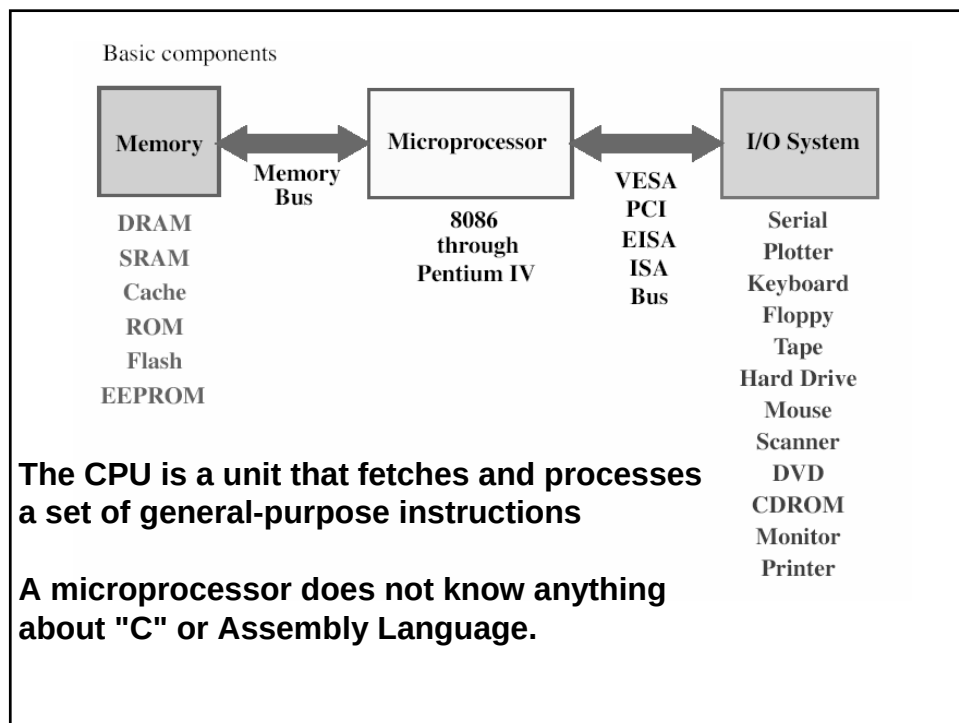
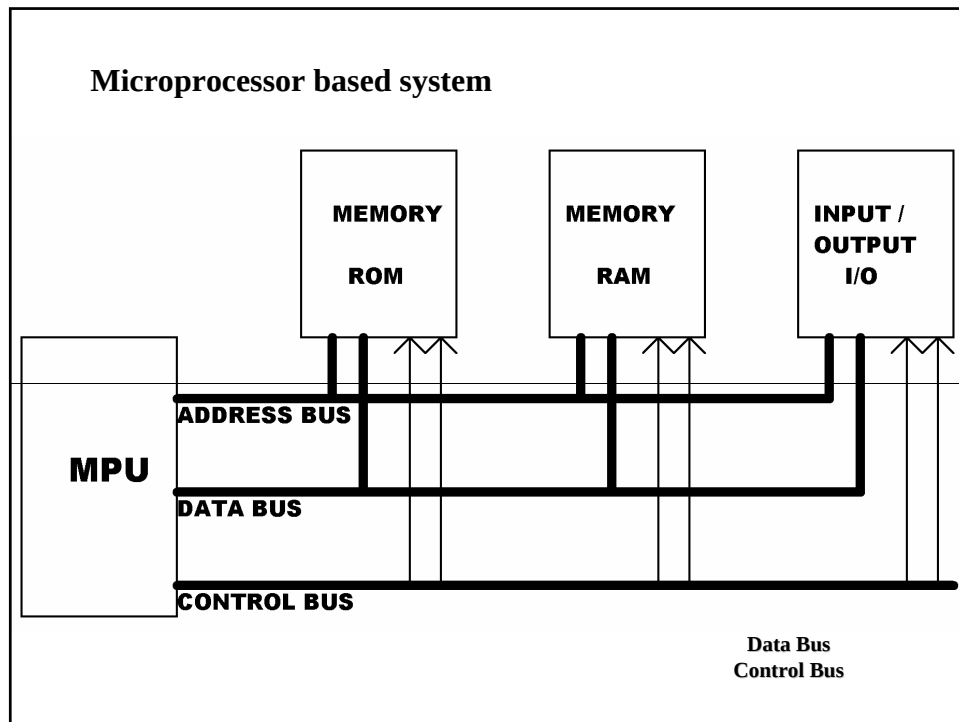
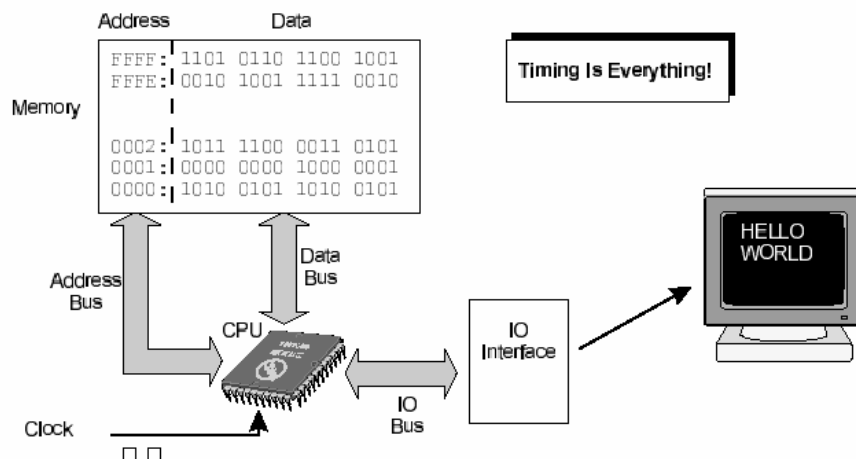


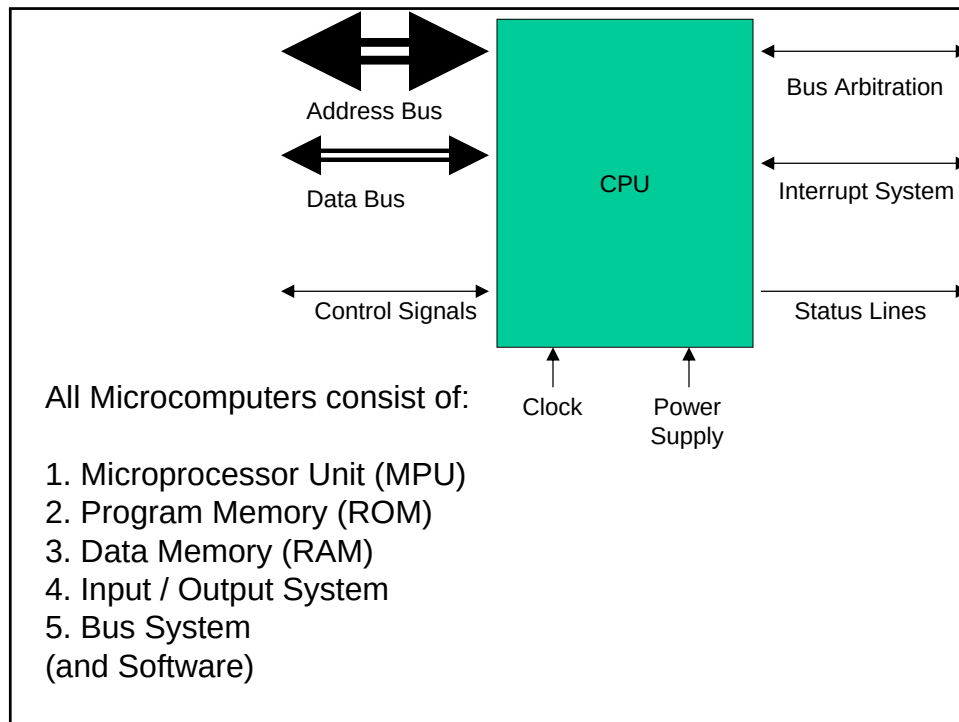


Examples of Embedded Systems

- Consumer electronics
 - Cell phones, pagers, digital cameras, camcorders, PDAs, DVD players, calculators
- Home Appliances
 - Microwave ovens, answering machines, thermostats, home security systems
- Office Automation
 - Fax machines, copiers, printers, scanners
- Business Equipment
 - Cash registers, curbside check-in, alarm systems, card readers, ATMs, product scanners
- Automobiles
 - Transmission control, cruise control, fuel injection, antilock brakes, active suspension
- Computing system peripherals and networking systems
 - Networking equipment, routers, printers, network cards, monitors and displays
- Any many more
- E.g. In 1999, a typical American household had one PC and about 40-50 embedded computers, this has risen to about 300-400 by 2004. An average car can have more than 50 embedded computers.

Microprocessor based system





The microprocessor is a single chip which is capable of processing data and controlling all of the components which make up the microcomputer system. It contains all of the circuits needed to create the 'brain of the microcomputer'.

1. Temporary storage locations in the form of a number of registers which can hold binary values (information), representing program instruction or data.

2. The Arithmetic Logic unit (ALU). This part of the MPU performs both arithmetic and logical operations

3. Timing and Control Circuits : that keep all of the other parts of system (memory & I/O) working together in the right time sequence.

A Bus is a common communications pathway used to carry information between the various elements of a computer system. The term BUS refers to a group of wires or conduction tracks on a printed circuit board (PCB) through which binary information is transferred from one part of the microcomputer to another.

There are three main bus groups

1. ADDRESS BUS

2. DATA BUS

3. CONTROL BUS

Data Bus:

The Data Bus carries the data which is transferred throughout the system. (bi-directional)

Address Bus:

An address is a binary number that identifies a specific memory storage location or I/O port involved in a data transfer

The Address Bus is used to transmit the address of the location to the memory or the I/O port.

The Address Bus is unidirectional (one way): addresses are always issued by the MPU.

The Control Bus: is another group of signals whose functions are to provide synchronization (timing control) between the MPU and the other system components. Control signals are unidirectional, and are mainly outputs from the MPU. Example Control signals

RD read signal asserted to read data into MPU

WR write signal asserted to write data from MPU

Bus Sizes: Buses are specified in width, number of bits.

The width of the external **Data bus** generally gives a very good indication of the internal MPU architecture, particularly the registers size and the ALU.

The **Data Bus** width limits the bandwidth of the data that can be transferred per bus operation.

Address Bus :

The MPU Address bus size specifies how much memory and I/O locations may accessed.

Number of locations = 2^N where N is number of Address lines

The Address Bus may be specified using the following 16 Bits Wide, (A15 to A0), 16 Address Lines

In this instance $2^N = 2^{16} = 65536$ locations

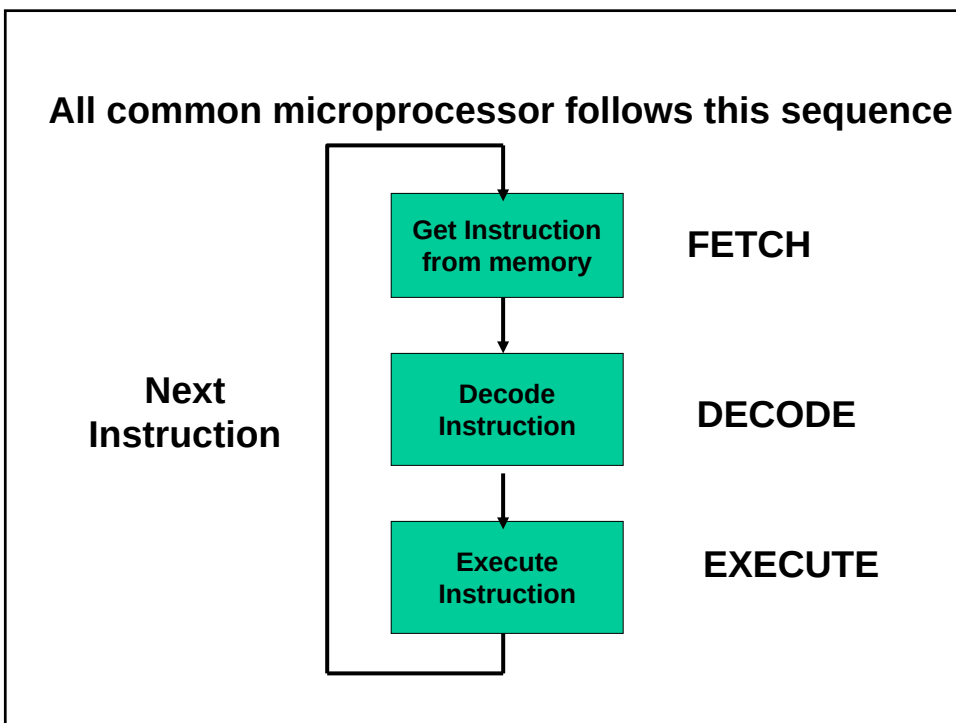
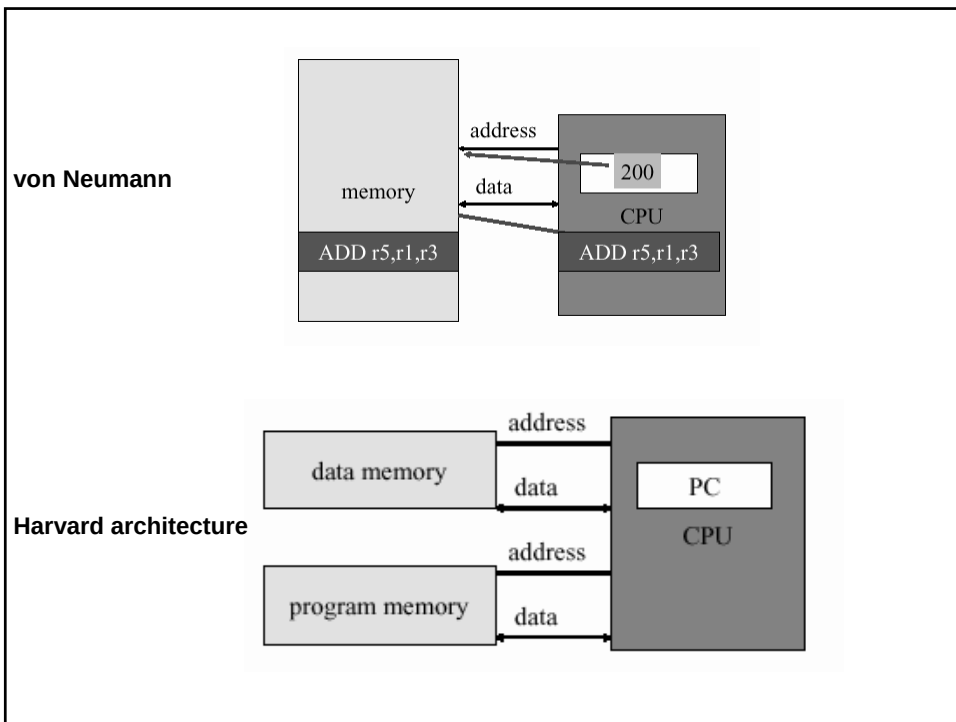
Microprocessors are classified by

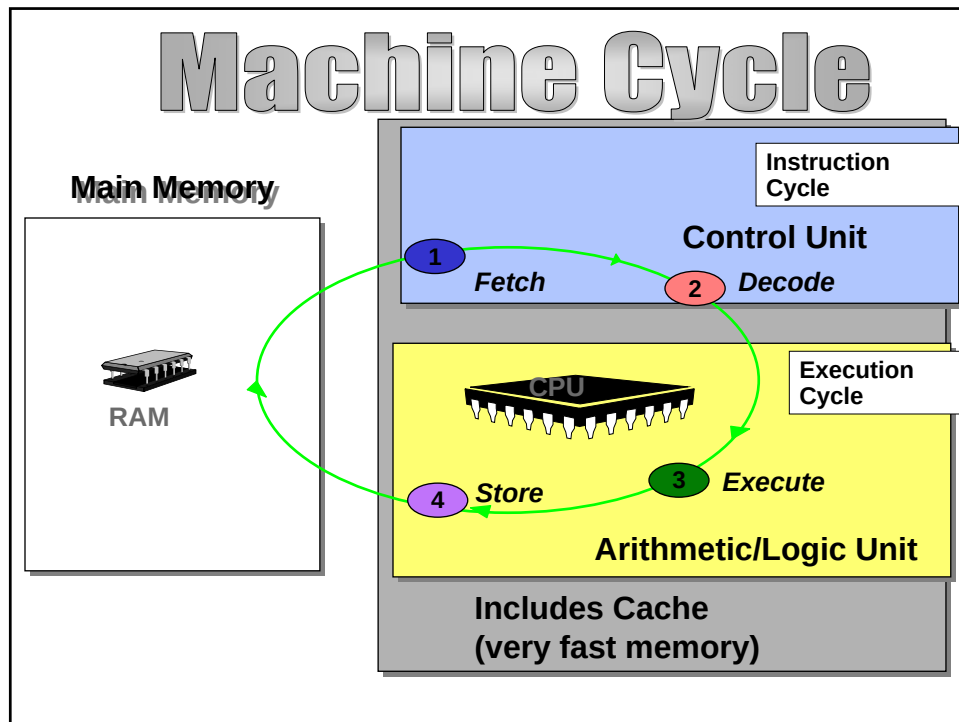
- The semiconductor technology of their design (TTL, transistor-transistor logic; CMOS, complementary-metal-oxide semiconductor; or ECL, emitter-coupled logic),

- The width of the data format (4-bit, 8-bit, 16-bit, 32-bit, or 64-bit) they process; and by their

- instruction set (CISC, complex-instruction-set computer, or RISC, reduced-instruction-set computer; see RISC processor).

von Neumann or Harvard (Memory usage)





A Simplified view of MPU program execution

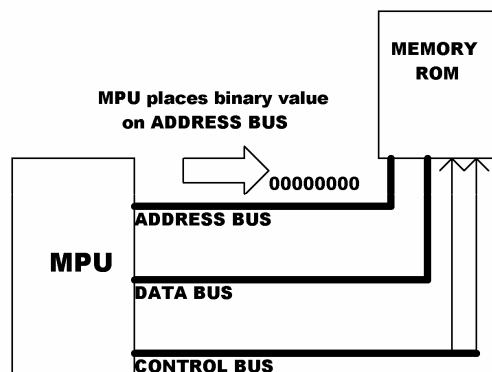
A MPU program is stored in memory as a sequence of binary values.

These binary values represent instructions. Each instruction has an unique binary value.

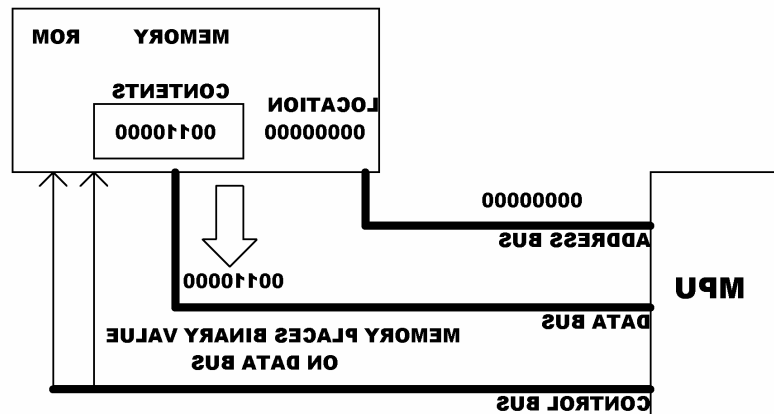
The MPU operation is controlled by reading and executing each instruction one at a time.

1. The MPU places an address on the ADDRESS BUS.

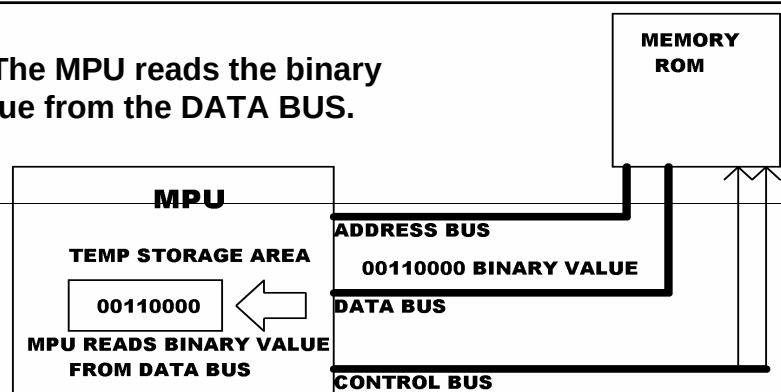
The address is a binary value which represents a unique memory location.



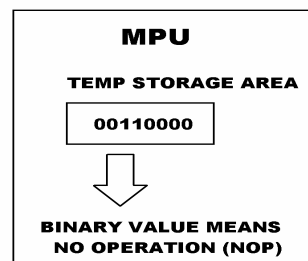
2. The Memory places the contents of the addressed location onto the DATA BUS. The contents is a binary value.



3. The MPU reads the binary value from the DATA BUS.



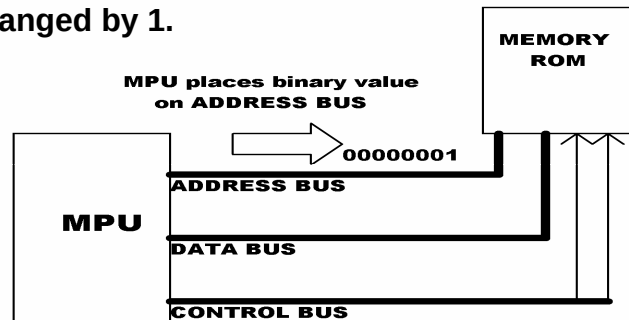
4. The MPU interprets what operation the BINARY Value represents and completes (executes) this operation.



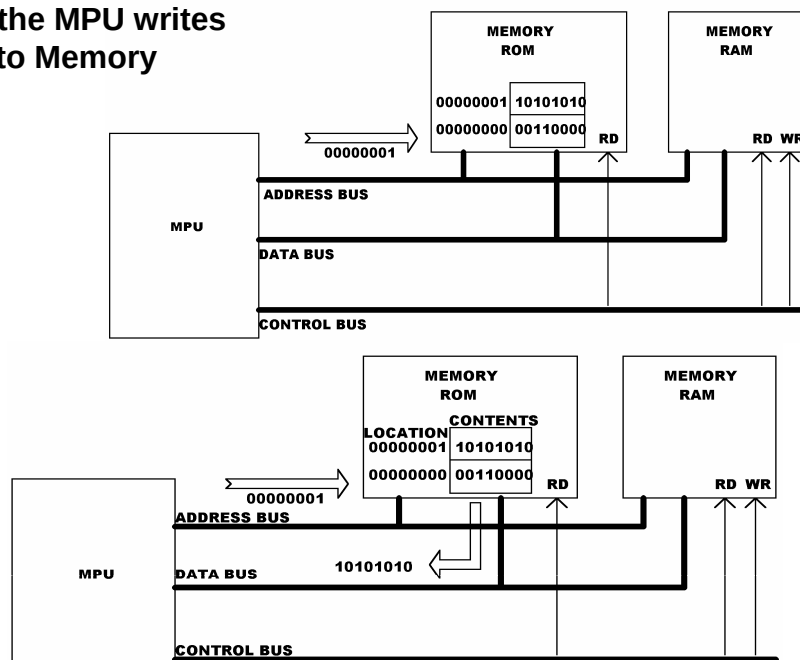
5. The MPU has completed its first instruction of the program stored as binary values in the memory.

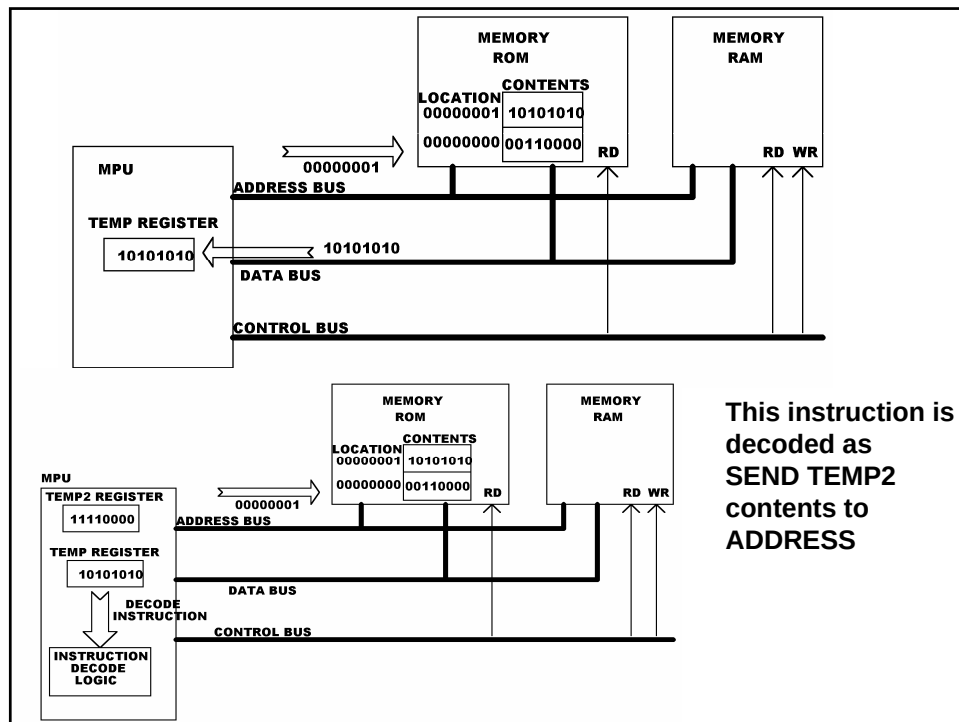
The MPU needs to be told what to do next. The next instruction is in the next memory location. Stages 1. through 4. are repeated,

Note. At Stage 1. The MPU places a new binary value onto the ADDRESS BUS. This is the memory location of the next instruction. The binary value on the ADDRESS BUS has only changed by 1.



How the MPU writes data to Memory

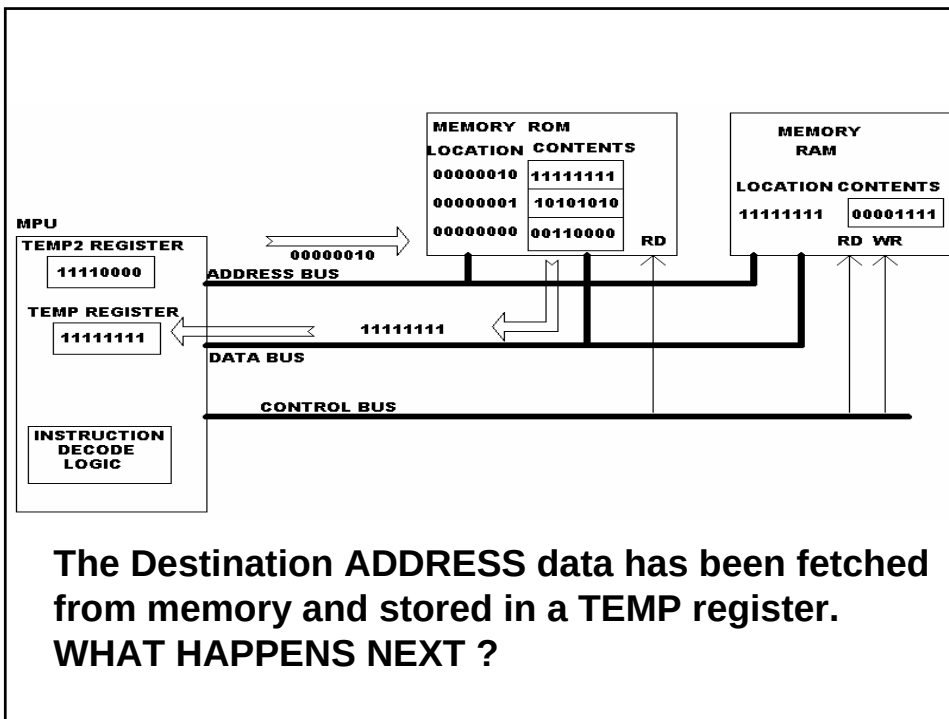




The MPU instruction decode logic now need to execute this instruction
SEND TEMP2 to an ADDRESS

TO WHAT ADDRESS ...

This particular MPU instruction decodes as saying that the **DESTINATION ADDRESS** will be in the next memory location. The MPU then fetches the binary value from the next location, as this will be the **ADDRESS** where TEMP2 contents will be SENT.

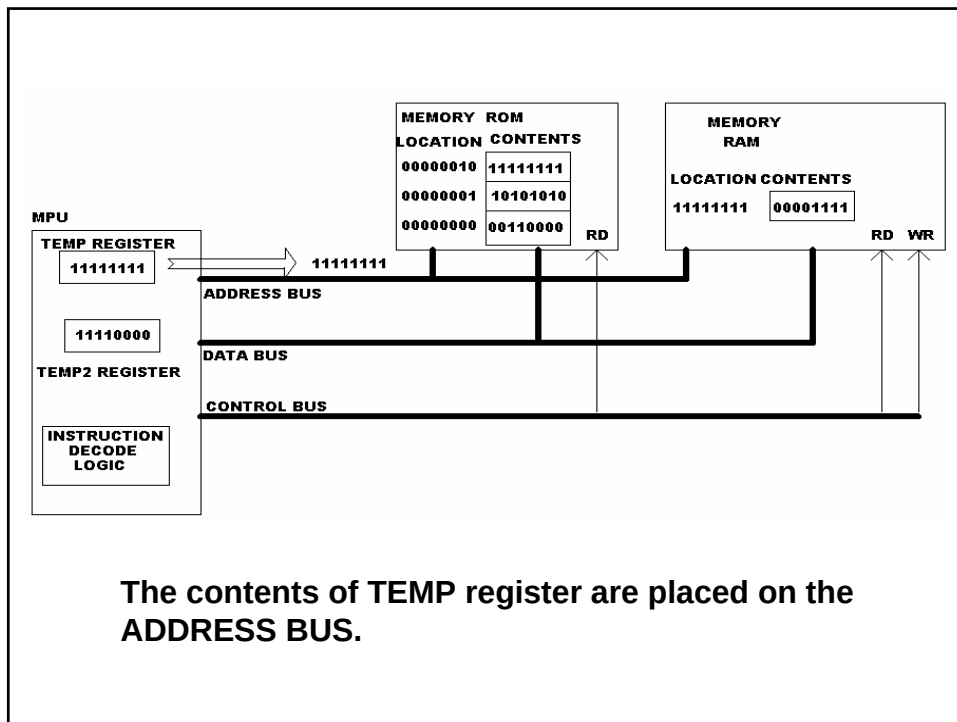


The TEMP register holds the DESTINATION ADDRESS where we wish to SEND our DATA (Binary Value) which is stored in TEMP2 register.

We do not try to decode the contents of TEMP because it is not an instruction, it is data associated with a specific operation, it is known as an OPERAND

WHAT HAPPENS NEXT ..

Remember we want TEMP2 contents to be sent to the MEMORY ADDRESS which is currently held in TEMP.

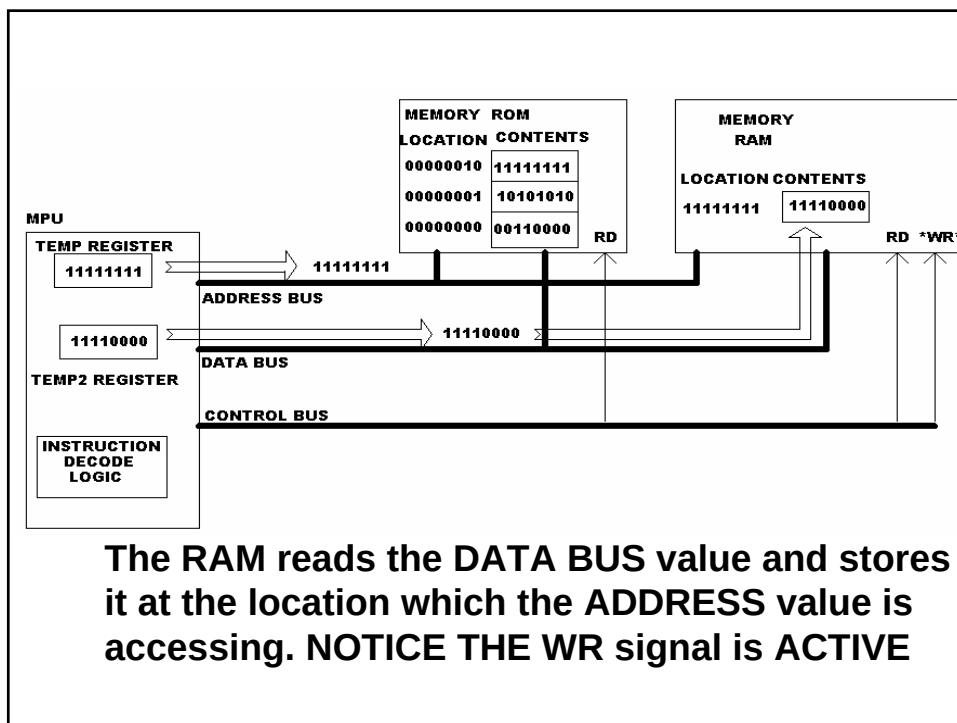
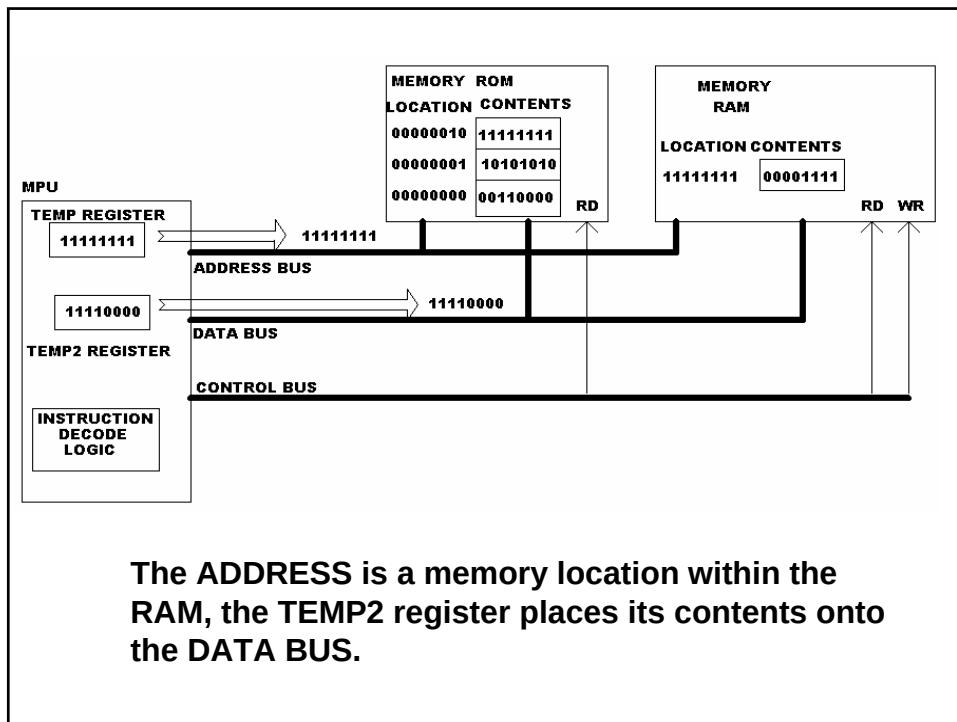


The MEMORY WRITE part of our instruction has started. In this phase, since we are sending DATA to MEMORY or WRITING TO MEMORY we must be accessing a MEMORY store which allows us to do so.

ROM is read only memory, where our program is stored, we cannot write to it.

The RAM random access memory ,(read/write memory) is where our DATA will be written.

Notice the control signal lines on each of the memory types.



The instruction has now completed executing
The result of this operation is that the contents
of TEMP2 register have been WRITTEN to a RAM
Memory location 11111111.

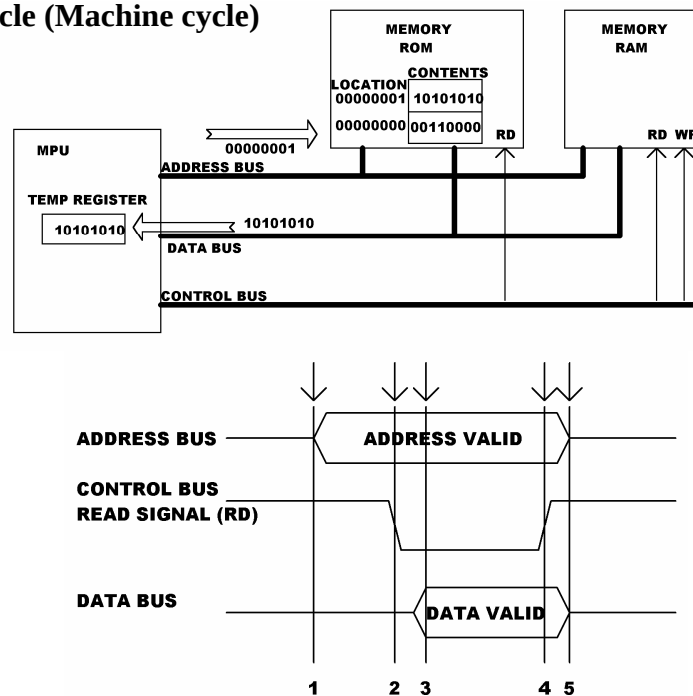
Prior to the DATA WRITE the contents of RAM location was 00001111.
After the MEMORY WRITE RAM Memory location now contains
11110000 the previous DATA value has been overwritten.

A Quick Review :

The MPU fetches instructions one at a time from memory
The instruction is interpreted by MPU and then this
operation is executed. This called the **FETCH DECODE
EXECUTE CYCLE**. The program, sequence of instructions
are held in consecutive memory locations.

We have seen that not every binary value read from
memory is an instruction. Some of the values can be
DATA associated with an instruction.

Read cycle (Machine cycle)



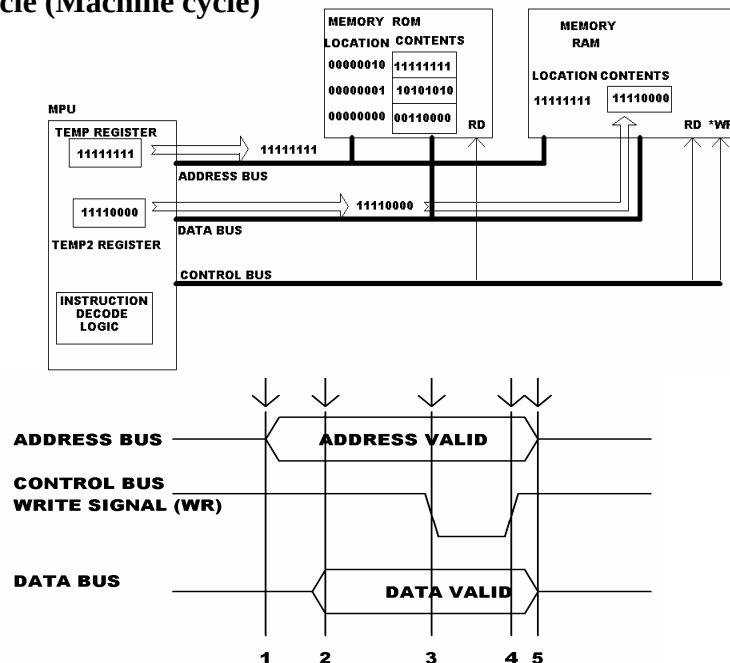
Simple Read Cycle

1. Address onto Address Bus
2. READ (RD) Control Signal Asserted by MPU
3. Data Placed from Memory onto the Data Bus
4. RD signal De-Asserted, Data into MPU temp register.
5. Read Cycle Ends

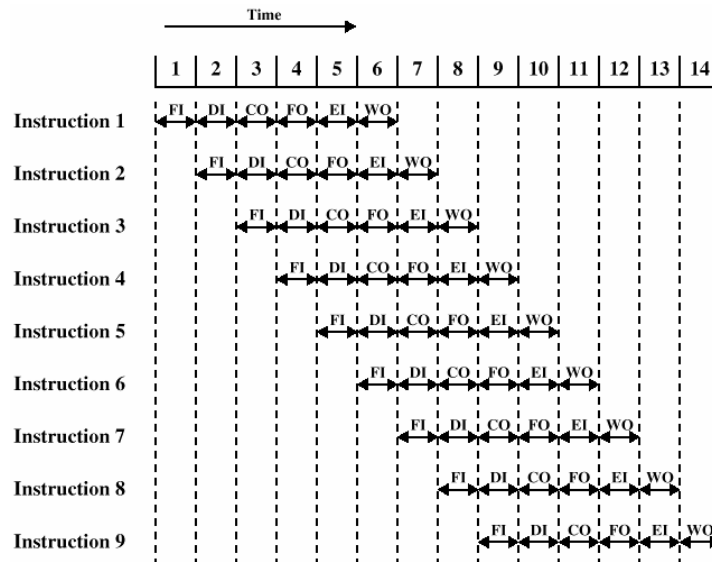
Write Cycle

1. Address placed on ADDRESS BUS
2. Data placed onto DATA BUS
3. Write (WR) Control Signal Asserted by MPU
4. Write (WR) Control Signal De-Asserted and Data Written into Memory
5. Write Cycles completes.

Write cycle (Machine cycle)



Pipeline Timing Diagram



Pipeline Stages

- Different architectures use different number of stages
 - smaller pipelines have less complexity
 - larger pipelines are generally faster
 - here, we decompose the fetch-execute cycle into 6 stages:
 - FI – fetch instruction
 - DI – decode instruction
 - CO – calculate operand addresses
 - FO – fetch operands (from registers)
 - EI – execute instruction (includes computing branch results and all ALU operations)
 - WO – write or store result in registers (or memory)
 - the design of the pipeline stages has a major impact on the architecture's performance

Levels of Programming Languages

1) Machine Language

- Consists of individual instructions that will be executed by the CPU one at a time

2) Assembly Language (Low Level Language)

- Designed for a specific family of processors (different processor groups/family has different Assembly Language)
- Consists of symbolic instructions directly related to machine language instructions one-for-one and are assembled into machine language.

3) High Level Languages

- e.g. : C, C++ and Vbasic
- Designed to eliminate the technicalities of a particular computer.
- Statements compiled in a high level language typically generate many low-level instructions.

Advantages of Assembly Language

1. Shows how program interfaces with the processor, operating system, and BIOS.
2. Shows how data is represented and stored in memory and on external devices.
3. Clarifies how processor accesses and executes instructions and how instructions access and process data.
4. Clarifies how a program accesses external devices.

Reasons for using Assembly Language

1. A program written in Assembly Language requires considerably less memory and execution time than one written in a high-level language.
2. Assembly Language gives a programmer the ability to perform highly technical tasks that would be difficult, if not impossible in a high-level language.
3. Although most software specialists develop new applications in high-level languages, which are easier to write and maintain, a common practice is to recode in assembly language those sections that are time-critical.
4. Resident programs (that reside in memory while other program execute) and interrupt service routines (that handle input and output) are almost always develop in Assembly Language.

➤ High-Level Language (e.g. C++, Java, Lisp, Pascal, APL, ...)

`A := B + C`

Assignment Statement

➤ Low-Level Language -> Assembly Language

➤ (e.g. Intel Pentium Assembly Language, Motorola 680x0 Assembly Language, Java Bytecode)

`MOV AX, [B]
ADD AX, [C]
MOV [A], AX`

Assembly Language
Instructions

➤ (Binary) Machine Code

`1A01
3A02
2A00`

Machine Code Instructions (hex)

High Level Language Level Language

Int: A,B,C

A := B + C

Assembly Language for Language for A:=B+C

```
MOV  AX,[B]
ADD  AX,[C]
MOV  [A],AX
```

Machine Code for Code for A:=B+C

MOV	AX,[B]	0001	0011	1111	1101
ADD	AX,[C]	0011	0011	1111	1110
MOV	[A],AX	0010	0011	1111	1100

Instruction Execution Cycle

- The sequence of steps the processor goes through to do the complete instruction and to get ready for the next, until the program is finished or stopped
- Specific steps (simplified)
 - IF: instruction fetched
 - ID: instruction decoded, control signals asserted
 - RT/EX: register transfer and/or execution
 - PC: increment program counter
- These steps are repeated for each instruction execution cycle until the program is done, or until something is done to make the program stop

- Opcodes are represented by abbreviations, called mnemonics, that indicate the operation. Common examples:
- ADD Add
- SUB Subtract
- DIV Divide
- LOAD Load data from memory
- STOR Store data to memory

$$Y = (A - B) / (C + D \times E)$$

Instruction	Comment
• SUB Y, A, B	• Y ← A - B
• MPY T, D, E	• T ← D × E
• ADD T, T, C	• T ← T + C
• DIV Y, Y, T	• Y ← Y / T

*Three-address
instructions*

$$Y = (A-B) / (C+D \times E)$$

- MOVE Y, A
- SUB Y, B
- MOVE T, D
- MPY T, E
- ADD T, C
- DIV Y, T

Two-address instructions

$$Y = (A-B) / (C+D \times E)$$

iNSTRUCTIONS

- LOAD D
- MPY E
- ADD C
- STOR Y
- LOAD A
- SUB B
- DIV Y
- STOR Y

Comment

- $AC \longleftarrow D$
- $AC \longleftarrow AC \times E$
- $AC \longleftarrow AC + C$
- $Y \longleftarrow AC$
- $AC \longleftarrow A$
- $AC \longleftarrow AC - B$
- $AC \longleftarrow AC / Y$
- $Y \longleftarrow AC$

One-address Instructions

Examples

- MOV BL,AL
- Opcode for MOV = 100010
- We'll encode AL so
 - D = 0 (AL source operand)
- W bit = 0 (8-bits)
- MOD = 11 (register mode)
- REG = 000 (code for AL)
- R/M = 011

opcode	D	W	MOD	REG	R/M
--------	---	---	-----	-----	-----

MOV AL,BL = 10001000 11000011 = 88 C3h

ADD AX,[SI] = 00000011 00000100 = 03 04 h

ADD [BX],[DI] + 1234h, AX = 00000001 10000001 ____ h
= 01 81 34 12 h

The basic function as stated is to perform logic and maths operations.

The Accumulator and the Flags registers play an important role in the operation of the ALU.

The Accumulator is used in all ALU operations.

The Flags or Status Register is a group of bits representing the result of the ALU operation. The Flags register contains bits such as (OV) representing an overflow flag. If the result of an addition is greater than 255 then the OV is set to indicate this condition. More Details to come.

The type of functions performed by the ALUs of MPU include

Addition	Shift Left	Subtraction	Shift Right
Logical AND	Decrement	Logical OR	Increment
			Logical XOR

What is Microprocessor ?

It is a program controlled semiconductor device (IC), which fetches, decode and executes instructions.

What is the function of microprocessor in a system?

The microprocessor is the master in the system, which controls all the activity of the system. It issues address and control signals and fetches the instruction and data from memory. Then it executes the instruction to take appropriate action.

What are the basic units of a microprocessor ?

The basic units or blocks of a microprocessor are ALU, an array of registers and control unit.

what is Software and Hardware?

The *Software* is a set of instructions or commands needed for performing a specific task by a programmable device or a computing machine.

Without software the Hardware is an idle machine.

What is assembly language?

The language in which the mnemonics (short -hand form of instructions) are used to write a program is called assembly language.

The manufacturers of microprocessor give the mnemonics.

What are machine language and assembly language programs?

The software developed using 1's and 0's are called machine language, programs. The software developed using mnemonics are called assembly language programs.

What is the drawback in machine language and assembly language, programs?

The machine language and assembly language programs are machine dependent. The programs developed using these languages for a particular machine cannot be directly run on another machine .

Define opcode and operand.

Opcode (Operation code) is the part of an instruction / directive that identifies a specific operation. Operand is a part of an instruction / directive that represents a value on which the instruction acts.

What is pipelined architecture?

In pipelined architecture the processor will have number of functional units and the execution time of functional units are overlapped. Each functional unit works independently most of the time.

What is fetch and execute cycle?

In general, the instruction cycle of an instruction can be divided into fetch and execute cycles. The fetch cycle is executed to fetch the opcode from memory. The execute cycle is executed to decode the instruction and to perform the work instructed by the instruction.

Define machine cycle.

Machine cycle is defined as the time required to complete one operation of accessing memory, I/O, or acknowledging an external request. This cycle may consist of three to six T-states.

What is processor cycle (Machine cycle)?

The processor cycle or machine cycle is the basic operation performed by the processor. To execute an instruction, the processor will run one or more machine cycles in a particular order.

What is Instruction cycle?

The sequence of operations that a processor has to carry out while executing the instruction is called Instruction cycle. Each instruction cycle of a processor indium consists of a number of machine cycles.

What is opcode fetch cycle?

The opcode fetch cycle is a machine cycle executed to fetch the opcode of an instruction stored in memory. Every instruction starts with opcode fetch machine cycle.

Why interfacing is needed for I/O devices?

Generally I/O devices are slow devices. Therefore the speed of I/O devices does not match with the speed of microprocessor. And so an interface is provided between system bus and I/O devices.

What is the need for Port?

The I/O devices are generally slow devices and their timing characteristics do not match with processor timings. Hence the I/O devices are connected to system bus through the ports.

.What is a port?

The port is a buffered I/O, which is used to hold the data transmitted from the microprocessor to I/O device or vice-versa.