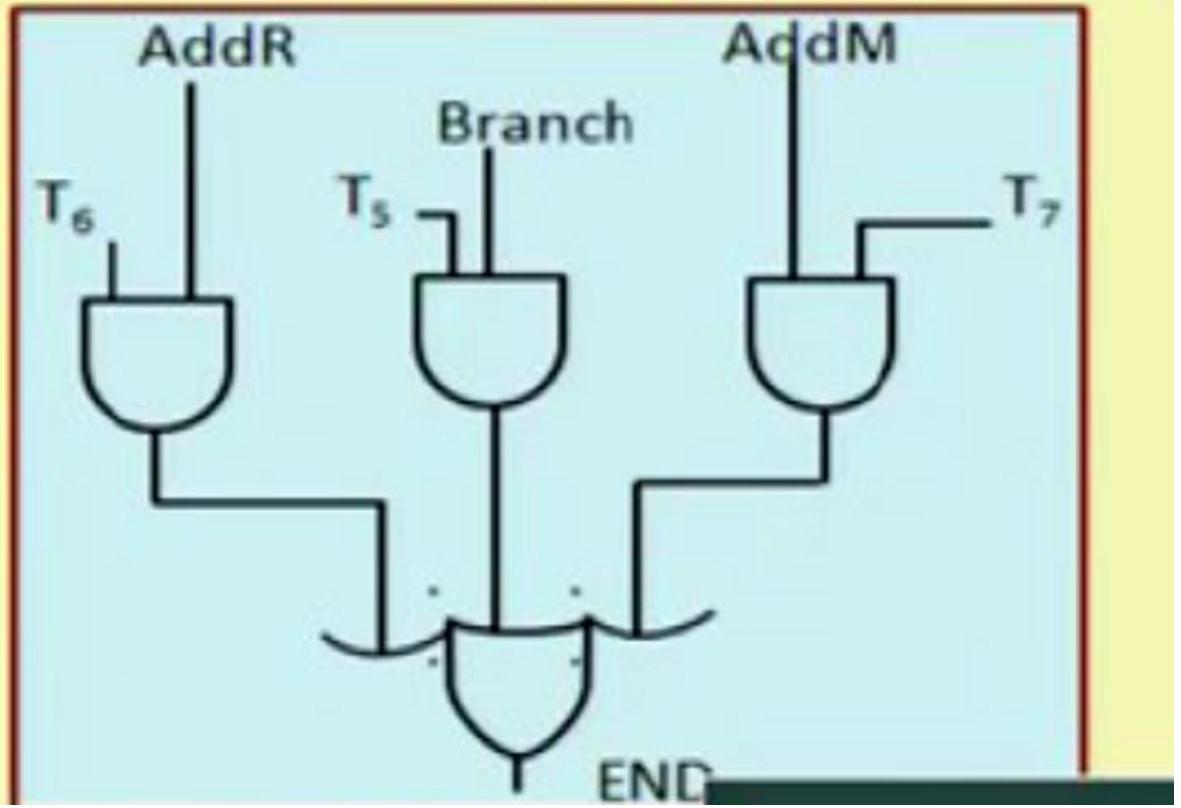
	BRANCH Label
1	PC_{out}, MAR_{in}, Read, Select4, Add, Z_{in}
2	Z_{out}, PC_{in}, Y_{in}, WMFC
3	MDR_{out}, IR_{in}
4	Offset-field-of-IR_{out}, SelectY, Add, Z_{in}
5	Z_{out}, PC_{in}, End

Generation of Pc_{in} and END

$PC_{in} = T_2 + T_5.Branch$



$END = T_6.ADDR + T_5.Branch + T_7.AddM$



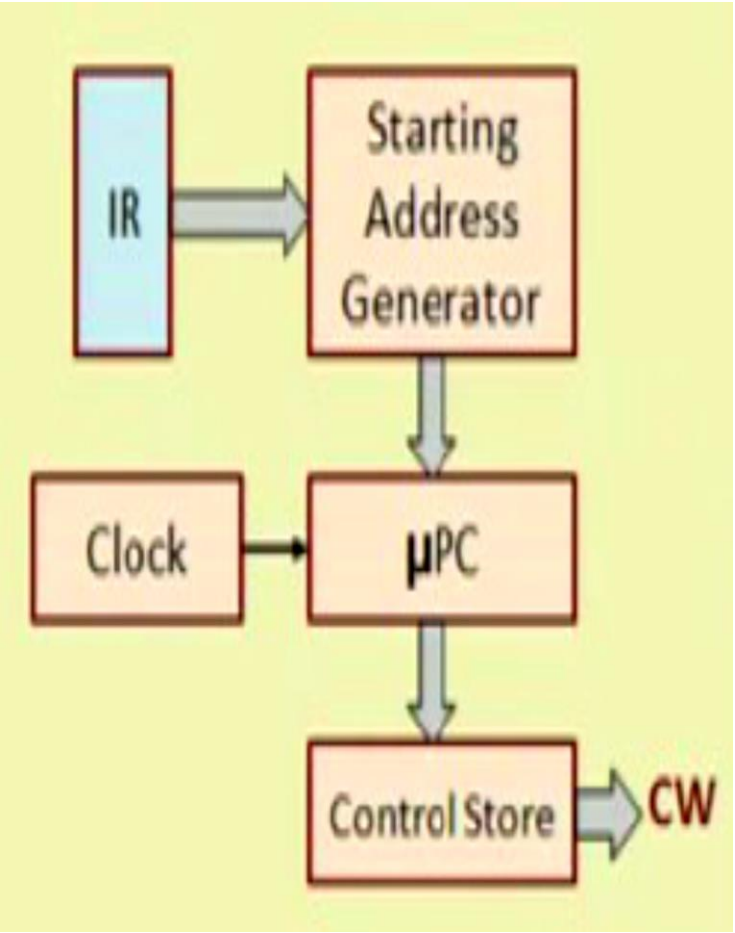
Exercise

- Give the control sequence for the execution of the instruction Add ,R1,[R3]
- Generation of Z_{in} control signal for the processor

Zin = T1 + T5.ADDR + T6.ADDM + T4.Branch + T6.ADDR3

Microprogrammed control unit design

- Control signals are generated by a program similar to machine language program.
- The **Control Store** (CS) stores the microroutines for all instructions of an ISA.
- The sequence of the steps corresponding to the control sequence of a machine instruction is the **microroutine**.
- Each sequence of steps is a **control word** (CW) whose individual bits represent the various control signals.
- Individual control words in a microroutine are called **microinstructions**



Microprogrammed control unit design

- Control-unit generates the control signals for an instruction by sequentially reading CWs of corresponding micro routine from CS.
- The µPC is used to read CWs sequentially from CS
- Every time a new instruction is loaded into IR, output of Starting Address Generator is loaded into µPC.
- Then, µPC is automatically incremented by clock causing successive microintructions to be read from CS

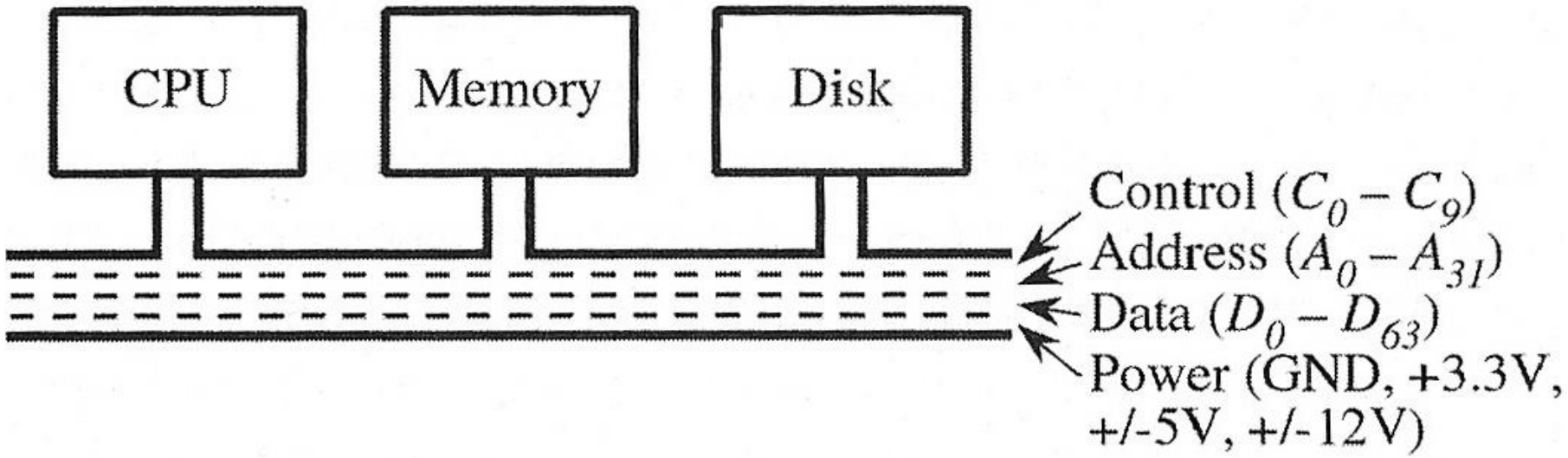
ADD R1, R2
T1: Pcout, MARin, Read, Select4, ADD,Zin
T2: Zout, Pcin, Yin, WMFC
T3: MDRout, Irin
T4: R1out, Yin, SelectY
T5: R2out, ADD, Zin
T6: Zout, R1in

Example for Microinstructions

Micro-instr.	...	PC_{in}	PC_{out}	MAR_{in}	Read	MDR_{out}	IR_{in}	Y_{in}	Select	Add	Z_{in}	Z_{out}	$R1_{out}$	$R1_{in}$	$R2_{out}$	WMFC	End	...
1	0	0	1	1	1	0	0	0	1	1	1	0	0	0	0	0	0	0
2	0	1	0	0	0	0	0	1	0	0	0	1	0	0	0	1	0	0
3	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	1	0	0	0	0	1	0	0	0	0	0
5	0	0	0	0	0	0	0	0	0	1	1	0	0	0	1	0	0	0
6	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	0	1	0

Bus

A bus is a collection on of wires and connectors through which the data is transmitted.

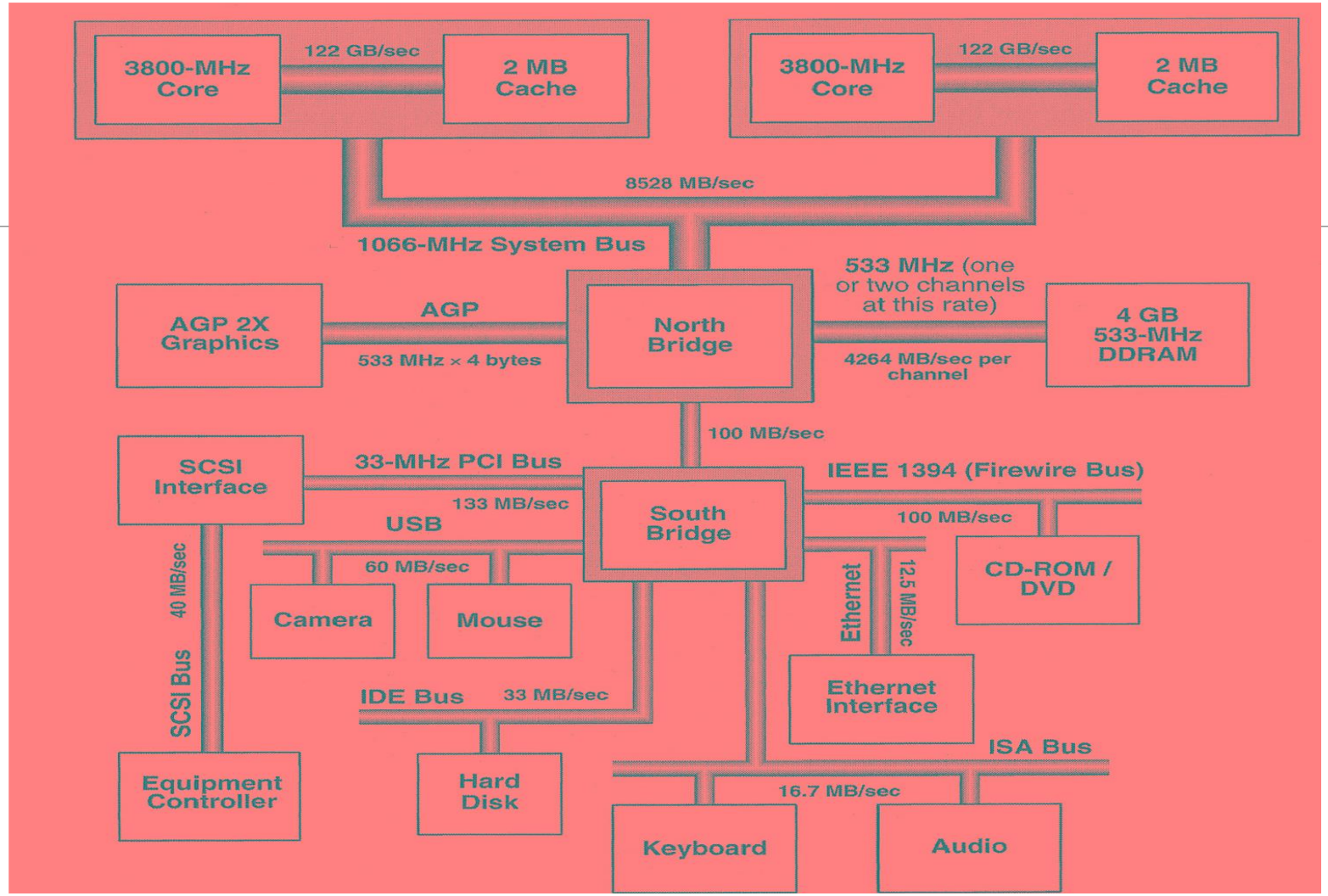


Bus Standard

- Bus Protocol
  - Rules determining the format and transmission of data through the bus
- Parallel Bus
  - Data transmitted in parallel
  - Advantage: It is fast
  - Disadvantage: High cost for long distance communication, inter-line interference at high frequency
- Serial Bus
  - Data transmitted serially
  - Advantage: Low cost for long distance communication, no interference
  - Disadvantage: Slow

Some Terminologies

- Bus Master and Slaves- The device that controls the bus is master, others are slave
- Local or System Bus- Bus that connects CPU and memory
- Front Side Bus- Original Concept connects CPU to components
  - Modern intel architecture connects CPU to NorthBridge chipset
- Back Side Bus-Connects CPU to L2 cache
- Memory Bus-Connects North Bridge chipset to memory
- AGP Bus-Connects North Bidge chipset to the GPU
- ISA,PCI,Firewire,USB,PCI-Express Bus:
  -Connects motherboard to peripherals
- Bus width- Number of wires available for transferring data
- Bus bandwidth-Total amount of data that can be transferred over the bus per unit time



Source: Intel Corp.

Synchronous Versus Asynchronous Bus

- Synchronous Bus: There is a common clock between the sender and the receiver that synchronizes bus operation
- Asynchronous Bus:
  - There is no common clock
  - Bus master and slave have to handshake during the process of communication

Synchronous Bus

- On a synchronous bus, all devices derive timing information from a control line called the bus clock
- The signal on this line has two phases: a high level followed by a low level
- The two phases constitute a clock cycle. The first half of the cycle between the low-to-high and high-to-low transitions is often referred to as a clock pulse
- The address and data lines in the Figure (next slide) are shown as if they are carrying both high and low signal levels at the same time
- This is a common convention for indicating that some lines are high and some low, depending on the particular address or data values being transmitted
- The crossing points indicate the times at which these patterns change
- A signal line at a level half-way between the low and high signal levels indicates periods during which the signal is unreliable, and must be ignored by all devices.

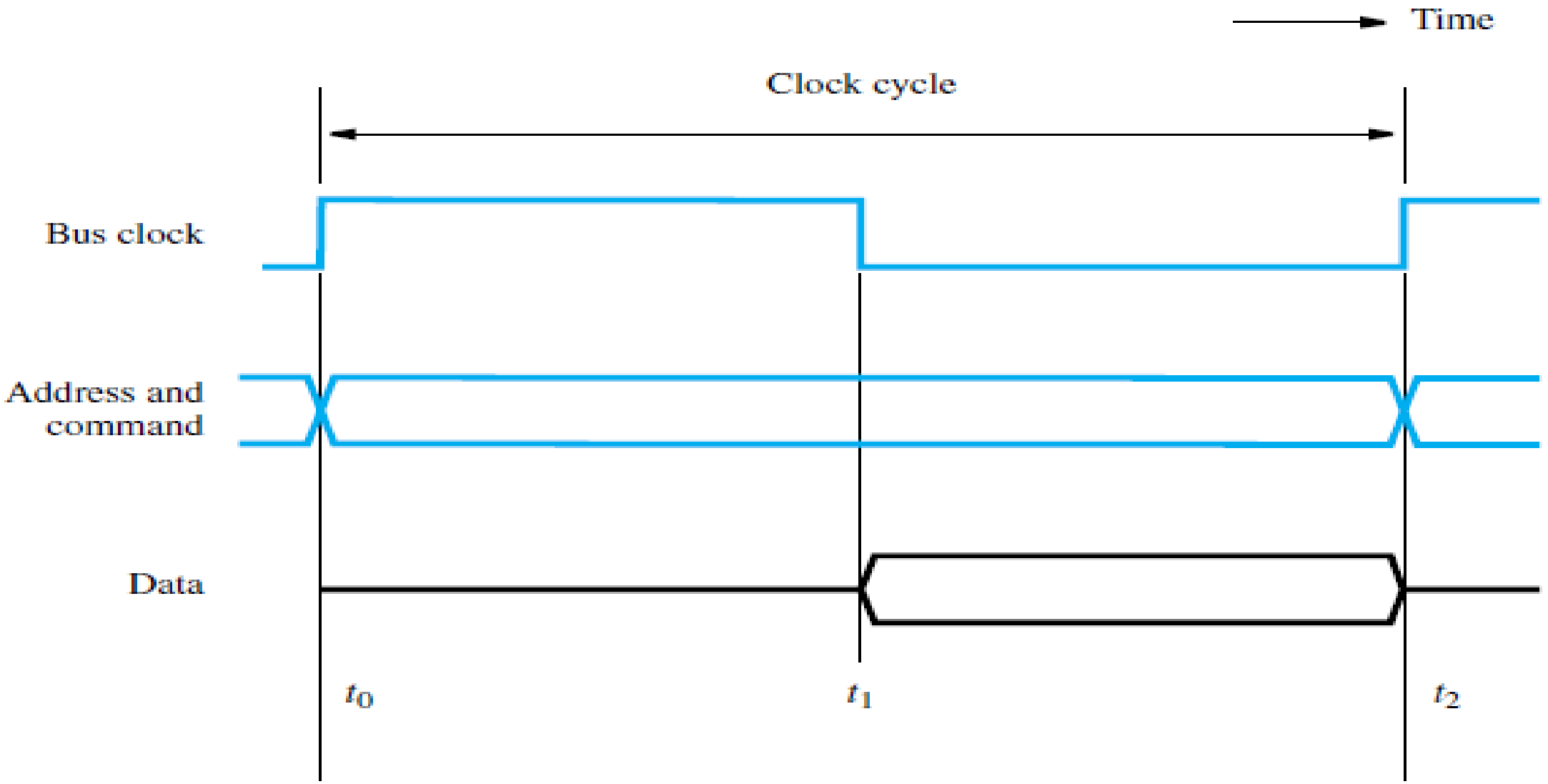


Figure:Timing of an input transfer on synchronous bus

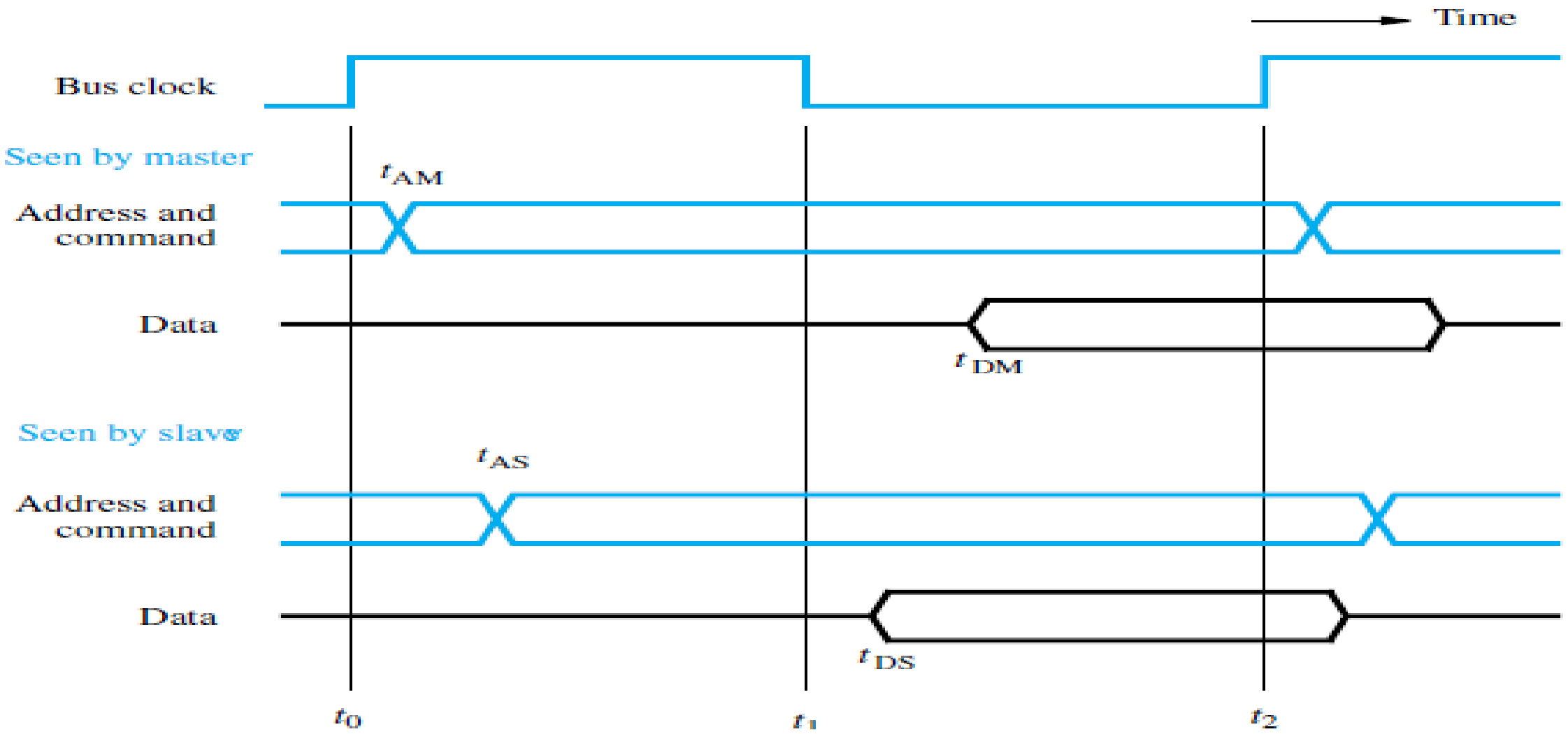
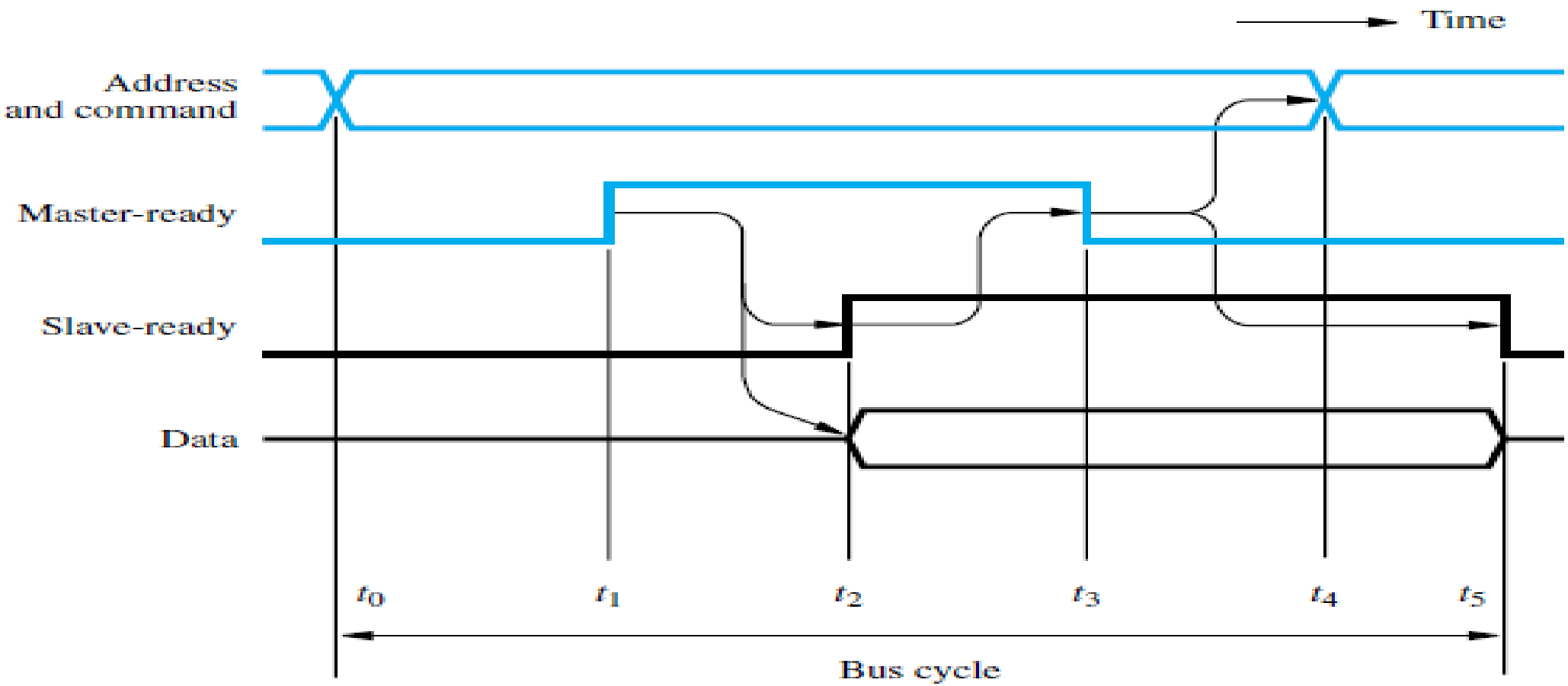


Figure: A detailed timing diagram for the input transfer

Asynchronous Bus

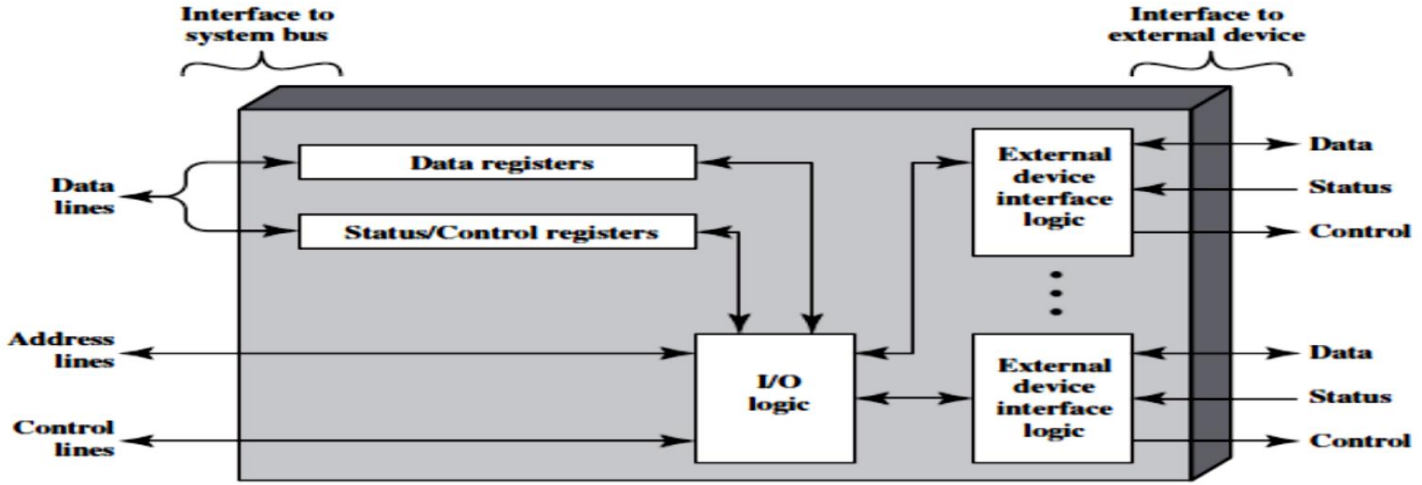
- An alternative scheme for controlling data transfers on a bus is based on the use of a handshake protocol between the master and the slave
- A handshake is an exchange of command and response signals between the master and the slave
- It is a generalization of the way the Slave-ready signal is used .
- A control line called Master-ready is asserted by the master to indicate that it is ready to start a data transfer.
- The Slave responds by asserting Slave-ready.

Asynchronous Bus



Interface Circuits

- The I/O interface of a device consists of the circuitry needed to connect that device to the bus.
- On one side of the interface are the bus lines for address, data, and control.
- On the other side are the connections needed to transfer data between the interface and the I/O device.
  - This side is called a port, and it can be either **a parallel or a serial port**.



Interface Circuits

- A parallel port transfers multiple bits of data simultaneously to or from the device.
- A serial port sends and receives data one bit at a time.
- Communication with the processor is the same for both formats; the conversion from a parallel to a serial format and vice versa takes place inside the interface circuit.

Interface Circuits

Let us recall the functions of an I/O interface

1. Provides a register for temporary storage of data
2. Includes a status register containing status information that can be accessed by the processor
3. Includes a control register that holds the information governing the behavior of the interface
4. Contains address-decoding circuitry to determine when it is being addressed by the processor
5. Generates the required timing signals
6. Performs any format conversion that may be necessary to transfer data between the processor and the I/O device, such as parallel-to-serial conversion in the case of a serial port

Parallel Port

- First, we describe an interface circuit for an 8-bit input port that can be used for connecting a simple input device, such as a keyboard.

- Then, we describe an interface circuit for an 8-bit output port, which can be used with an output device such as a display.

Keyboard to processor connection

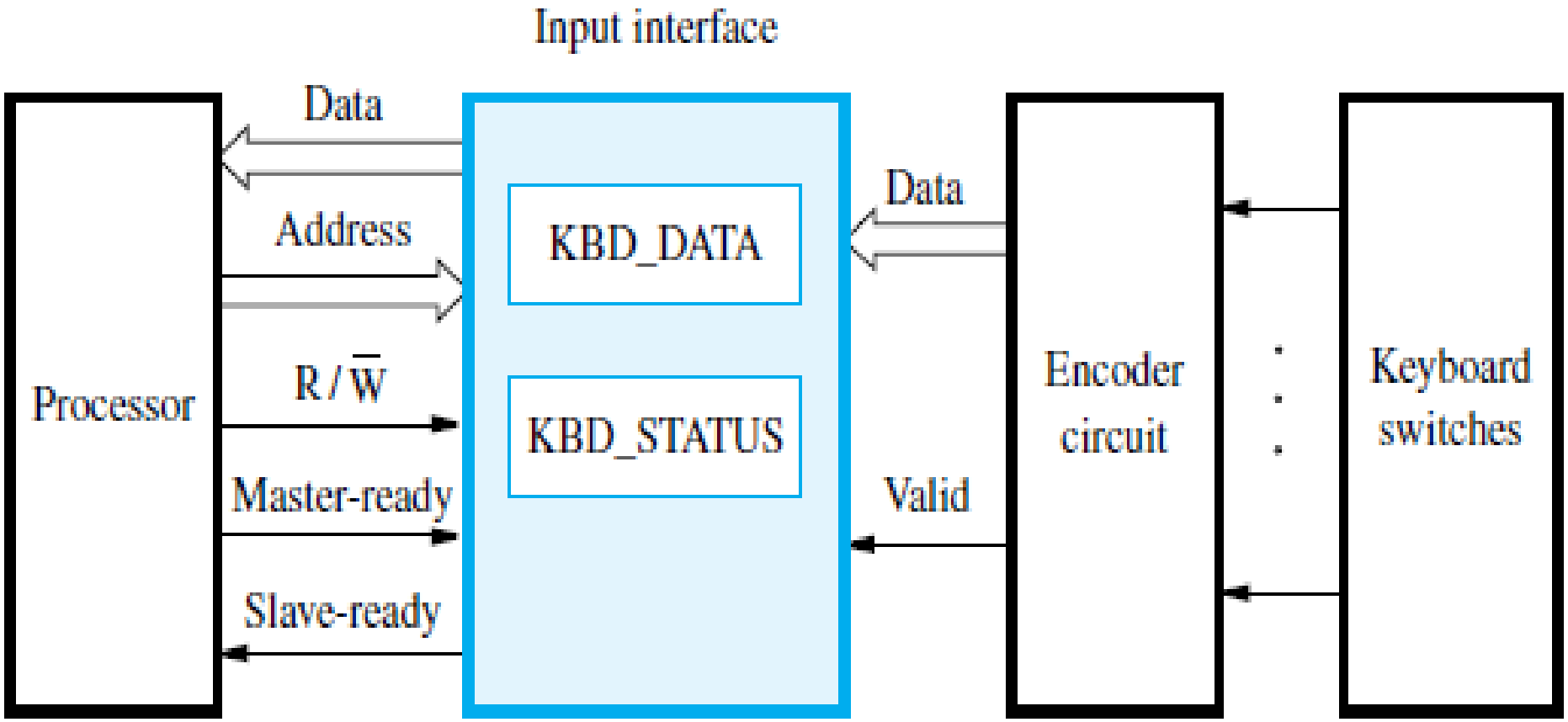


Figure 4.10:Keyboard to processor connection

Keyboard to processor connection

- Figure shows a circuit that can be used to connect a keyboard to a processor
- Assume that interrupts are not used, so there is no need for a control register.
- There are only two registers: a data register, KBD_DATA, and a status register, KBD_STATUS. The latter contains the keyboard status flag, KIN.

Keyboard to processor connection

- A typical keyboard consists of mechanical switches that are normally open.
- When a key is pressed, its switch closes and establishes a path for an electrical signal
- This signal is detected by an encoder circuit that generates the ASCII code for the corresponding character
- A difficulty with such mechanical pushbutton switches is that the contacts *bounce* when a key is pressed, resulting in the electrical connection being made then broken several times before the switch settles in the closed position
- Although bouncing may last only one or two milliseconds, this is long enough for the computer to erroneously interpret a single pressing of a key as the key being pressed and released several times.

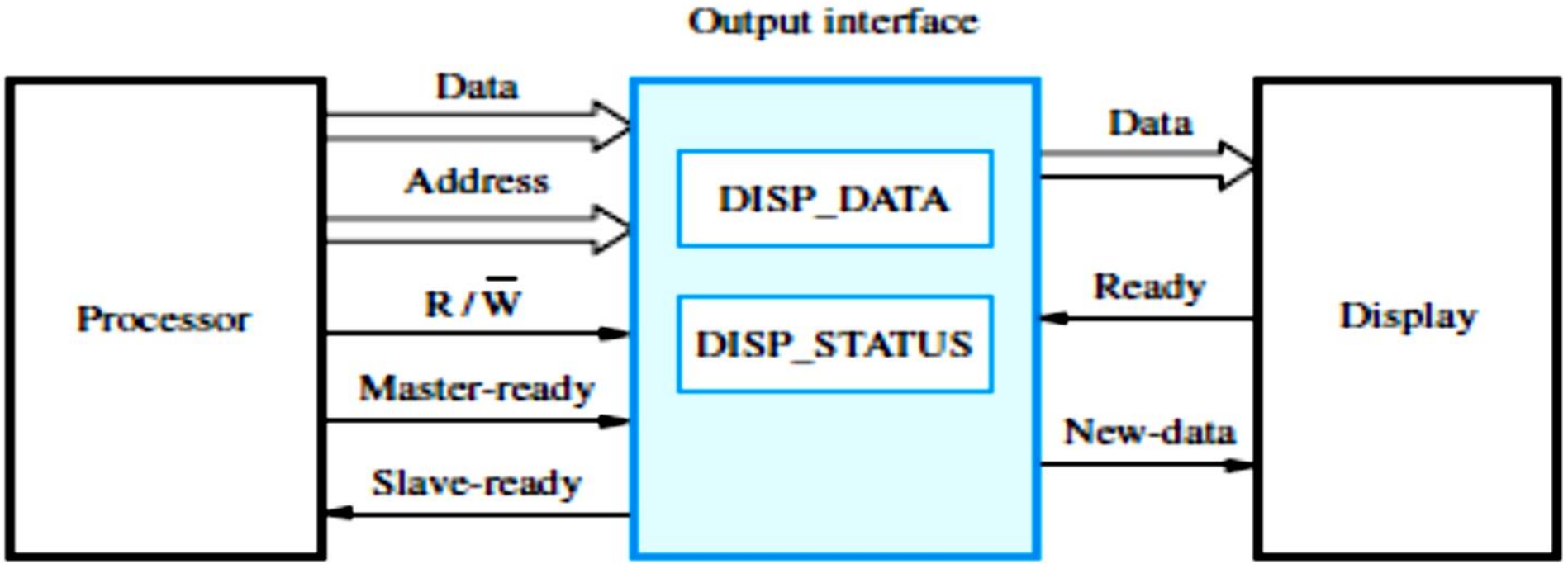
Keyboard to processor connection

- The effect of bouncing can be eliminated using a simple debouncing circuit, which could be part of the keyboard hardware or may be incorporated in the encoder circuit
- Alternatively, switch bouncing can be dealt with in software
- The software detects that a key has been pressed when it observes that the keyboard status flag, KIN, has been set to 1.
- The I/O routine can then introduce sufficient delay before reading the contents of the input buffer, KBD_DATA, to ensure that bouncing has subsided
- When debouncing is implemented in hardware, the I/O routine can read the input character as soon as it detects that KIN is equal to 1.

Keyboard to processor connection

- The output of the encoder consists of one byte of data representing the encoded character and one control signal called Valid.
- When a key is pressed, the Valid signal changes from 0 to 1, causing the ASCII code of the corresponding character to be loaded into the KBD_DATA register and the status flag KIN to be set to 1
- The status flag is cleared to 0 when the processor reads the contents of the KBD_DATA register
- The interface circuit is shown connected to an asynchronous bus on which transfers are controlled by the handshake signals Master-ready and Slave-ready
- The bus has one other control line, $R/\overline{W}$, which indicates a Read operation when equal to 1.

Output Interface



Output Interface

- We have assumed that the display uses two handshake signals, New-data and Ready, in a manner similar to the handshake between the bus signals Master-ready and Slave-ready
- When the display is ready to accept a character, it asserts its Ready signal, which causes the DOUT flag in the DISP_STATUS register to be set to 1
- When the I/O routine checks DOUT and finds it equal to 1, it sends a character to DISP_DATA. This clears the DOUT flag to 0 and sets the New-data signal to 1
- In response, the display returns Ready to 0 and accepts and displays the character in DISP_DATA
- When it is ready to receive another character, it asserts Ready again, and the cycle repeats

Interconnection Standards

- Standard interfaces have been developed to enable I/O devices to use interfaces that are independent of any particular processor.
  - For example, a memory key that has a USB connector can be used with any computer that has a USB port.
- Most standards are developed by a collaborative effort among a number of companies.
  - In many cases, the IEEE (Institute of Electrical and Electronics Engineers) develops these standards further and publishes them as IEEE Standards.

Universal Serial Bus (USB)

- The Universal Serial Bus (USB) is the most widely used interconnection standard.
- A large variety of devices are available with a USB connector, including mice, memory keys, disk drives, printers, cameras, and many more.
- The commercial success of the USB is due to its simplicity and low cost.
- The original USB specification supports two speeds of operation, called low-speed (1.5 Megabits/s) and full-speed (12 Megabits/s).
- Later, USB 2, called High-Speed USB, was introduced. It enables data transfers at speeds up to 480 Megabits/s.
- As I/O devices continued to evolve with even higher speed requirements, USB 3 (called Superspeed) was developed.
- It supports data transfer rates up to 5 Gigabits/s.


USB Type A
USB Type B
USB 3.0
USB Mini
USB Micro
USB Type C
USB Micro B
TechTerms.com

USB

- The USB has been designed to meet several key objectives:
  - Provide a simple, low-cost, and easy to use interconnection system
  - Accommodate a wide range of I/O devices and bit rates, including Internet connections, and audio and video applications
  - Enhance user convenience through a “plug-and-play” mode of operation

PCI Bus

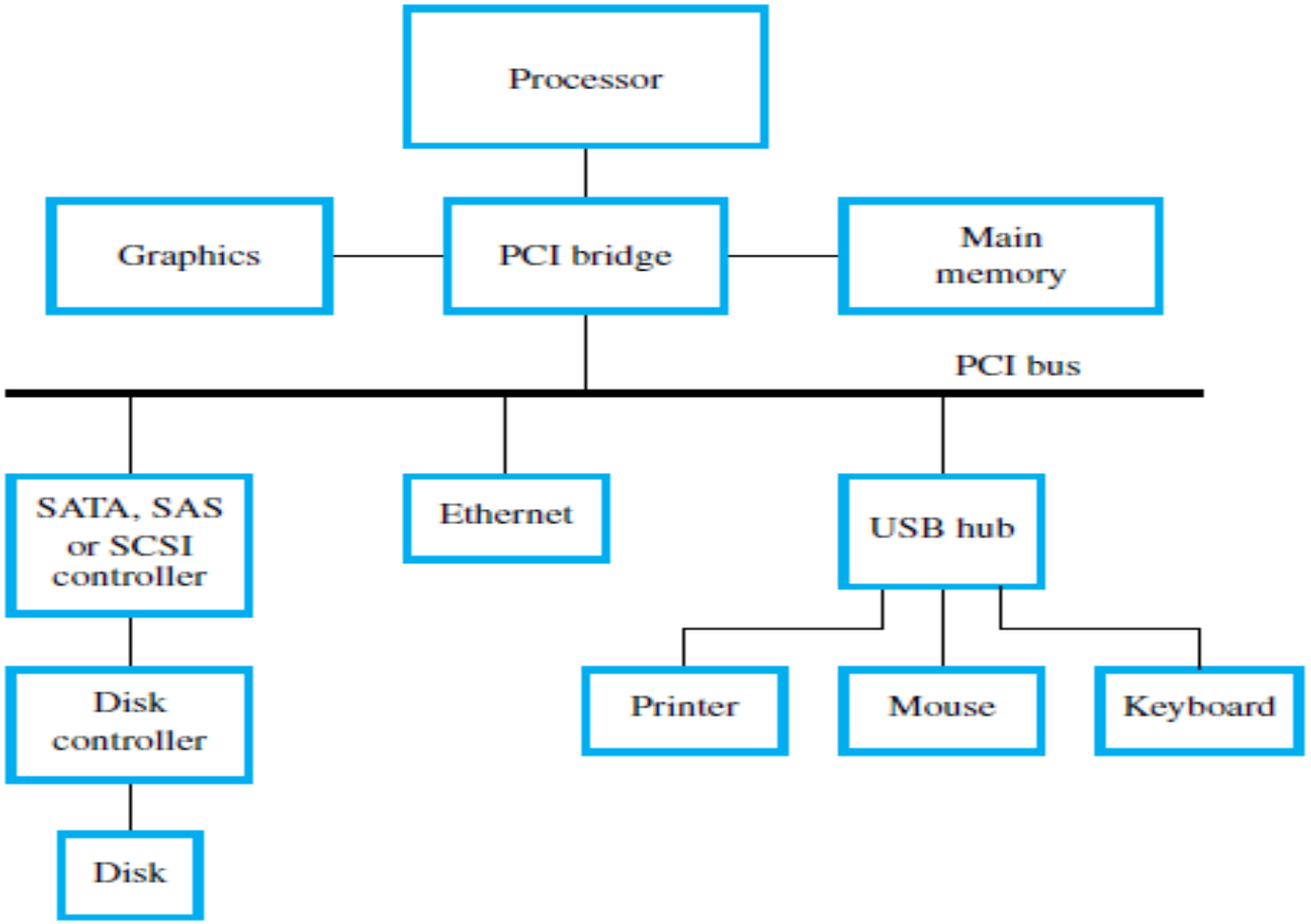
The PCI (Peripheral Component Interconnect) bus was developed as a low-cost, processor-independent bus.

It is housed on the motherboard of a computer and used to connect I/O interfaces for a wide variety of devices.

A device connected to the PCI bus appears to the processor as if it is connected directly to the processor bus.

Its interface registers are assigned addresses in the address space of the processor.

Use of a PCI bus in a computer system



SCSI Bus

- The acronym SCSI stands for Small Computer System Interface.
- It refers to a standard bus defined by the American National Standards Institute (ANSI).
- The SCSI bus may be used to connect a variety of devices to a computer. It is particularly well-suited for use with disk drives.
- It is often found in installations such as institutional databases or email systems where many disks drives are used.

SATA

- In the early days of the personal computer, the bus of a popular IBM computer called AT, which was based on Intel's 8080 microprocessor bus, became an industry standard.
- It was named ISA, for Industry Standard Architecture. An enhanced version, including a definition of the basic software needed to support disk drives, was later named ATA, for AT Attachment bus.
- A serial version of the same architecture became known as SATA, which is now widely used as an interface for disks.
- Like all standards, several versions of SATA have been developed with added features and higher speeds.

SAS

- This is a serial implementation of the SCSI bus, hence its name: Serially Attached SCSI.
- It is primarily intended for connecting magnetic disks and CD and DVD drives.
- It uses serial, point-to-point links that are similar to SATA.
- A SAS link can transfer data in both directions simultaneously

PCI Express

- The demands placed on I/O interconnections are ever increasing.
- Internet connections, sophisticated graphics devices, streaming video and high-definition television are examples of applications that involve data transfers at very high speed.
- The PCI Express interconnection standard (often called PCIe) has been developed to meet these needs and to anticipate further increases in data transfer rates, which are inevitable as new applications are introduced.
- PCI Express uses serial, point-to-point links interconnected via switches to form a tree structure