Unit 1
Introduction

Process: a series of actions or steps taken to achieve an end result
Processor: a machine that completes process
IC: multifunction circuit are combined in a single chip

Microprocessor:

- → It is a multipurpose, programmable, clock-driven, register-based electronic device that reads binary instruction from a storage device (memory), accepts binary as input and processes data according to those instructions and provides result as output.
- → Each MP communicates and operates in the binary number 0 1nd 1, called bits.
- → Each MP has fixed sets of instructions in the form of binary pattern called a machine language.

Applications of MP

The applications of microprocessors are not bound. They can be used virtually anywhere and in any field. However, the applications are sorted as follows:

Test Instruments

Microprocessors are widely used in devices such as signal generators, oscilloscopes, counters, digital multi-meters, x-ray analyzers, blood group analyzers, baby incubator, frequency synthesizers, data acquisition systems, spectrum analyzers etc. For example fluke 6010A synthesized signal generator uses 4004 microprocessor.

Communications

Communication today requires tens of thousands of circuits to be managed. Data should be received, checked for errors and further analysis should also be performed. The speed at which the microprocessor can take decisions and compute errors is truly substantial.

Computer

The microprocessor is a central processing unit (CPU) of the microcomputers. It can perform arithmetic and logic functions as well as control function. The control unit of microprocessor sends signals to input, output units, memory, ALU and arrange the sequence of their controlling operation.

Industries

The microprocessor is widely used in data monitoring systems, smart cameras for quality control, automatic weighing, batching systems, assembly machine control, torque certification systems, machine tool controller etc.

Evolution of MP (Intel Series)

Intel 4004

- — Year of introduction 1970
- — 4-bit MP
- — 4 KB memory
- — Speed: 108 KHz
- — World's first MP

Intel 8008

- — Year of introduction 1972
- — 8-bit version of 4004
- — 16 KB memory
- — Slow

Intel 8080

- — Year of introduction 1974
- — 8-bit MP

- Speed: 2 MHz
- First general purpose MP used as CPU of computer
- 64 KB memory

Intel 8085

- Year of introduction 1975
- 8-bit MP
- 64 KB memory
- 8-bit data bus

Intel 8086

- Year of introduction 1976
- 16-bit MP
- 1 MB memory
- 6-byte instruction queue (cache)
- 16-bit data bus
- Speed: 4.77 MHz

Intel 8088

- Year of introduction 1979
- First MP used in the original IBM PC
- 8-bit data bus
- 1 MB memory
- Speed: 4.77 MHz
- 4-byte instruction queue (cache)

Intel 80286

- Year of introduction 1982/1983
- 16-bit MP with memory and protection
- 16 MB memory
- Speed: 8 MHz
- Multitasking feature implemented

Intel 80386

- Year of introduction 1986
- First practical 32-bit MP
- 4 GB memory

Intel 80486

- Year of introduction 1989
- 32-bit high performance MP
- 4 GB memory

Pentium

- Year of introduction 1993
- 32-bit MP
- 64-bit external data bus
- 4 GB memory

— 16 KB cache

Microcomputer

→ It is a computer with a CPU as a MP designed for personal use.
→ It contains MP, memory and I/O unit.
→ It is smaller than mainframe and minicomputer.

Microcontroller

→ It is a small single chip computer or single IC containing MP, memory and programmable I/O peripherals.

MP	Input
Memory	Output

Von Neumann Architecture

→ Program can be saved like data in the memory unit and can be accessed when needed. This approach is called 'Stored Program Concept' and was first adopted by John von Neumann.

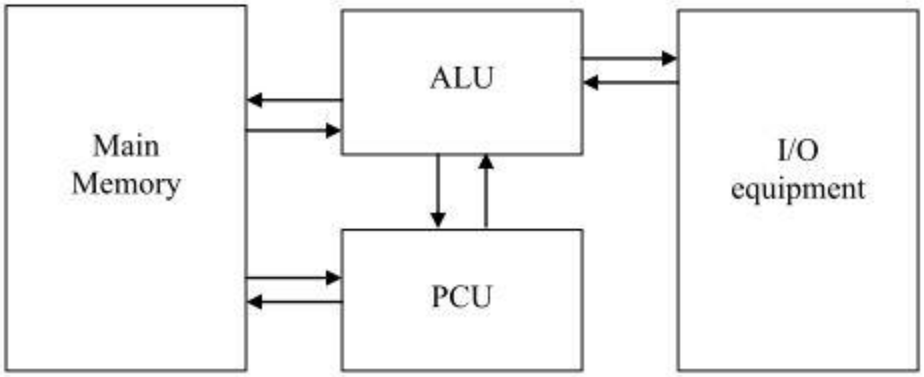
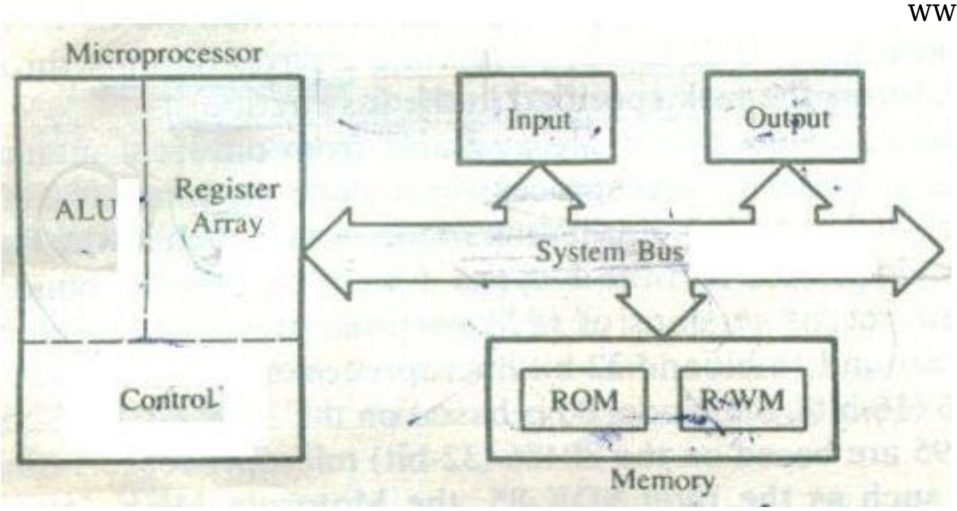


Figure: Von Neumann architecture

→ In this architecture, data and instructions are stored in a single set of main memory.
→ Instruction fetch and data operation cannot occur at the same time because they share a common bus.
→ The program control unit (PCU) reads program instruction, decodes instruction for ALU and determines the sequence of instruction to be executed.
→ The ALU performs arithmetic and logical operations.
→ It is a basic architecture of today's computer.
→ The another architecture like this is Harvard architecture in which instruction and data have separate memory space; and data & instruction can be accessed at the same time. This is newer approach to von Neumann architecture.

Basic Organization of Microcomputer



1. **MP**
   i. **ALU**
      — This unit executes all arithmetic and logical operations as specified by instruction set; and produces output.
      — The results of addition, subtraction, and logical operations (AND, OR, XOR) are stored in the registers or in memory unit or sent to output unit.

   ii. **Register unit**
      — Consists of various registers.
      — Used for temporary storage of data during execution of data.

   iii. **CU**
      — Controls the operations of different instructions.
      — Provides necessary timing and control signals to all the operations in the MP and peripherals including memory.

2. **Memory**

→ Stores binary information such as instruction and data, and provide these information to MP when required.
→ To execute programs, the MP reads data and instructions from memory and performs the computing operations.

3. **System bus**

→ The system bus is a communication path between MP and peripherals.
→ It is used to carry data, address and control signals.
→ It consists:
   — Data bus: carries data
   — Address bus: carries address
   — Control bus: carries control signals

4. **I/O bus**

→ Input unit is used to input instruction or data to the MP externally.
→ Output unit is used to carry out the information from the MP unit.

Unit 1
Introduction

Process: a series of actions or steps taken to achieve an end result
Processor: a machine that completes process
IC: multifunction circuit are combined in a single chip

Microprocessor:

→ It is a multipurpose, programmable, clock-driven, register-based electronic device that reads binary instruction from a storage device (memory), accepts binary as input and processes data according to those instructions and provides result as output.

→ Each MP communicates and operates in the binary number 0 1nd 1, called bits.

→ Each MP has fixed sets of instructions in the form of binary pattern called a machine language.

Applications of MP

The applications of microprocessors are not bound. They can be used virtually anywhere and in any field. However, the applications are sorted as follows:

Test Instruments

Microprocessors are widely used in devices such as signal generators, oscilloscopes, counters, digital multi-meters, x-ray analyzers, blood group analyzers, baby incubator, frequency synthesizers, data acquisition systems, spectrum analyzers etc. For example fluke 6010A synthesized signal generator uses 4004 microprocessor.

Communications

Communication today requires tens of thousands of circuits to be managed. Data should be received, checked for errors and further analysis should also be performed. The speed at which the microprocessor can take decisions and compute errors is truly substantial.

Computer

The microprocessor is a central processing unit (CPU) of the microcomputers. It can perform arithmetic and logic functions as well as control function. The control unit of microprocessor sends signals to input, output units, memory, ALU and arrange the sequence of their controlling operation.

Industries

The microprocessor is widely used in data monitoring systems, smart cameras for quality control, automatic weighing, batching systems, assembly machine control, torque certification systems, machine tool controller etc.

Evolution of MP (Intel Series)

Intel 4004

- Year of introduction 1970
- 4-bit MP
- 4 KB memory
- Speed: 108 KHz
- World's first MP

Intel 8008

- Year of introduction 1972
- 8-bit version of 4004
- 16 KB memory
- Slow

Intel 8080

- Year of introduction 1974
- 8-bit MP

- Speed: 2 MHz
- First general purpose MP used as CPU of computer
- 64 KB memory

Intel 8085

- Year of introduction 1975
- 8-bit MP
- 64 KB memory
- 8-bit data bus

Intel 8086

- Year of introduction 1976
- 16-bit MP
- 1 MB memory
- 6-byte instruction queue (cache)
- 16-bit data bus
- Speed: 4.77 MHz

Intel 8088

- Year of introduction 1979
- First MP used in the original IBM PC
- 8-bit data bus
- 1 MB memory
- Speed: 4.77 MHz
- 4-byte instruction queue (cache)

Intel 80286

- Year of introduction 1982/1983
- 16-bit MP with memory and protection
- 16 MB memory
- Speed: 8 MHz
- Multitasking feature implemented

Intel 80386

- Year of introduction 1986
- First practical 32-bit MP
- 4 GB memory

Intel 80486

- Year of introduction 1989
- 32-bit high performance MP
- 4 GB memory

Pentium

- Year of introduction 1993
- 32-bit MP
- 64-bit external data bus
- 4 GB memory

— 16 KB cache

Microcomputer

→ It is a computer with a CPU as a MP designed for personal use.

→ It contains MP, memory and I/O unit.

→ It is smaller than mainframe and minicomputer.

Microcontroller

→ It is a small single chip computer or single IC containing MP, memory and programmable I/O peripherals.

MP	Input
Memory	Output

Von Neumann Architecture

→ Program can be saved like data in the memory unit and can be accessed when needed. This approach is called 'Stored Program Concept' and was first adopted by John von Neumann.

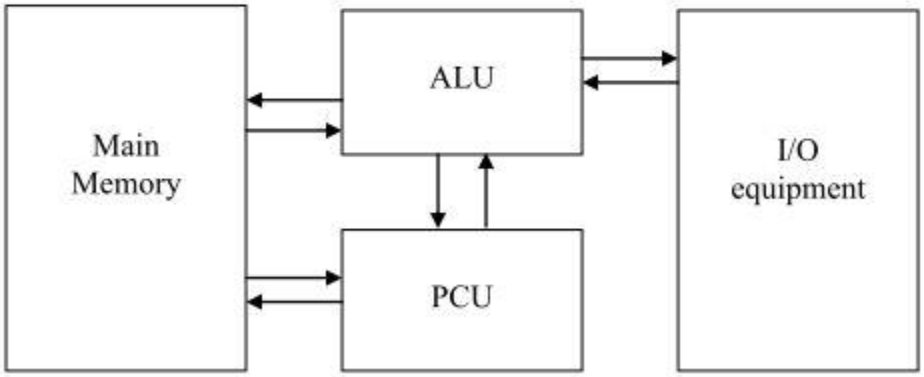


Figure: Von Neumann architecture

→ In this architecture, data and instructions are stored in a single set of main memory.

→ Instruction fetch and data operation cannot occur at the same time because they share a common bus.

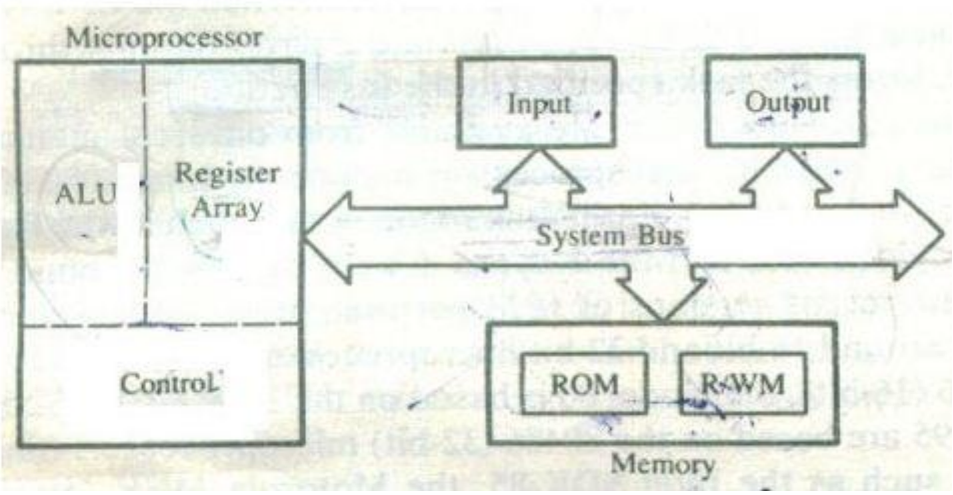
→ The program control unit (PCU) reads program instruction, decodes instruction for ALU and determines the sequence of instruction to be executed.

→ The ALU performs arithmetic and logical operations.

→ It is a basic architecture of today's computer.

→ The another architecture like this is Harvard architecture in which instruction and data have separate memory space; and data & instruction can be accessed at the same time. This is newer approach to von Neumann architecture.

Basic Organization of Microcomputer



1. **MP**
   i. **ALU**
   - This unit executes all arithmetic and logical operations as specified by instruction set; and produces output.
   - The results of addition, subtraction, and logical operations (AND, OR, XOR) are stored in the registers or in memory unit or sent to output unit.

   ii. **Register unit**
   - Consists of various registers.
   - Used for temporary storage of data during execution of data.

   iii. **CU**
   - Controls the operations of different instructions.
   - Provides necessary timing and control signals to all the operations in the MP and peripherals including memory.

2. **Memory**

→ Stores binary information such as instruction and data, and provide these information to MP when required.

→ To execute programs, the MP reads data and instructions from memory and performs the computing operations.

3. **System bus**

→ The system bus is a communication path between MP and peripherals.

→ It is used to carry data, address and control signals.

→ It consists:
   - Data bus: carries data
   - Address bus: carries address
   - Control bus: carries control signals

4. **I/O bus**

→ Input unit is used to input instruction or data to the MP externally.

→ Output unit is used to carry out the information from the MP unit.

Unit 2
Basic Computer Architecture

SAP-1 Architecture

The Simple-As-Possible (SAP)-1 computer is a very basic model of a microprocessor explained by Albert Paul Malvino. The SAP-1 design contains the basic necessities for a functional Microprocessor. Its primary purpose is to develop a basic understanding of how a microprocessor works, interacts with memory and other parts of the system like input and output. The instruction set is very limited and is simple.

SAP (Simple-As-Possible)-1 is the first stage in the evolution toward modern computers.

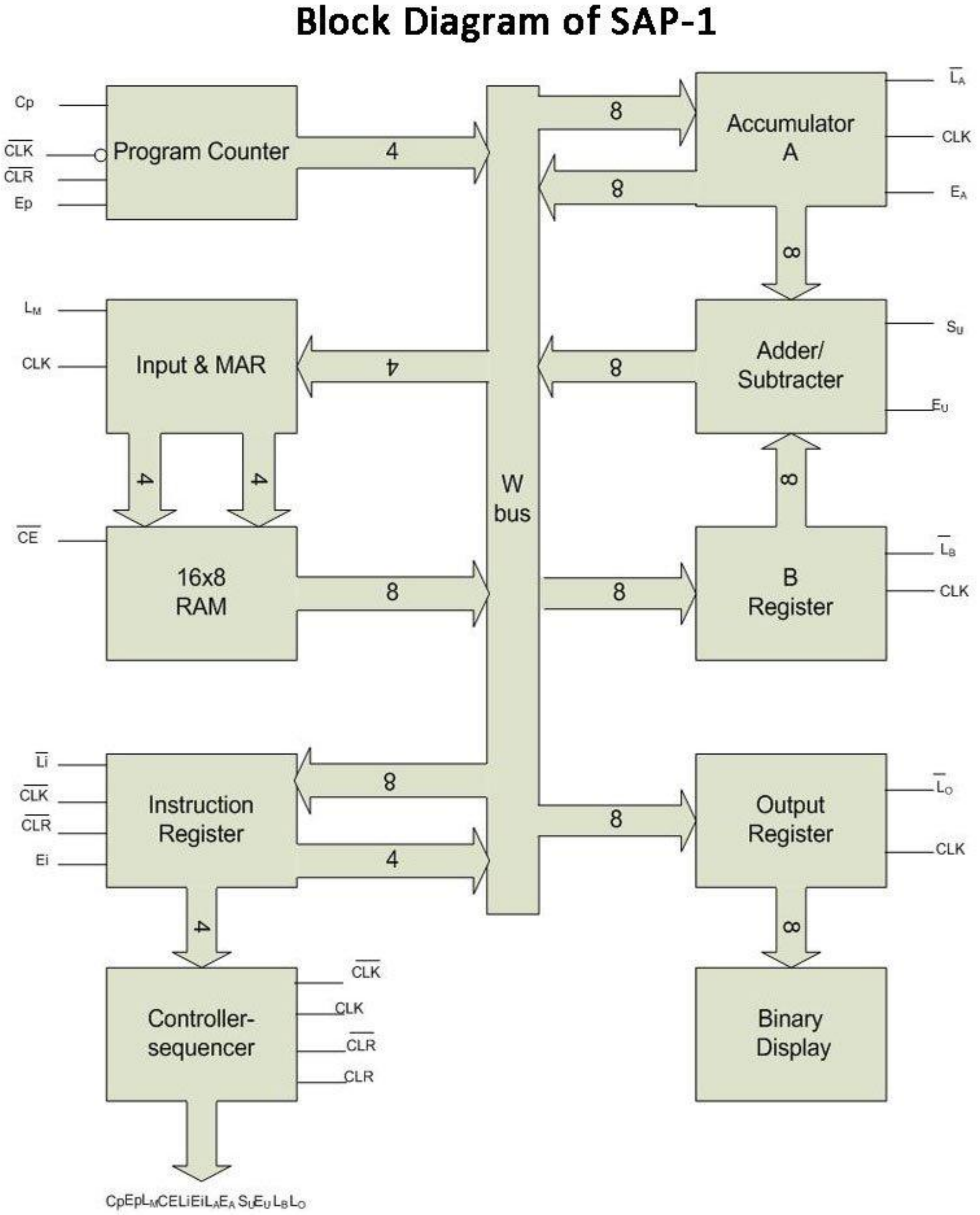


Figure: Architecture of SAP-1 Microprocessor/Computer

SAP is Simple-As-Possible Computer. The type of computer is specially designed for the academic purpose and nothing has to do with the commercial use. The architecture is 8 bits and comprises of 16 X 8 memory i.e. 16 memory location with 8 bits in each location, therefore, need 4 address lines which either comes from the PC (Program Counter which may be called instruction pointer) during computer run phase or may come from the 4 address switches during the program phase. All instructions (5 only) get stored in this memory. It means SAP cannot store program having more than 16 instructions.

SAP can only perform addition and subtraction and no logical operation. These arithmetic operations are performed by an adder/subtractor unit.

There is one general purpose register (B register) used to hold one operand of the arithmetic operation while another is kept by the accumulator register of the SAP-1.

In addition, there are 8 LEDs which work as output unit and connected with the 8 bit output register.

All timely moment of data or activities are performed by the controller/sequencer part of the SAP-1.

Program Counter

- It counts from 0000 to 1111.
- It signals the memory address of next instruction to be fetched and executed.

Inputs and MAR (Memory Address Register)

- During a computer run, the address in PC is latched into Memory Address Register (MAR).

The RAM

- The program code to be executed and data for SAP1 computer is stored here.
- During a computer run, the RAM receives 4-bit addresses from MAR and a read operation is performed. Hence, the instruction or data word stored in RAM is placed on the W bus for use by some other part of the computer.
- It is asynchronous RAM, which means that the output data is available as soon as valid address and control signal are applied.

Instruction Register

- IR contains the instruction (composed of OPCODE+ADDRESS) to be executed by SAP1 computer.

Controller-Sequencer

- it generates the control signals for each block so that actions occur in desired sequence. CLK signal is used to synchronize the overall operation of the SAP1 computer.
- A 12-bit word comes out of the Controller-Sequencer block. This control word determines how the registers will react to the next positive CLK edge.

Accumulator

- It is a 8 bit buffer register that stores intermediate results during a computer run.
- It is always one of the operands of ADD, SUB and OUT instructions.

Adder/Subtractor

- it is a 2's complement adder-subtractor.

- this module is asynchronous (unclocked), which means that its contents can change as soon as the input words change

B-register

- It is 8 bit buffer register which is primarily used to hold the other operand (one operand is always accumulator) of mathematical operations.

Output Register

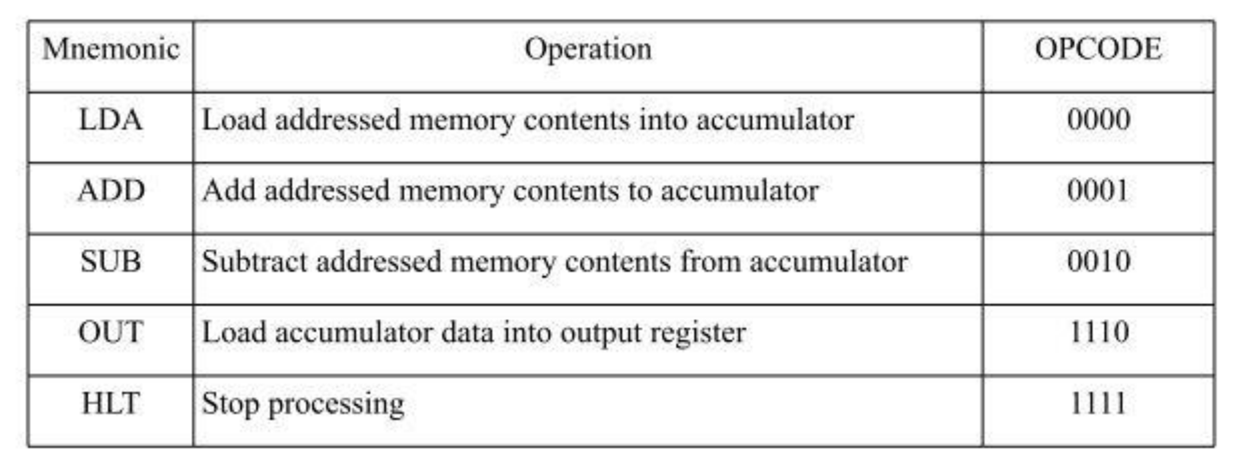
- This registers hold the output of OUT instruction.

Binary Display

- It is a row of eight LEDs to show the contents of output register.
- Binary display unit is the output device for the SAP-1 microprocessor.

SAP-1 Instructions

SAP-1 instruction set consists of following instructions



Mnemonic	Operation	OPCODE
LDA	Load addressed memory contents into accumulator	0000
ADD	Add addressed memory contents to accumulator	0001
SUB	Subtract addressed memory contents from accumulator	0010
OUT	Load accumulator data into output register	1110
HLT	Stop processing	1111

The instruction format of SAP-1 Computer is

(XXXX) (XXXX)

The first four bits make the opcode while the last four bits make the operand (address).

Machine cycle and Instruction cycle

SAP1 has six T-states (three fetch and three execute cycles) reserved for each instruction. Not all instructions require all the six T-states for execution. The unused T- state is marked as No Operation (NOP) cycle. Each T-state is called a machine cycle for SAP1. A ring counter is used to generate a T-state at every falling edge of clock pulse. The ring counter output is reset after the 6th T-state.

FETCH CYCLE – T1, T2, T3 machine cycle

EXECUTE CYCLE - T4, T5, T6 machine cycle

- Complete code includes opcode and operand
- One instruction is executed in one instruction cycle

- Instruction cycle may consist of many machine cycles
- For SAP-1, Instruction cycle = Machine cycle
- Instruction cycle = Fetch cycle + Execution cycle
- Fetch cycle is generally same for all instructions
- Complete code includes opcode and operand
- Like LDA 04H ⟹ 0000 0100
- One instruction is executed in one instruction cycle

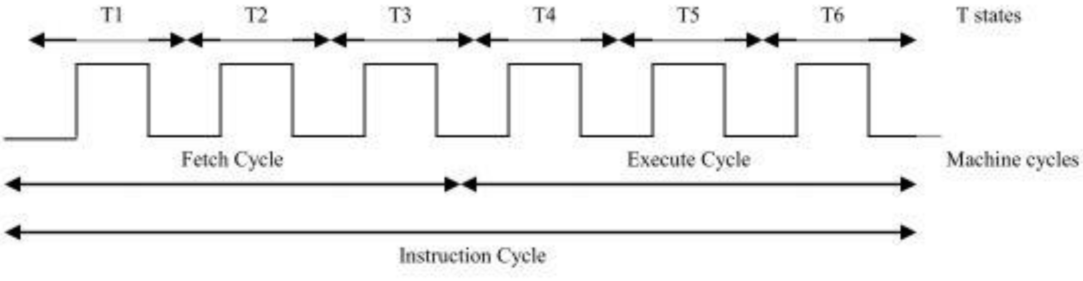


Fig: Instruction Cycle

Fetch and Execution Cycle of SAP-1 instructions

- SAP-1 instruction cycle: 3 clock cycles to fetch and decode phase, 3 clock cycles to execute
- the first three states are:
    1. address
    2. increment
    3. memory
- controller has a 6-bit ring counter which continuously cycles from 000001 up to 100000 then resets (must be set to 000001 when we initialize the computer)
- ring counter is clocked on clock high-to-low transition, most of the other circuits in the computer on clock low-to-high transition

Fetch Cycle

- Address state: enable PC to bus three-state output, MAR load line
- Increment state: enable PC increment (and perhaps wait for memory access time)
- Memory state: enable memory CE, IR load line
- IR is loaded on the low-to-high clock transition, so stabilizes before state 4 is entered

  t1: MAR ← PC
  t2: PC ← PC +1
  t3: IR ← RAM

Execution Cycle -- LDA

- state 4: enable IR to bus three-state output, MAR load line
- state 5: enable memory CE, accumulator load line
- state 6: enable nothing

  t4: MAR ← (IR (Address of operand))
  t5: Accumulator ← RAM
  t6: nothing

Execution Cycle -- ADD

- state 4: enable IR to bus three-state output, MAR load line
- state 5: enable memory CE, register B load line
- state 6: enable add, ALU to bus three-state output, accumulator load line

  t4: MAR ← (IR (Address of B))

t5: B ← RAM
t6: Accumulator ← Accumulator + B

Micro program

- each SAP-1 block has some control lines:
  - each three-state driver has an enable line which connects the driver to the bus
  - the program counter also has a count line which, when high, increments the contents on the next low-to-high clock transition
  - all registers have a load line (active low)
  - the ALU has a subtract line which is high for subtraction and low for addition
- implementing the SAP-1 instructions means raising and lowering these control lines at the appropriate times
- these 12 control lines are the micro program word
- controller must, on each clock cycle, produce 12 bits
- some of these bits are on-off (e.g. three-state output lines) and have a "default off" state
- some of these bits are A/B (e.g. the add-subtract line) and have a "default don't care"
- some bits are active high, some are active low

Controller Implementation

- How to generate micro words?
- if a bit is only on during one cycle, connect it to the corresponding ring counter bit
- if a bit is only on for an instruction, connect it to that instruction (as decoded by a 4-bit 1-of-16 decoder)
- if a bit is only on for an instruction and a cycle, connect it to the AND of the ring counter bit and the decoder output
- if a bit is on for multiple instruction and/or cycles, work out the truth table and use AND/OR or multiplexers to implement it
- alternative: use a ROM

SAP-2 Architecture

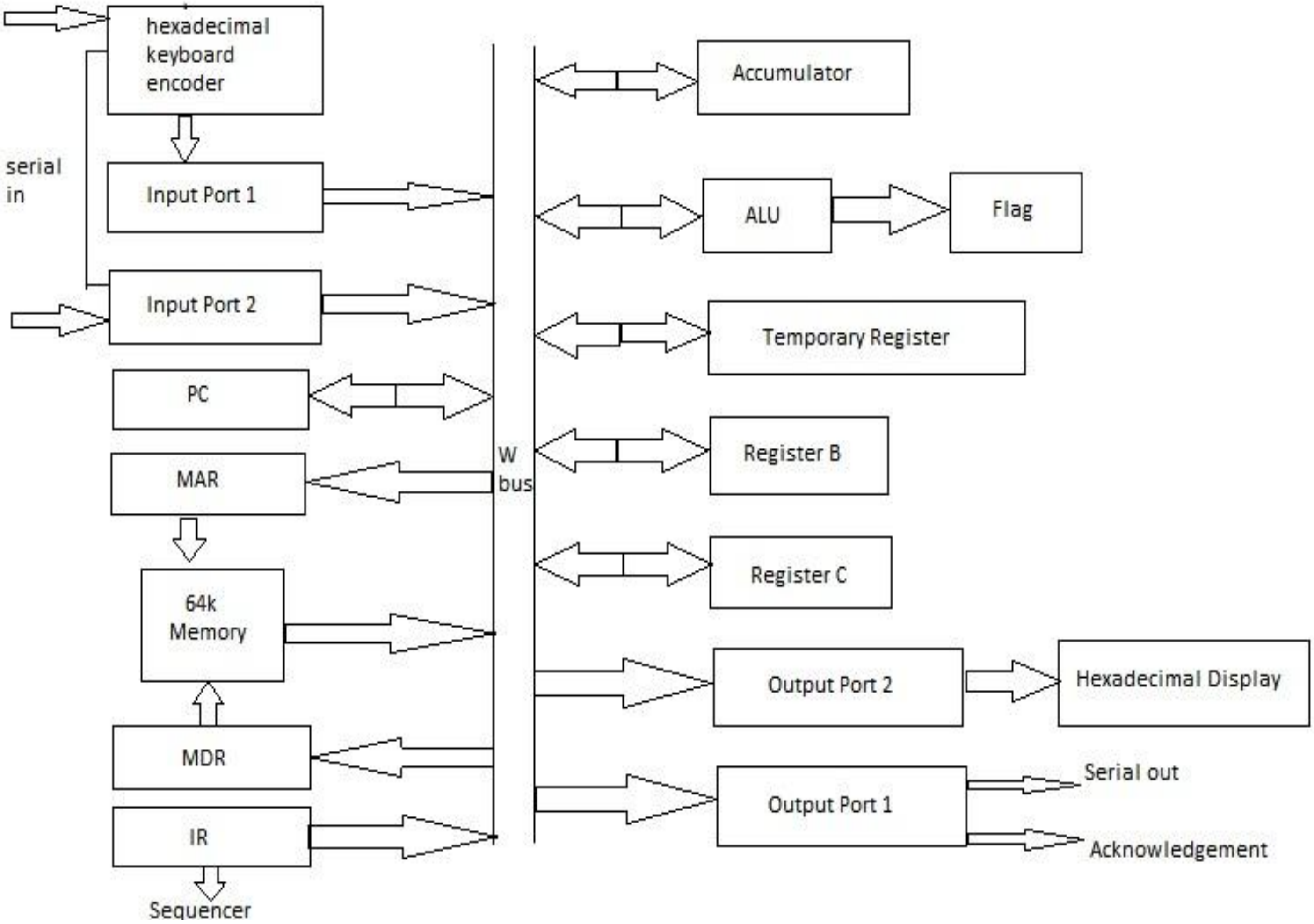


Fig: Architecture of SAP-1 Microprocessor/Computer

- **Hexadecimal Keyboard Encoder**: The hexadecimal keyboard encoder receives the data from outer environment and converts it into hexadecimal form. The system can understand and send them to the input port.
- **Input Ports**: The SAP-2 contains two input ports which input the data in the system in the most convenient way.
- **PC**: PC is the program counter that holds the address of the next instruction to be fetched. It initializes from 0000H to 1111H during the execution.
- **MAR**: MAR is the memory address register that stores the complete format of the address sent by the program counter. It stores the final address of the memory word that needs some computations.
- **64 K Memory**: It contains 64 K memory where data and instruction reside. All the computations are performed relative to the memory.
- **MDR**: MDR is the Memory Data Register which stores the data or operand that is fetched from the memory which is needed for computation.
- **IR**: The IR is the Instruction Register that holds the complete format of the Instruction that is to be executed.
- **Control Sequencer**: It provides necessary timing signals like T0, T1, T2, ….. and control signals providing the direction for executing the program.
- **Accumulator**: The result of all the mathematical operations is stored in accumulator. It is one of the operand of ADD, OUT, SUB instruction. It is also known as processor register.
- **ALU and Flag**: The ALU perform all the arithmetic and logical calculations. The flag reflect the intermediate changes on the values during execution.
- **Temporary register, B, C**: They are the second operand of the mathematical operations. The register B and C is accessible to the programmer.
- **Output Ports**: It consists of two output ports to show the result of OUT instruction.

- **Hexadecimal Display**: Unlike Sap-1 which has binary display, Sap-2 has a hexadecimal display to show outputs in the LEDs.

Architectural differences with SAP-1

- Jump Instructions: loadable PC
- 16-bit program counter
- 8-bit opcode, 42 instructions
- 2 input ports, 2 output ports
- 2K ROM, up to 62K RAM (16-bit addresses) with read and write
- memory data register (MDR) buffers reads and writes
- accumulator can write to bus
- Temporary, B and C registers
- 16 arithmetic and logic operations in ALU
- sign and zero flag

Bidirectional registers

- connect the inputs to the outputs
- load and enable never simultaneously active
- on load, outputs are 3-state, input is taken from the bus
- on enable, inputs are ignored, output goes to the bus
- 1/2 as many pins
- 1/2 as much bus capacitance

Flags

- 2 flip flops, sign flag and zero flag
- set during arithmetic and logic operations to reflect final accumulator contents
- JM jumps only if the sign flag is set (minus result)
- JZ jumps only if the zero flag is set (zero result)
- JNZ jumps if the zero flag is clear (non-zero result)

  Q. What is the value of the sign bit if the accumulator contents are zero?

In-Class Exercise

- work in groups of up to 3
- design a circuit to implement the flags
- inputs are: 8 bits from the accumulator, clock (use the positive-going edge), 1 L_F control line (active high)
- outputs are the flags: Z_F, S_F

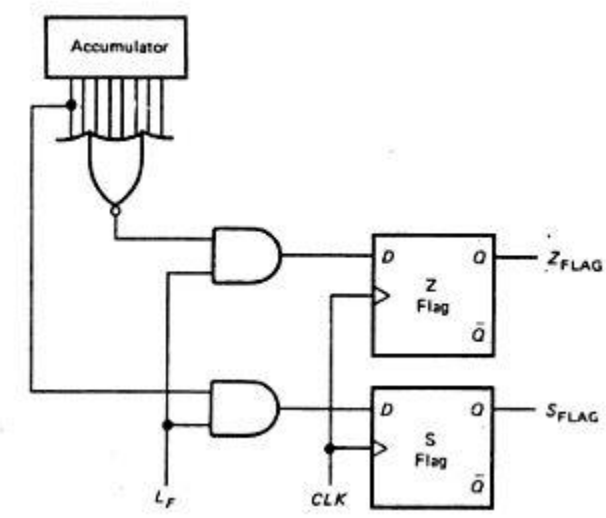


Fig: Setting the flags

SAP-2 Instruction Sets

- same fetch cycle (T_1, T_2, T_3) as SAP-1
- memory reference instructions: LDA, STA (3 bytes, lower byte before higher byte)
- immediate instructions: MVI reg, value (2 bytes)
- register instructions: MOV, ADD and SUB, INR and DCR, ANA, ORA, XRA, CMA (1 byte), ANI, ORI, XRI (2 bytes)
- jump and call instructions: JMP, JM, JZ, JNZ, CALL (3 bytes), RET (1 byte)
- CALL saves return address in memory `FFFEH` and `FFFFH`
- NOP, HLT, RAL, RAR (1 byte), IN, OUT (2 bytes)

In-Class Exercise 1

- hand-assemble the following program starting at address 2000H
- opcodes are: “IN” is DBH, "MOV B,A" is 47H, "DCR A" is 3DH, "MOV C,A" is 4FH, "ANA B" is A0H, "MOV A,C" is 79H, JNZ is 79H
- What does the program do?

```
START   IN 01H
LOOP    MOV B, A
        DCR A
        MOV C, A
        ANA B
        MOV A, C
        JNZ LOOP
DONE
```

Fig: SAP-2 Instruction Sets

(I have explained in class.)

Differences between SAP-1 and SAP-2 Architecture

SAP-1	Sap-2
It has 8-bit bus.	It has 16-bit bus.
PC is 4-bit.	PC is 16-bit.
It does not have hexadecimal keyboard encoder.	It has hexadecimal keyboard encoder.
It has single input.	It has two input ports.
MAR receives 4-bit address from PC.	MAR receives 16-bit address from PC.
It does not have ROM.	It has 2 KB ROM.
It has 16 KB memory.	It has 62 KB memory.
It does not have MDR.	It has MDR.
It has only adder/subtractor.	It has ALU.
It does not have flag.	It has 2 flags.
It does not have temporary register.	It has temporary register.
It has single register (B).	It has 2 registers (B and C).
It has single output port.	It has 2 output ports.
It has 5 instruction sets.	It has 42 instruction sets.

Unit 3
Instruction Cycle

T-state

→ It is the time period of a single cycle of the clock frequency.

Machine Cycle

→ The number of T-states required performing a read or writing operation either from memory or I/O.

→ A machine cycle may consist of three to six T-states.

Instruction Cycle

→ It is the total number of machine cycles required to execute a complete instruction.

Machine Cycles of 8085 microprocessor

i) Op-code fetch cycle

→ The MP uses this cycle to take the op-code of an instruction from the memory location to processor.

→ The op-code is taken from memory and transferred to instruction register for decoding and execution.

→ The time required to complete this cycle is 4 to 6 T-states.

ii) Memory read cycle

→ The MP executes these cycles to read data from memory.

→ The address of memory location is given by instruction.

→ The time required to complete the memory read cycle is 3 T-states.

iii) Memory write cycle

→ The MP executes these cycles to write data to memory.

→ The address of memory is given by instructions.

→ The time required to complete the memory write cycle is 3 T-states.

iv) I/O read cycle

→ The MP executes these cycles to read data from I/O devices.

→ The address of I/O port is given by instruction.

→ The time required to complete the I/O read cycle is 3 T-states.

v) I/O write cycle

→ The MP executes these cycles to write data into I/O devices.

→ The address of I/O port is given by instruction.

→ The time required to complete the I/O write cycle is 3 T-states.

vi) Interrupt acknowledge cycle

→ In the response to interrupt request input **INTR**, the MP executes these cycles to get information from the interrupting devices.

→ The time required to complete this cycle is 3 T-states.

Fetch & Execute Operation: Timing Diagram

→ The graphical representation of status of various signals involved during a machine cycle with respect to time is called timing diagram.

→ This gives basic idea of what is happening in the system when the instruction is getting fetched and executed, at what instant which signal is getting activated.

→ The signals involved during machine cycle are CLK, $A_{15} – A_8$, $AD_7 – AD_0$, IO/M(bar), RD(bar), WR(bar) and S_1, S_0.

IO/M(bar)	S_1	S_0	Operation
0	0	0	Halt
0	0	1	Memory write
0	1	0	Memory read
0	1	1	Op-code fetch
1	0	1	IO write
1	1	0	IO read
1	1	1	Interrupt acknowledge

Timing diagram for op-code fetch cycle

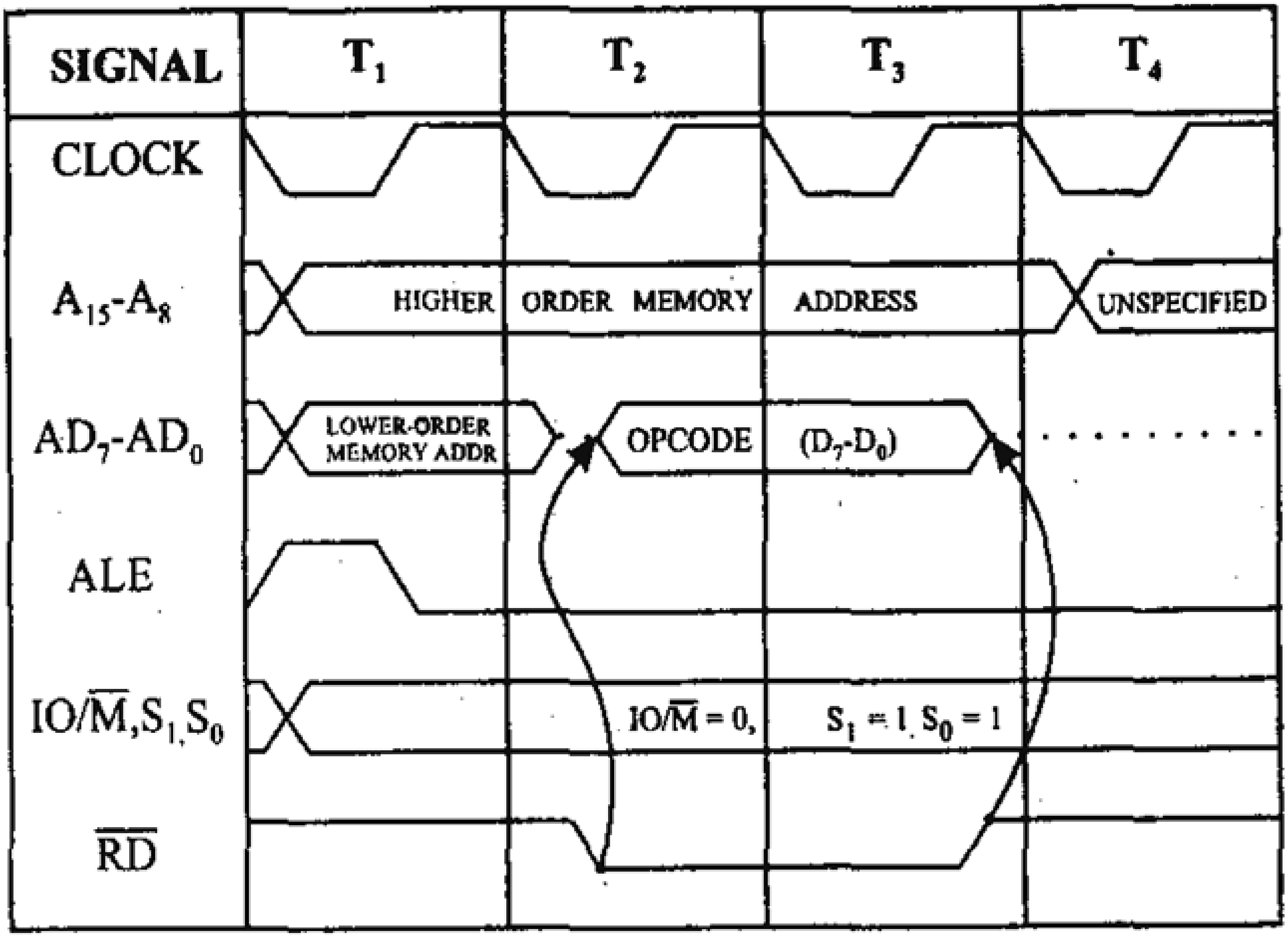


Fig: Timing Diagram for Op-code Fetch Machine Cycle

The op-code fetch timing diagram can be explained as below:

i) The MP places the 16-bit memory address from the program counter on address bus. At time period T_1, the higher order memory address is placed on the address lines $A_{15} – A_8$. When ALE is high, the lower address is placed on the bus $AD_7 – AD_0$. The status signal IO/M(bar) goes low indicating the memory operation and two status signals $S_1 = 1$, $S_0 = 1$ to indicate op-code fetch operation.

ii) At time period T_2, the MP sends RD(bar) control line to enable the memory read. When memory is enabled with RD(bar) signal, the op-code value from the addressed memory location is placed on the data bus with ALE low.

iii) The op-code value is reached at processor register during T_3 time period. When data (op-code value) is arrived, the RD(bar) signal goes high. It causes the bus to go into high impedance state.

iv) The op-code byte is placed in instruction decoder of MP and the op-code is decoded and executed. This happens during time period T_4.

Timing diagram for memory read cycle

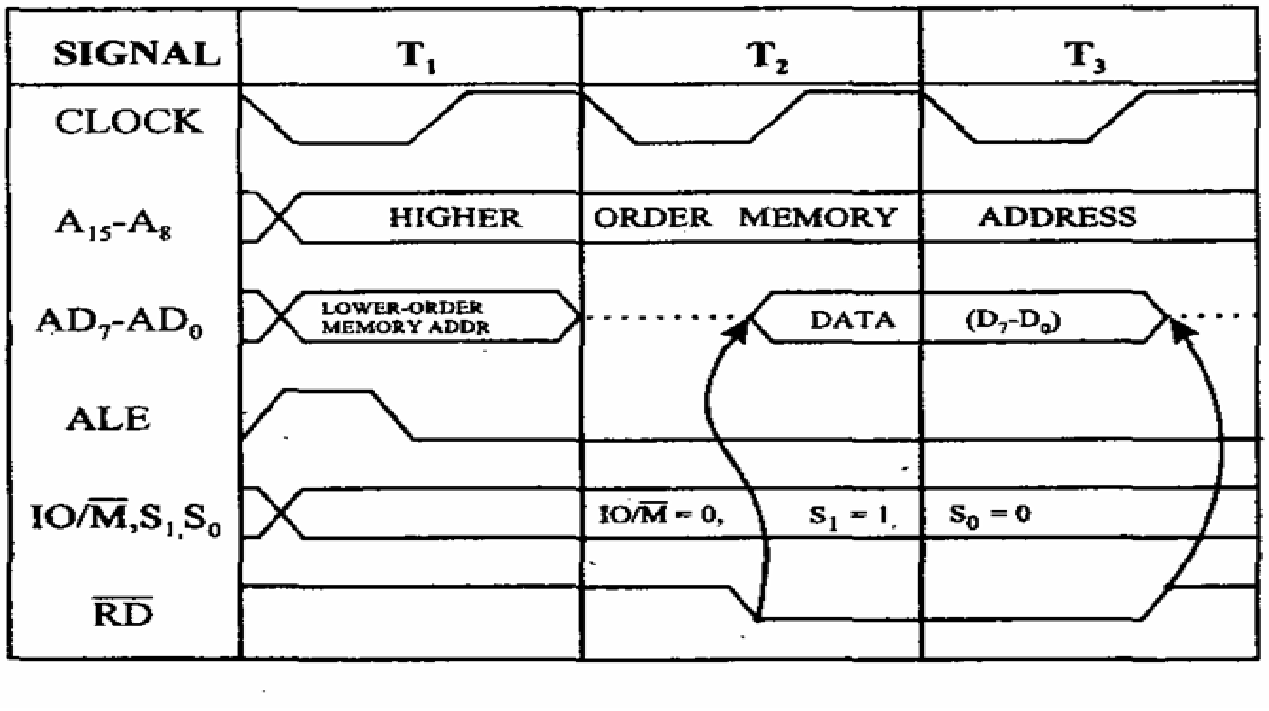


Fig: Timing Diagram for Memory Read Machine Cycle

The memory read timing diagram can be explained as below:

i) The MP places the 16-bit memory address from the program counter on address bus. At time period T_1, the higher order memory address is placed on the address lines A_{15} – A_8. When ALE is high, the lower address is placed on the bus AD_7 – AD_0. The status signal IO/M(bar) goes low indicating the memory operation and two status signals S_1 = 1, S_0 = 0 to indicate memory read operation.

ii) At time period T_2, the MP sends RD(bar) control line to enable the memory read. When memory is enabled with RD(bar) signal, the data from the addressed memory location is placed on the data bus with ALE low.

iii) The data is reached at processor register during T_3 state. When data is arrived, the RD(bar) signal goes high. It causes the bus to go into high impedance state.

Timing diagram for memory write cycle

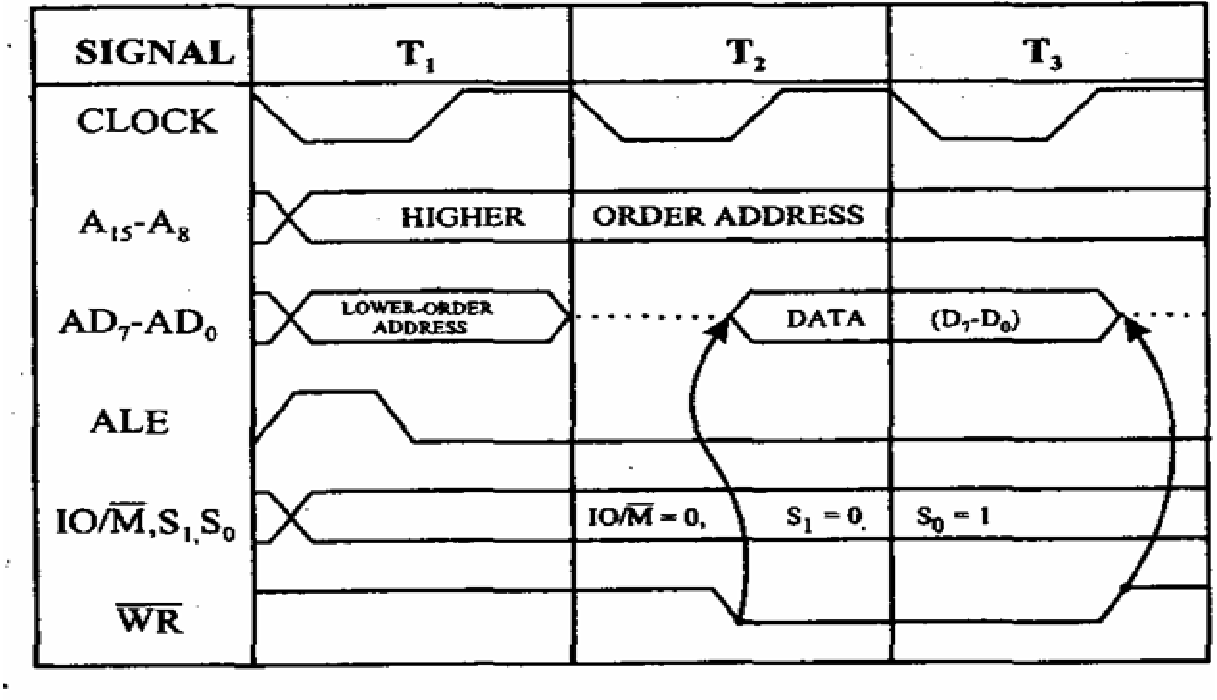


Fig: Timing Diagram for Memory Write Machine Cycle

The memory write timing diagram can be explained as below:

i) The MP places the 16-bit memory address from the program counter on address bus. At time period T_1, the higher order memory address is placed on the address lines $A_{15} – A_8$. When ALE is high, the lower address is placed on the bus $AD_7 – AD_0$. The status signal IO/M(bar) goes low indicating the memory operation and two status signals $S_1 = 0$, $S_0 = 1$ to indicate memory write operation.

ii) At time period T_2, the MP sends WR(bar) control line to enable the memory write. When memory is enabled with WR(bar) signal, the data from the processor is placed on the addressed location with ALE low.

iii) The data is reached at memory location during T_3 state. When data is reached, the WR(bar) signal goes high. It causes the bus to go into high impedance state.

Timing diagram for IO read cycle

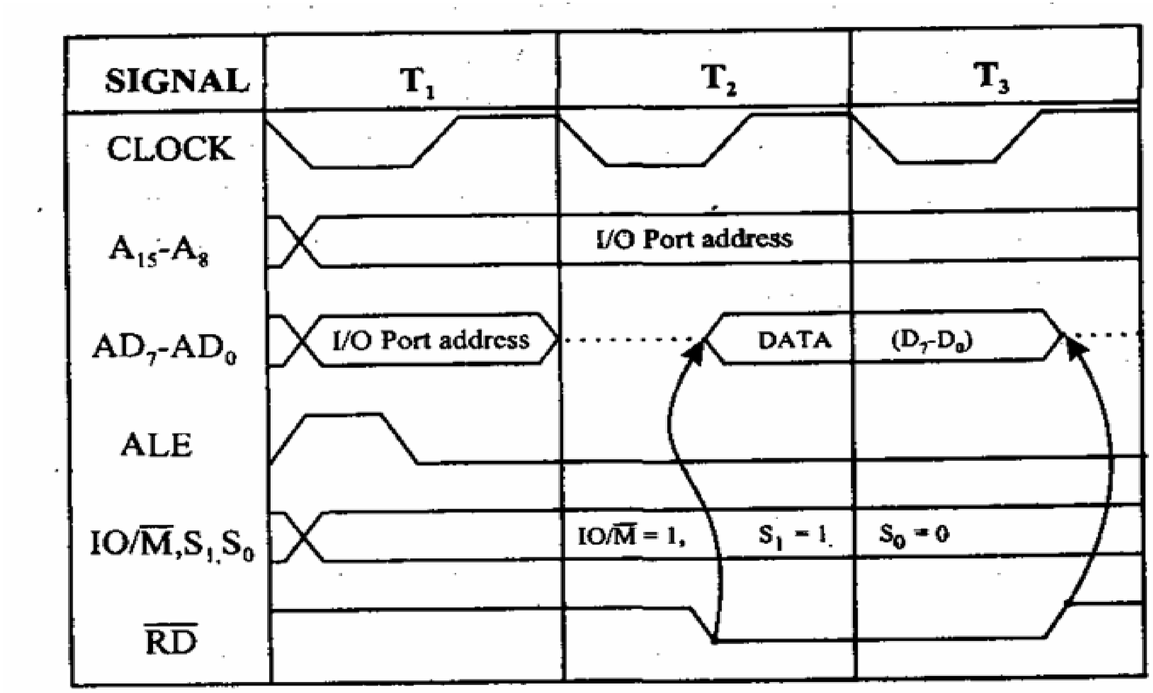


Fig: Timing Diagram for IO Read Machine Cycle

Describe yourself.

Timing diagram for IO write cycle

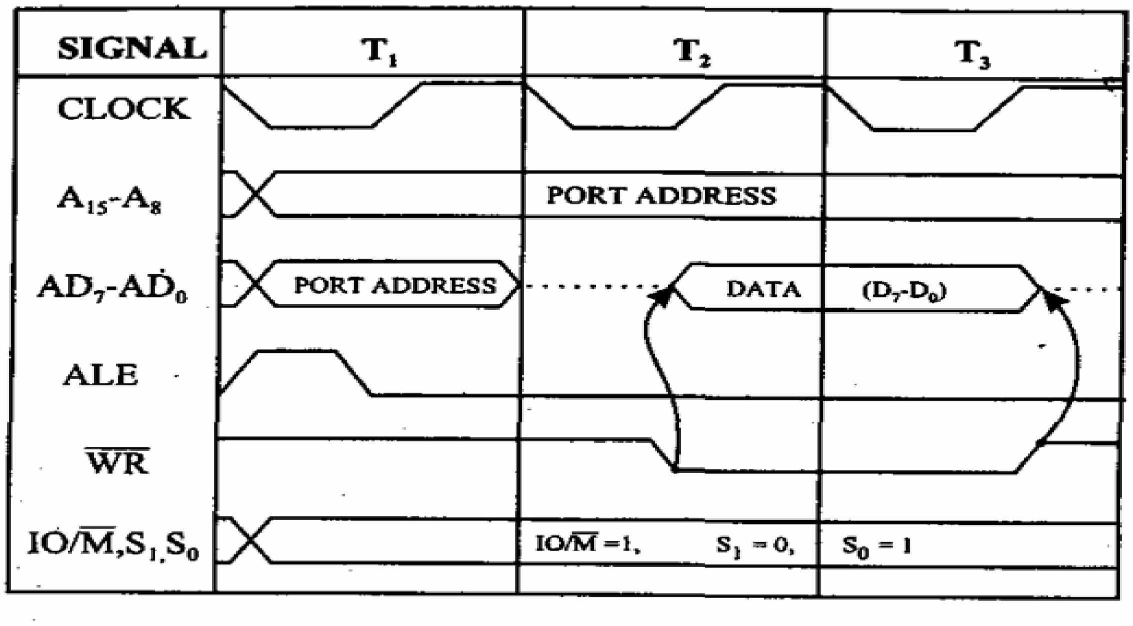


Fig: Timing Diagram for IO Write Machine Cycle

Describe yourself.

Timing Diagram of MOV, MVI, IN, OUT, LDA, STA

i) MOV

E.g. MOV A, B

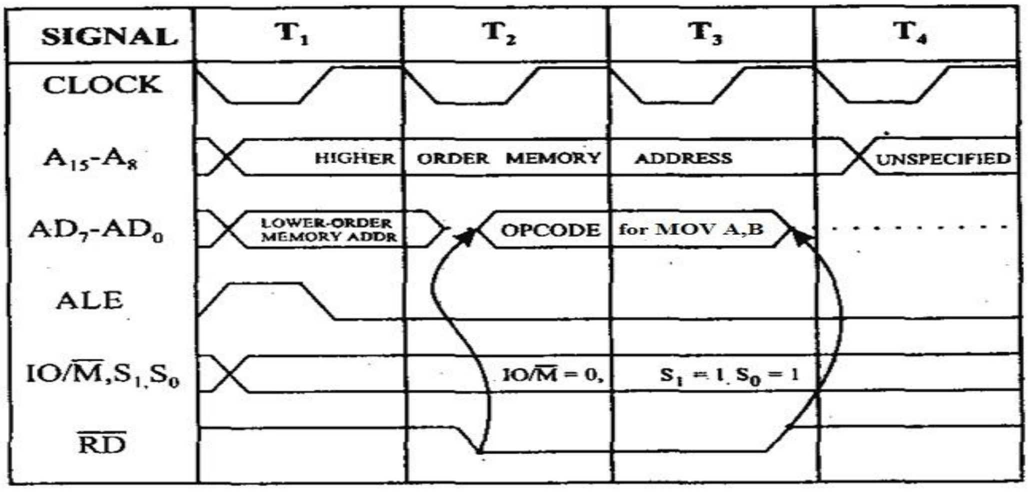


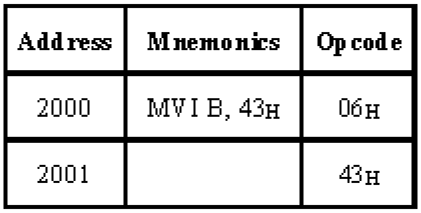
Fig: Timing Diagram for MOV A,B

ii) MVI

Timing diagram for MVI B, 43H

→ Fetching the Op-code 06H from the memory 2000H. (OF machine cycle)

→ Read (move) the data 43H from memory 2001H. (memory read)



Address	Mnemonics	Opcode
2000	MVI B, 43H	06H
2001		43H

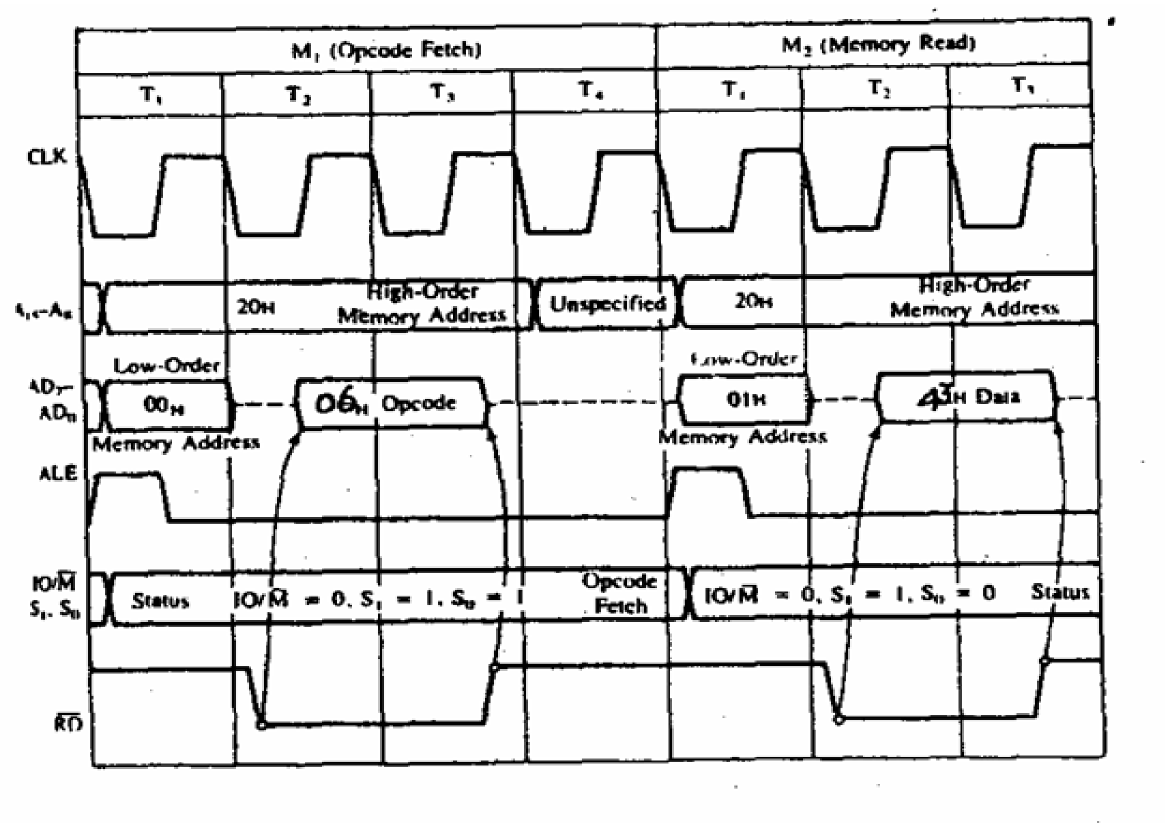


Fig: Timing Diagram for MVI B, 43H

iii) IN

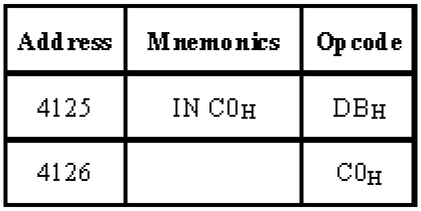
Timing diagram for IN C0H

→ Fetching the Op-code DBH from the memory 4125H.

→ Read the port address C0H from 4126H.

→ Read the content of port C0H and send it to the accumulator.

→ Let the content of port is 5EH.



Address	Mnemonics	Opcode
4125	IN $C0_H$	DB_H
4126		$C0_H$

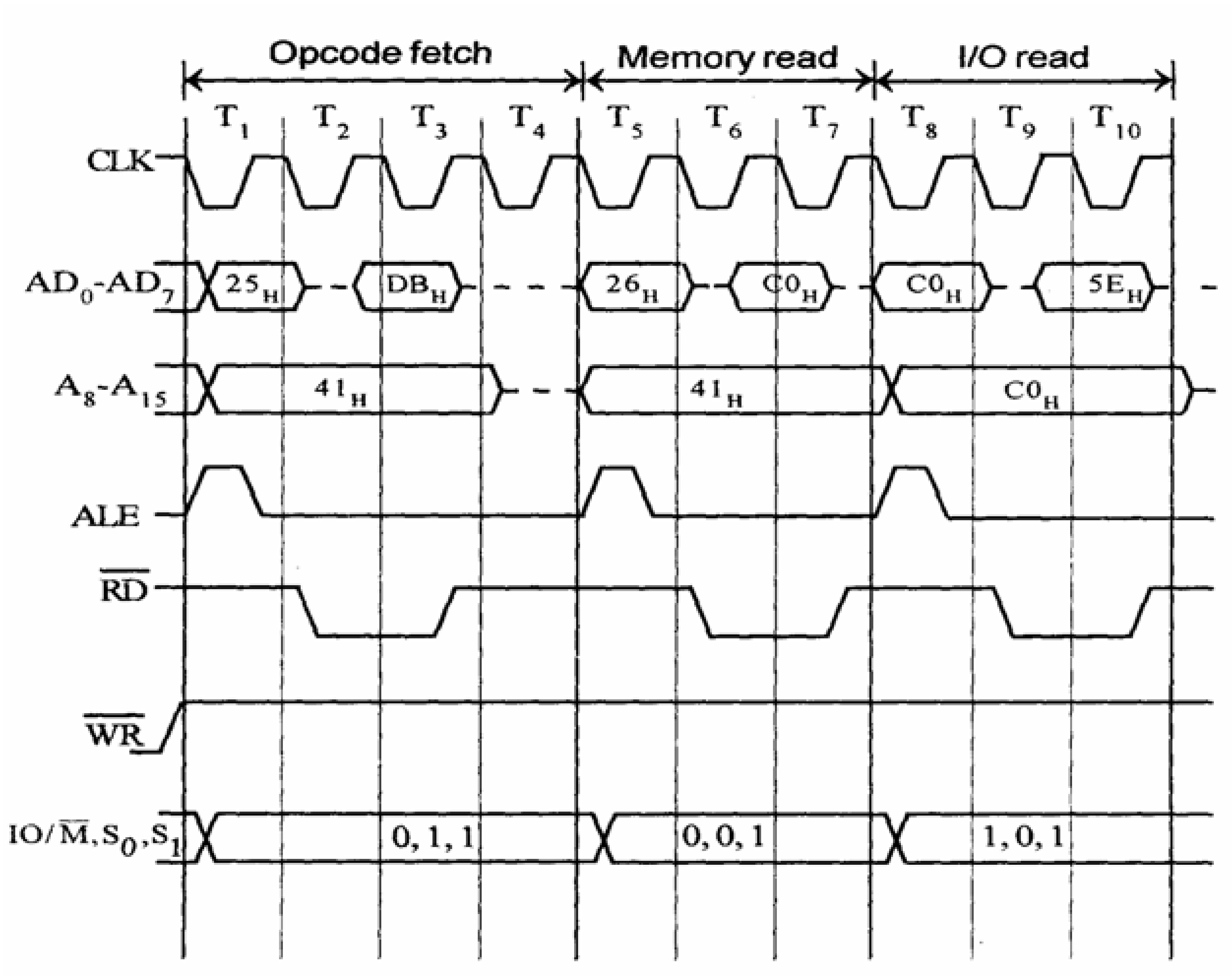


Fig: Timing Diagram for IN C0H

iv) OUT

E.g. OUT 07H

Do yourself.

v) LDA

E.g. LDA 2030H

Do yourself.

vi) STA

<u>Timing diagram for STA 526AH</u>

→ STA means Store Accumulator -The content of the accumulator is stored in the specified address (526A).

→ The op-code of the STA instruction is said to be 32H. It is fetched from the memory 41FFH (see fig). - OF machine cycle

→ Then the lower order memory address is read (6A). – Memory Read Machine Cycle

→ Read the higher order memory address (52).- Memory Read Machine Cycle

→ The combination of both the addresses is considered and the content from accumulator is written in 526A. – Memory Write Machine Cycle

→ Assume the memory address for the instruction and let the content of accumulator is C7H. So, C7H from accumulator is now stored in 526A

Address	Mnemonics	Opcode
4125	IN C0$_H$	DB$_H$
4126		C0$_H$

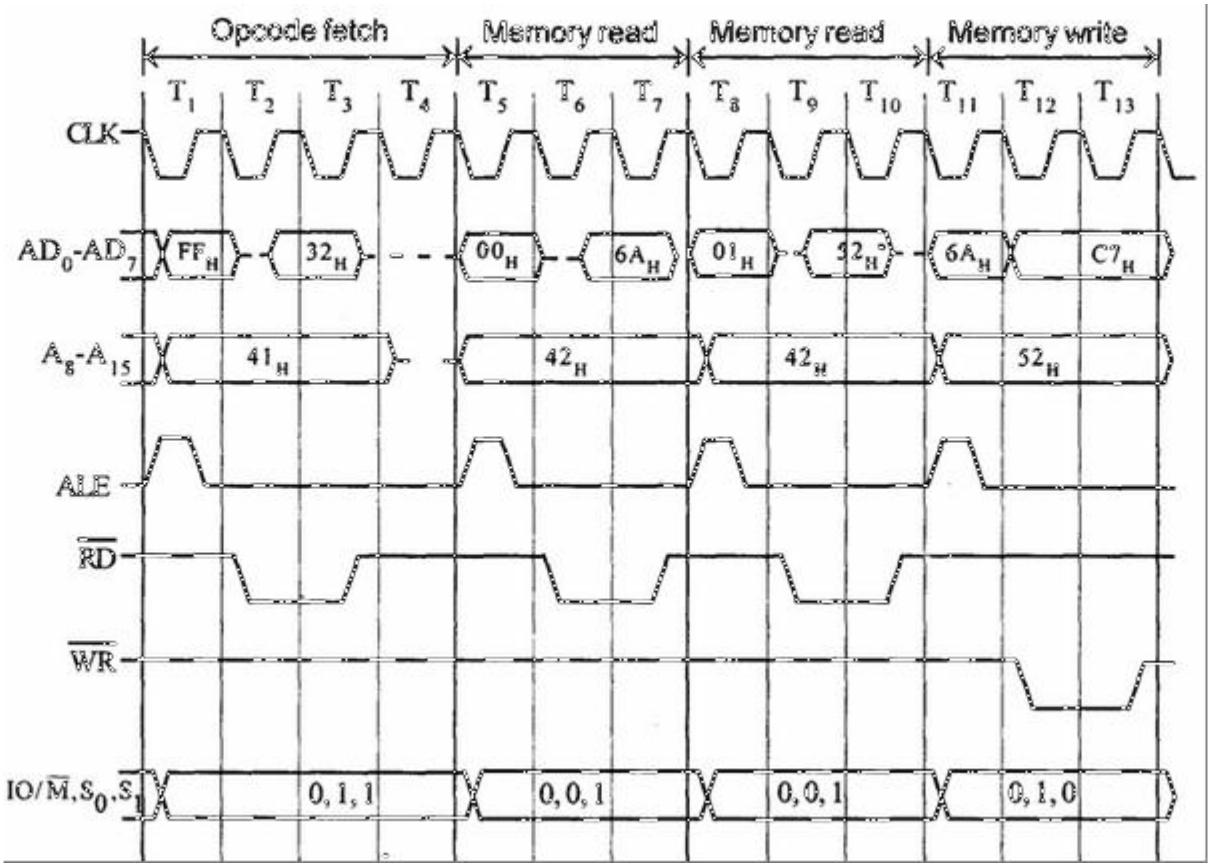


Fig: Timing Diagram for STA 526AH

Fetch and Execute Overlap

→ The instruction cycle consists fetching the instruction from memory called fetch cycle and decode & execute the instruction called execution cycle.

→ During the fetch, the instruction is read from the memory. During execution, instruction is decoded and then executed. Decoding of instruction need not to be referenced (i.e. memory reference).

→ So, during execution of instruction, there is a time that does not need memory. At that time, we can fetch the instruction from memory. This fetching of next instruction while execution of an instruction is known as fetch and execute overlap.

Unit 4
Intel 8085/8086/8088

Intel 8085 microprocessor

Functional Block Diagram

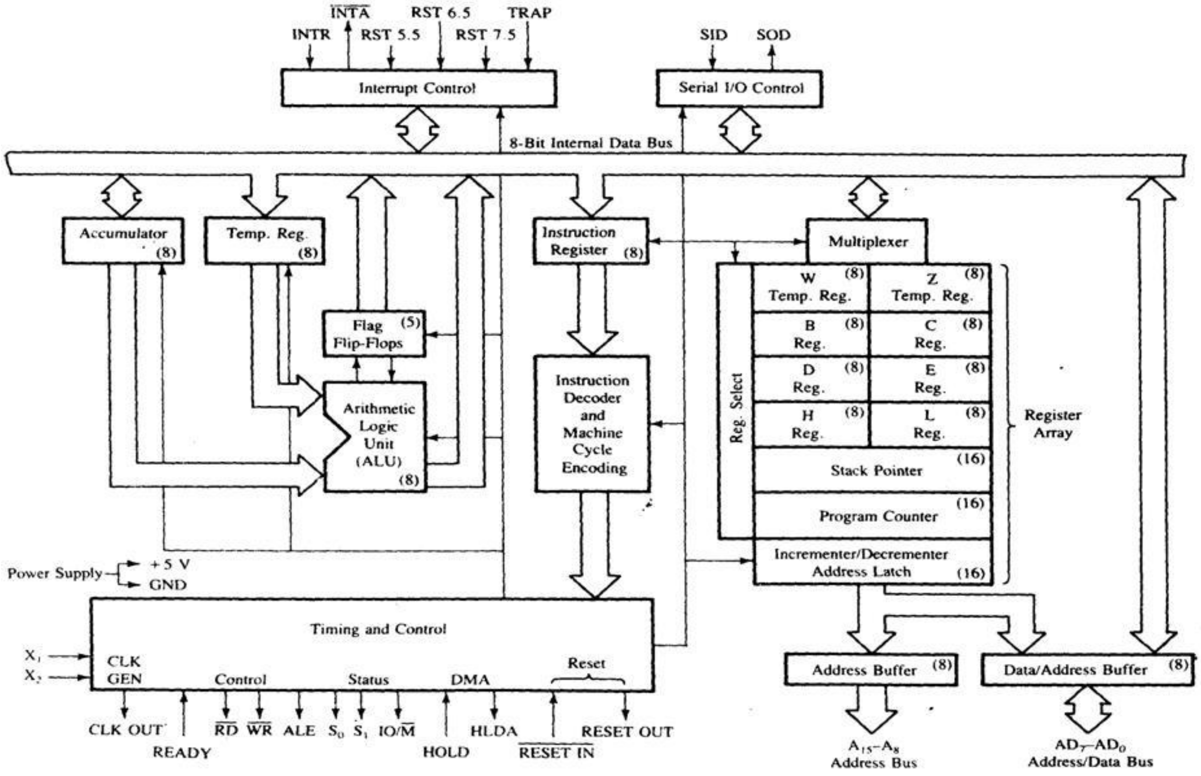


Fig: Functional Block Diagram of 8085 Microprocessor

1. ALU
   - → The ALU performs the actual numerical and logic operation such as ‘add’, ‘subtract’, ‘AND’, ‘OR’ etc.
   - → Uses data from memory and from Accumulator to perform arithmetic operation and always stores result of operation in Accumulator.
   - → The ALU consists of accumulator, flag register and temporary register.

   a. Accumulator
      - → The accumulator is an 8-bit register that is a part of arithmetic/logic unit (ALU). This register is used to store 8-bit data and to perform arithmetic and logical operations. The result of an operation is stored in the accumulator.
      - → The accumulator is also identified as register A.

   b. Flag register
      - → 8085 has 8-bit flag register. There are only 5 active flags.

S	Z		AC		P		C

Fig: 8085 flag register

- → Flags are flip-flops which are used to indicate the status of the accumulator and other register after the completion of operation.
- → These flip-flops are set or reset according to the data condition of the result in the accumulator and other registers.

i. Sign flag(S):

- → Sign flag indicates whether the result of a mathematical or logical operation is negative or positive.

→ If the result is negative, this flag will be set (i.e. S=1) and if the result is positive, the flag will be reset (i.e. S=0).

ii. Zero flag (Z):

→ Zero flag indicates whether the result of a mathematical or logical operation is zero or not.

→ If the result of current operation is zero, the flag will be set (i.e. Z=1) otherwise the flag will be reset (Z=0).

→ This flag will be modified by the result in the accumulator as well as in the other register.

iii. Auxiliary carry flag (AC):

→ in operation when a carry is generated by bit D_3 and passes on to bit D_4,the AC flag will be set otherwise AC flag will be reset.

→ This flag is used only internally for BCD operation and is not available for the programmer to change the sequence of program with the jump instruction.

iv. Parity flag (P):

→ This flag indicates whether the current result is of even parity (no. of 1's is even) or odd parity (no. of 1's is odd).

→ If even parity, P flag will be set otherwise reset.

v. Carry flag (C):

→ This flag indicates whether during an addition or subtraction operation carry or borrow is generated or not.

→ If carry or borrow is generated, the flag will be set otherwise reset.

2. Timing and control unit

→ This unit produces all the timing and control signal for all the operation.

→ This unit synchronizes all then MP operation with the clock and generates the control signals necessary for communication between the MP and peripherals.

3. Instruction register and decoder

→ The instruction register and decoder are part of ALU. When an instruction is fetched from memory, it is loaded in the instruction register.

→ The decoder decodes the instruction and establishes the sequence of events to follow.

→ The IR is not programmable and cannot be accessed through any instruction.

4. Register array

→ The register unit of 8085 consists of

— Six general-purpose data registers B,C,D,E,H,L

— Two internal registers W and Z

— Two 16-bit address registers PC (program counter) and SP (stack pointer)

— One increment/decrement counter register

— And, one multiplexer (MUX)

→ The six general-purpose registers are used to store 8-bit data. They can be combined as register pairs BC, DE, and HL to perform some 16-bit operations.

→ The two internal registers W and Z are used to hold 8-bit data during the execution of some instructions, **CALL** and **XCHG** instructions.

→ SP is 16-bit registers used to point the address of data stored in the stack memory. It always indicates the top of the stack.

→ Pc is 16-bit register used to point the address of the next instruction to be fetched and executed stored in the memory.

5. System bus

a. Data bus

→ It carries 'data', in binary form, between MP and other external units, such as memory.

→ Typical size is 8 or 16 bits.

b. Address bus
   - → It carries ‘address’ of operand in binary form.
   - → Typical size is 16-bit.

c. Control Bus
   - → Control Bus are various lines which have specific functions for coordinating and controlling MP operations.
   - → E.g.: Read/Write control line.

6. Interrupt Control
   - → Interrupt is a signal, which suspends the routine what the MP is doing, brings the control to perform the subroutine, completes it and returns to main routine.
   - → May be hardware or software interrupts. Some interrupts may be ignored (maskable), some cannot be called non-maskable interrupts.
   - → E.g. INTR, TRAP, RST 7.5, RST 6.5, RST 5.5

7. Serial I/O Control
   - → The MP performs serial data input or output (one bit at a time). In serial transmission, data bits are sent over a single line, one bit at a time.
   - → The 8085 has two signals to implement the serial transmission: SID (serial input data) and SOD (serial output data).

Pin Configuration

Properties

Single + 5V Supply
4 Vectored Interrupts (One is Non Maskable)
Serial In/Serial Out Port
Decimal, Binary, and Double Precision Arithmetic
Direct Addressing Capability to 64K bytes of memory

The Intel 8085 is a new generation, complete 8-bit parallel central processing unit (CPU). The 8085 uses a multiplexed data bus. The address is split between the 8-bit address bus and the 8-bit data bus.

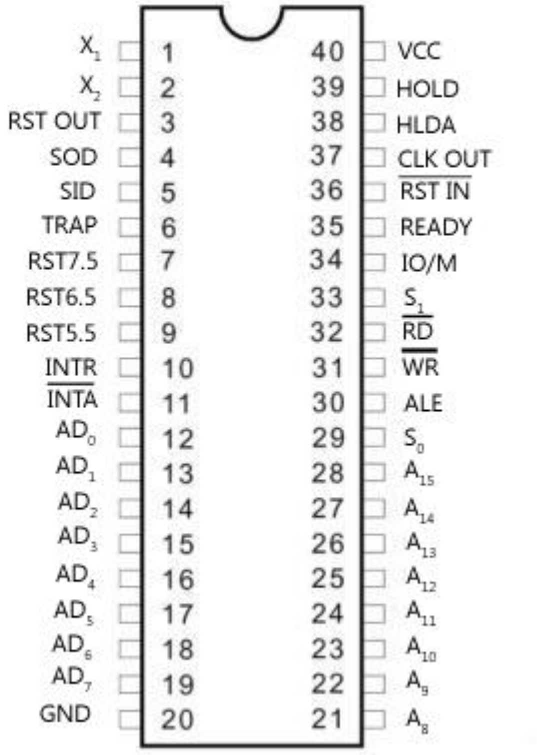


Fig: 8085 Pin Diagram

X1, X2 (Input)

A crystal (or RC, LC network) is connected at these two points. The frequency is divided into two; therefore to operate a system at 3 MHz, the crystal should have a frequency of 6 MHz.

RESET OUT (Output)

This signal indicates that the microprocessor is being reset. It is also used to reset other devices.

SOD (Output)/ SID (Input)

This signal is used for the transmission of data per bit in and out of the processor.

TRAP (Input)

It is a non-maskable interrupt and has the highest priority of any interrupt.

RST 5.5 / RST 6.5 / RST 7.5 (Inputs)

RESTART interrupts. These are the vector interrupts that transfer the program control to the specific memory locations.

RST 7.5	Highest Priority
RST 6.5	
RST 5.5	Lowest Priority

The priority of these interrupts is ordered as shown above. These interrupts have a higher priority than the INTR.

INTR (Input)

INTERRUPT REQUEST. This is used as a general purpose interrupt.

INTA (Output)

INTERRUPT ACKNOWLEDGE; this is used to acknowledge an interrupt.

AD0 -AD 7 (Input / Output- 3state)

Multiplexed Address/Data Bus; Lower 8 bits of the memory address (or I/0 address) appear on the bus during the first clock cycle of a machine state. It then becomes the data bus during the second and third clock cycles.

Vss
Ground Reference.

A8 - A15 (Output -3 State)
Address Bus; The most significant 8 bits of the memory address or the 8 bits of the I/0 address,

S0, S1 (Output)
These status signal, similar to IO/M, can identify various operations, but they are rarely used in small systems.

S1	S0	
0	0	HALT
0	1	WRITE
1	0	READ
1	1	FETCH

S1 can be used as an advanced R/W status.

ALE (Output)
Address Latch Enable: it indicates that the bits on AD7-AD0 act as lower 8-bit address bus (A7-A0) when logic high. If ALE=0, it act as data bus (D7-D0).

WR (Output 3state)
WRITE; indicates the data on the Data Bus is to be written into the selected memory or 1/0 location.

RD (Output 3state)
READ; indicates the selected memory or 1/0 device is to be read and that the Data Bus is available for the data transfer.

IO/M (Output)
This is a status signal used to differentiate between I/O and memory operations. When it is high, it indicates an I/O operation; when it is low, it indicates memory operation. This signal is combined with RD and WR to generate I/O and memory control signal.

READY (Input)
If Ready is high during a read or write cycle, it indicates that the memory or peripheral is ready to send or receive data. If Ready is low, the CPU will wait for Ready to go high before completing the read or write cycle.

RESET IN (Input)
When this pin goes low, it sets the Program Counter to zero and resets the MP, Interrupt Enable and HLDA flip-flops. None of the other flags or registers (except the instruction register) are affected. The CPU is held in the reset condition as long as Reset is applied.

CLK OUT (Output)
This signal can be used as the system clock for other devices.

HLDA (Output)
HOLD ACKNOWLEDGE; indicates that the CPU has received the Hold request and acknowledge that request.

HOLD (Input)
This signal indicates that a peripheral such as DMA controller is requesting the use of the address and data buses.

Vcc
+5 volt supply

Instruction

- → Instruction is a command or information given to the MP to perform a given specific task on specified data.
- → Each instruction has two parts: one the task to be performed and the other is data to be operated.
- → The code which specifies what operation to be performed is called operation code or opcode.
- → The data on which operation is performed is called operand.
  E.g. ADD B
  ADD is opcode and B is operand.
- → The complete instruction is combination of opcode and operand.

Addressing Modes

- → Each instruction performs an operation on the specified data. An operand must be specified for an instruction to be executed. The operand may be in the general purpose registers, accumulator or a memory location.
- → The method in which the operand is specified in an instruction is called addressing mode.
- → The various modes used in 8085 microprocessor are:

1. Implied or Implicit or Inherent Addressing

The instructions of this mode do not have operands. For example:

EI (Enable Interrupt),
STC (set the carry flag),
NOP (No operation)

2. Immediate Addressing

This is the simplest way of addressing. When it executes the instruction will operate on immediate hexadecimal number. The operand is present in instruction in this mode. This mode is used to define and use constants of set initial values of variables. The operand may be 8 bit data or 16 bit data. For example:

MVI B, 05H
LXI B, 7A21H (B←7A and C←21)
ADI 72H

3. Register Addressing

Register direct addressing mode means that a register is the source of an operand for an instruction. It is similar to direct addressing. For example:

MOV A, B
ADD B

4. Direct Addressing

This addressing mode is called direct because the effective address of the operand is specified directly in the instruction. Instructions using this mode may contain 2 or 3 bytes, with first byte as the Op-code followed by 1 or 2 bytes of address of data. Loading and storing in memory use 2 bytes of address while IN and OUT have one byte address. For example:

LDA 2035H	(A←M [2035H])
STA 2500H	(M [2500H]←A)
IN 07H	(A←port address 07H)

5. Register Indirect Addressing

The address of the operand is specified by register pair.

LDAX B	(if B=23 and C=50 then A←M [2350H])
STAX D	(if D=30 and E=10 then M [3010H]←A)
MOV A, M	(M=HL; if H=68 and L=32, then A←M [6832H)

Instruction and Data Flow

An instruction is a binary pattern designed to perform a specific function. The list of entire instructions is called the instruction set. The instruction set determines what function the microprocessor can perform.

The following notations are used in the description of the instructions:

R = 8085 8-bit registers (B, C, D, E, H, L)
M=memory register (location) pointed by value HL
Rs=register source
Rd=register destination (B, C, D, E, H, L)
Rp=register pair (BC, DE, HL)
( ) = contents of

The 8085 instruction set can be classified into the following five categories:

1. **Data Transfer (copy) Instructions**

These instructions perform the following six operations:

→ Load 8-bit number in a register.
→ Load 16-bit number in a register pair.
→ Copy from register to register.
→ Copy between register and memory.
→ Copy between I/O and accumulator.
→ Copy between registers and stack memory.

MVI R, 8-bit
MOV Rd, Rs
LXI Rp, 16-bit
OUT 8-bit
IN 8-bit
LDA 16-bit
STA 16-bit
LDAX Rp
STAX Rp
MOV R, M
MOV M, R

2. **Arithmetic Instructions**

The frequently used arithmetic operations are: Add, Subtract, Increment (add 1), Decrement (subtract 1)

ADD R
ADI 8-bit
ADD M
SUB R
SUI 8-bit
SUM M
INR R
INR M
DCR R
DCR M
INX Rp
DCX Rp

3. **Logical and Bit Manipulation Instructions**

These instructions include the following operations: AND, OR, X-OR, Compare, Rotate bits

ANA R
ANI 8-bit
ANA M

ORA R
ORI 8-bit
ORA M
XRA R
XRI 8-bit
XRA M
CMP R
CPI 8-bit

4. Branching Instructions

The following instructions change the program sequence.

JMP 16-bit
JZ 16-bit
JNZ 16-bit
JC 16-bit
JNC 16-bit
CALL 16-bit
RET

5. Miscellaneous Instructions

There are a number of instructions related with data transfer among the register, the stack operation instructions and interrupt operations of 8085 MP which are kept in this group. They are:

PUSH
POP
EI
DI

6. Machine Control Instructions

These instructions affect the operation of the processor.

HLT
NOP

Intel 8086/8088 microprocessor

8086 Microprocessor: Functional Block Diagram

The 8086 is a 16-bit microprocessor. The term 16 bit implies that its arithmetic logic unit, its internal registers, and most of its instructions are intended to work with 16 bit binary data. The 8086 has a 16 bit data bus, so it can read data from or write data to memory and ports either 16 bits or 8 bits at a time. The 8086 has a 20 bit address bus, so it can address any one of 2^{20}, or 1,048,576 memory locations.

8086 CPU is divided into 2 independent functional parts to speed up the processing namely **BIU** (Bus interface unit) & **EU** (execution unit).

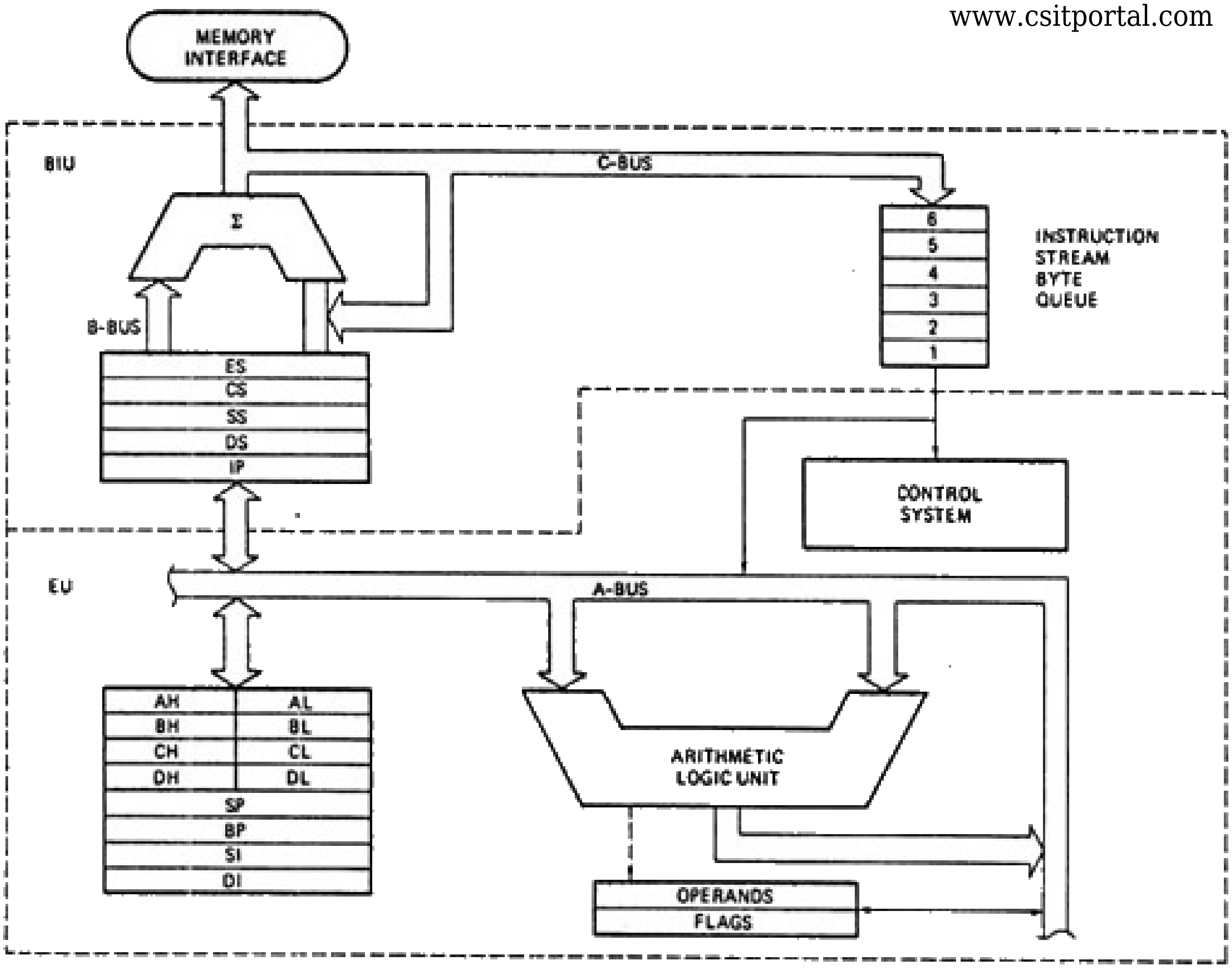


Fig: Functional Block Diagram of Intel 8086 microprocessor

BIU: It handles all transfers of data and addresses on the buses for the execution unit.

- Sends out addresses
- Fetches instructions from memory
- Read / write data from/to ports and memory i.e. handles all transfers of data and addresses on the busses

EU

- Tells BIU where to fetch instructions or data from
- Decodes instructions
- Executes instructions

Execution Unit

Instruction Decoder & ALU:

Decoder in the EU translates instructions fetched from the memory into a series of actions which the EU carries out.16-bit ALU in the EU performs actions such as AND, OR, XOR, increment, decrement etc.

FLAG Register:

It is a 16-bit register. 9-bit are used as different flags, remaining bits unused

				OF	DF	IF	TF	SF	ZF		AF		PF		CF

Fig: 16-bit flag register

Out of 9-flags, 6 are conditional (status) flags and three are control flags

Conditional flags:

These are set or reset by the EU on the basis of the results of some arithmetic or logic operation. 8086 instructions check these flags to determine which of two alternative actions should be done in executing the instructions.

- OF (Overflow flag): is set if there is an arithmetic overflow, i.e. the size of the result exceeds the capacity of the destination location.
- SF (Sign flag): is set if the MSB of the result is 1
- ZF (Zero flag): is set if the result is zero
- AF (Auxiliary carry flag): is set if there is carry from lower nibble to upper nibble or from lower byte to upper byte
- PF (Parity flag): is set if the result has even parity
- CF (Carry flag): is set if there is carry from addition or borrow from subtraction

Control flags:

They are set using certain instructions. They are used to control certain operations of the processor.

- TF (Trap flag): for single stepping through the program
- IF (Interrupt flag): to allow or prohibit the interruption of a program
- DF (Direction flag): Used with string instructions

General purpose Registers (GPRs):

- 8 GPRs AH, AL (Accumulator), BH, BL, CH, CL, DH, DL are used to store 8 bit data.
- AL register is also called the accumulator
- Used individually for the temporary storage of data
- GPRs can be used together (as register pair) to store 16-bit data words. Acceptable register pairs are:

AH-AL pair AX register
BH-BL pair BX register (to store the 16-bit data as well as the base address of the memory location)
CH-CL pair CX register (to store 16-bit data and can be used as counter register for some instructions like loop)
DH-DL pair DX register (to store 16-bit data and also used to hold the result of 16-bit data multiplication and division operation)

Pointer and Index registers: SP (Stack Pointer), BP (Base pointer), SI (Source Index), DI (Destination index)

Pointer Registers:

The two pointer registers, SP and BP are used to access data in the stack segment. The SP is used as offset from current Stack Segment during execution of instruction that involve stack. SP is automatically updated. BP contains offset address and is utilized in based addressing mode. Overall, these are used to hold the offset address of the stack address.

Index Registers:

EU also contains a 16 bit source index (SI) register and 16 bit destination index (DI) register. These registers can be used for temporary storage of data similarly as the general purpose registers. However they are specially to hold the 16-bit offset of the data *word*. SI and DI are used to hold the offset address of the data segment and extra segment memory respectively.

Bus Interface Unit

The QUEUE:

When EU is decoding or executing an instruction, bus will be free at that time. BIU pre-fetches up to 6-instructions bytes to be executed and places them in QUEUE. This improves the overall speed because in each time of execution of new instruction, instead of sending address of next instruction to be executed to the system memory and waiting from the memory to send back the instruction byte, EU just picks up the fetched instruction byte from the QUEUE.

The BIU stores these pre-fetched bytes in a first-in-first-out (FIFO) register set called a queue. Fetching the next instruction while the current instruction executes is called pipelining.

Segment Registers:

The BIU contains a dedicated address, which is used to produce the 20 bit address. The bus control logic of the BIU generates all the bus control signals, such as the READ and WRITE signals, for memory and I/O. The BIU also has four 16 bit segments registers namely:

- Code segment: holds the upper 16-bits of the starting addresses of the segment from which BIU is currently fetching instruction code bytes.
- Stack segment: store addresses and data while subprogram executes
- Extra segment: store upper 16-bits of starting addresses of two memory segments that are used for data.
- Data segment: store upper 16-bits of starting addresses of two memory segments that are used for data.

Code Segment Register (CS) and Instruction Pointer (IP)

All program instructions located in memory are pointed using 16 bits of segment register CS and 16 bits offset contained in the 16 bit instruction pointer (IP). The BIU computes the 20 bit physical address internally using the logical address that is the contents of CS and IP. 16 bit contents of CS will be shifted 4 bits to the left and then adding the 16 bit contents of IP. Thus, all instructions of the program are relative contents of IP. Simply stated, CS contains the base or start of the current code segment, and IP contains the distance or offset from this address to the next instruction byte to be fetched. Graphically,

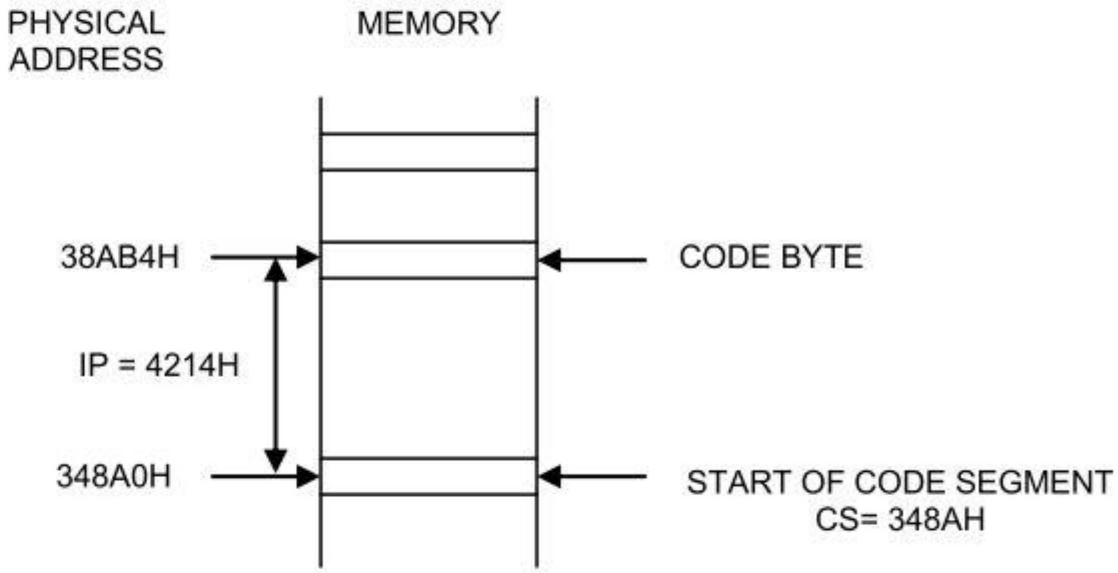


Fig: Diagram showing addition of IP to CS to produce the physical address of code byte Stack Segment Register (SS) and Stack Pointer (SP)

A stack is a section of memory to store addresses and data while a subprogram is in progress. The stack segment registers points to the current stack. The 20 bit physical stack address is calculated from the SS and SP. The programmer can also use Base Pointer (BP) instead of SP for addressing. In this case, the 20 bit physical address is calculated using SS and BP.

Introduction to 8088 microprocessor and its block diagram

(8088 is as same as 8086, however, there are very few differences between them. This note will make you clearer about 8086. Differences are listed at the end.)

Below is a block diagram of the organizational layout of the Intel 8088 processor. It includes two main sections: the Execution Unit (EU) and the Bus Interface Unit (BIU). The EU takes care of the processing including arithmetic and logic. The BIU controls the passing of information between the processor and the devices outside of the processor such as memory, I/O ports, storage devices, etc.

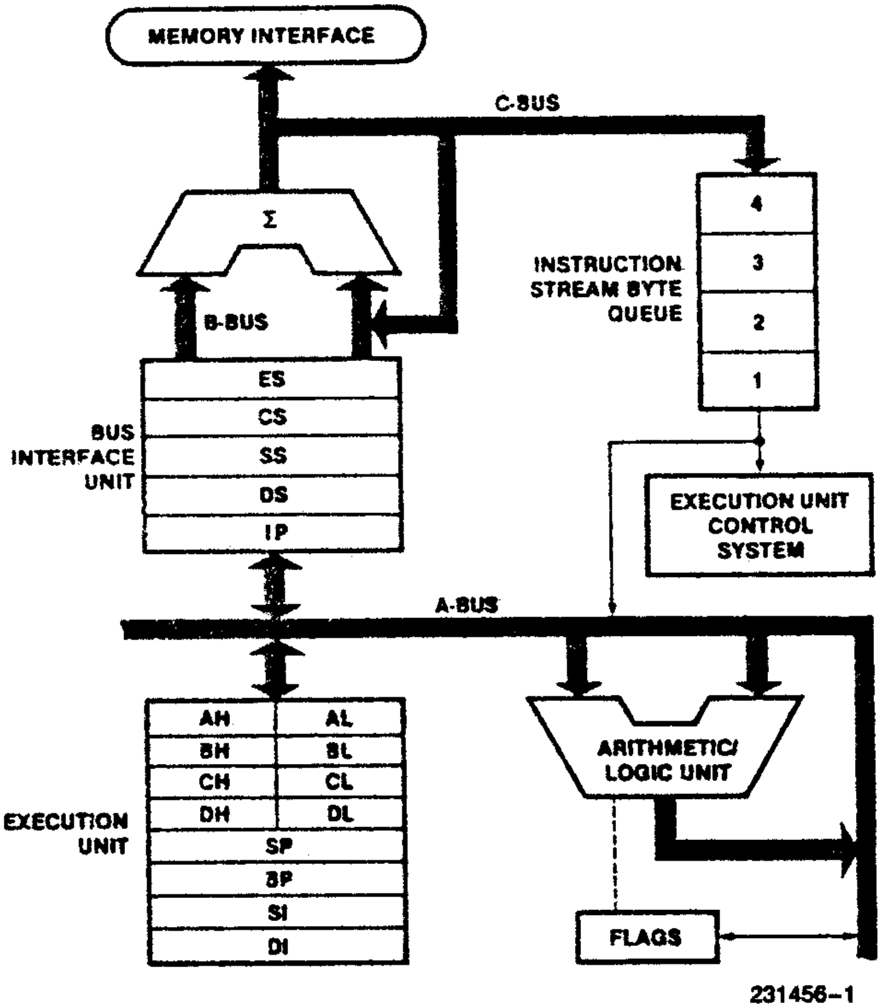


Figure 1. 8088 CPU Functional Block Diagram

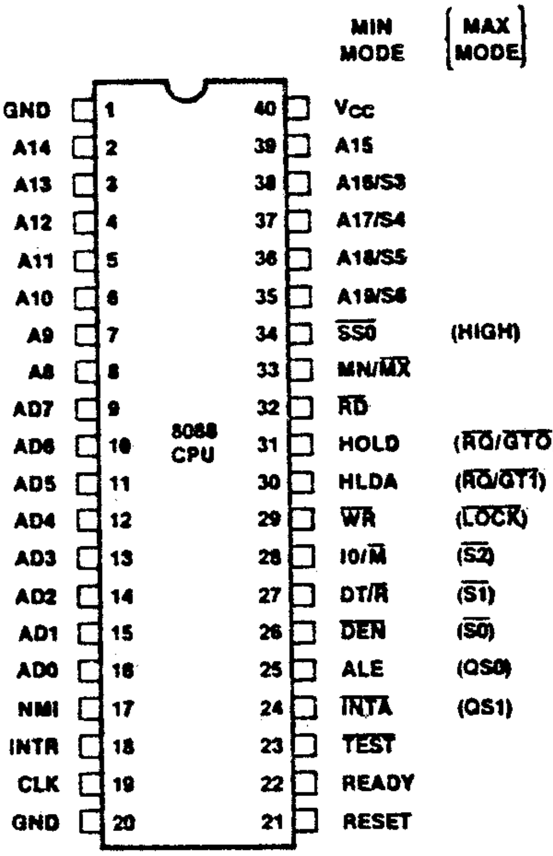


Figure 2. 8088 Pin Configuration

Fig: Functional block diagram of 8088 microprocessor

General Registers

The general registers are categorized into two sets: data and address. The data registers are for calculations; the address registers contain memory addresses and are used to point to the locations in memory where data will be retrieved or stored.

Examining the diagram shows that there are four pairs of registers at the top labeled AH, AL, BH, BL, CH, CL, DH, and DL. These are the data registers. Each of these registers is 8 bits long. Each pair, however, can also operate as a single 16 bit register. AH and AL can operate as a pair referred to as AX. This combining of registers is simply a concatenation, the 8 bits of AL simply tacked to the end of the 8 bits of AH. For example, if AH contains 10110000_2 ($B0_{16}$) and AL contains 01011111_2 ($5F_{16}$), then the virtual register AX contains 1011000001011111_2 ($B05F_{16}$).

Example: If CX contains the binary value $0110\ 1101\ 0110\ 1011_2$, what value does CH have?

Answer: CH contains $0110\ 1101_2$.

Intel has given each of these computational registers a name. These names are listed below:

- AX - Accumulator register
- BX - Base register
- CX - Counter register
- DX - Data register

Below the data registers in the block diagram are the address registers: SP, BP, DI, and SI. These are officially referred to as the pointer (SP and BP) and index registers (DI and SI). These registers are used with the segment registers to point to specific addresses in the memory space of the processor. We will address their operation in the section on the segment registers. It is sufficient at this point to say that they act like pointers in the programming language C or C++. Their basic function is as follows:

- SP is the stack pointer and it points to the "top plate" or last piece of data placed on the stack.
- BP (base pointer), SI (source index), and DI (destination index) are all pointers that the programmer has for their own use.

The Flags

Imagine the instrumentation on the dash board of a car. Blinking on and off occasionally behind the speedometer, tachometer, fuel gauge, and such, are a number of lights informally called "idiot lights". Each of these lights has a unique purpose. One comes on when the fuel is low. Another light up when the high beams are on. Another warns the driver of low coolant. There are many more lights, and depending on the type of car you drive, some may even replace a gauge such as oil pressure.

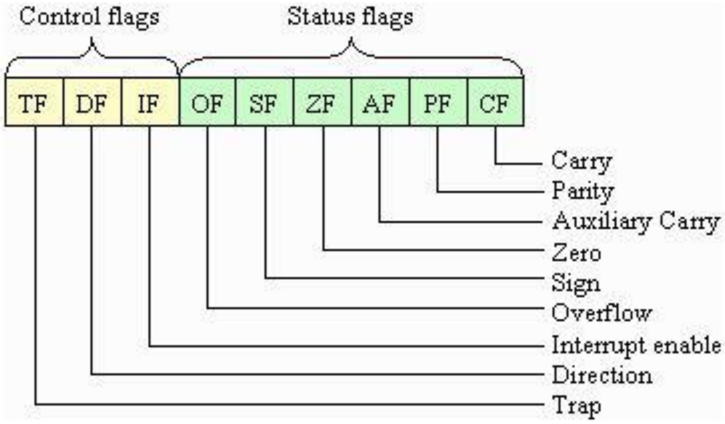
Now let's go back to the processor. There are a number of "idiot lights" that the processor can use, each one based on the result of the previous operation. For example, the addition of two numbers might produce a negative sign, an erroneous overflow, a carry, or a value of zero. Well, that would be four idiot lights: sign, overflow, carry, and zero.

Each of these idiot lights, otherwise known as flags, can be represented with a single bit. If the resulting number had a negative sign, the sign flag would equal 1. If the result was not a negative number, (zero or greater than zero) the sign flag would equal 0. (Side note: Motorola processors more correctly refer to this flag as the negative flag.)

For the sake of organization, these flags are grouped together to form a single number. That number is the ***flags register*** shown at the bottom of the EU section of the processor diagram. The individual bits of the flags are arranged as shown in the figure below:

				OF	DF	IF	TF	SF	ZF		AF		PF		CF

Fig: Flag Register



Example: Assume the flag register is set as shown below after an addition. Using these flags, what can you tell us about the result?

TF	DF	IF	OF	SF	ZF	AF	PF	CF
0	0	0	0	1	0	0	0	1

Answer: As a result of the addition, there was no overflow (OF=0), the result is negative (SF=1), it isn't zero (ZF=0, but you could've also told us that because it is negative), and there was a carry.

Example: If you were to add the binary number 10110101_2 and 10010110_2, how would the flags be set?
Answer: First, let's add the two numbers to see what the result is.

```
  1 0 1 1 0 1 0 1
+ 1 0 0 1 0 1 1 0
1 0 1 0 0 1 0 1 1
```

Now just go from left to right through the status flags.

- OF=1 -- There was an overflow, i.e., adding two negative numbers resulted in a positive number.
- SF=0 -- The result is positive.
- ZF=0 -- The result does not equal zero.
- AF=0 -- For now we won't worry about the auxiliary flag.
- PF=0 -- For now we won't worry about the parity flag.
- CF=1 -- There was a carry.

Arithmetic Logic Unit

As implied by the name, the Arithmetic Logic Unit (ALU) is the computation portion of the EU. Any time arithmetic or logic needs to be performed on numbers, the numbers are sent from the general registers to the ALU, the ALU performs the function, and the result is sent back to the general registers.

EU Control System

The EU Control System is a set of gates that control the timing, passing of data, and other items within the execution unit. It's analogous to a manager in business who doesn't necessarily know the details of the operation, but they plan what happens, where it happens, and when it happens.

Instruction Pointer

All program instructions located in memory are pointed using 16 bits of segment register CS and 16 bits offset contained in the 16 bit instruction pointer (IP). The BIU computes the 20 bit physical address internally using the logical address that is the contents of CS and IP. 16 bit contents of CS will be shifted 4 bits to the left and then adding the 16 bit contents of IP. Thus, all instructions of the program are relative contents of IP. Simply stated, CS contains the base or start of the current code segment, and IP contains the distance or offset from this address to the next instruction byte to be fetched.

Comparison with 8085 microprocessor

Differences between 8085 and 8086

8085	8086
8085 is 8 bit microprocessor	8086 is 16 bit microprocessor
8085 has 16 bit address bus	8086 has 20 bit address bus
Clock speed: 3 MHz	Clock speed can vary between 5, 8 and 10 MHz for three different MP.
8085 can access up to 2^16 = 64 Kb of memory	8086 can access up to 2^20 = 1 MB of memory
8085 doesn't have an instruction queue	8086 has instruction queue
8085 does not support pipelined architecture	8086 supports pipelined architecture.
8085 does not support multiprocessing support	8086 supports
8085 can address 2^8 = 256 I/O's	8086 can access 2^16 = 65,536 I/O's
8085 only supports integer and decimal	8086 supports integer, decimal and ASCII arithmetic.
8085 does not support Multiplication and Division	8086 supports.
8085 supports only single operating mode	8086 operates in two modes.
8085 requires less external hardware	8086 requires more external hardware.
The cost of 8085 is low.	The cost of 8086 is high.
In 8085, memory space is not segmented.	In 8086, memory space is segmented.

Differences between 8086 and 8088

8086	8088
The instruction Queue is 6 byte long.	The instruction Queue is 4 byte long.
In 8086 memory divides into two banks, up to 1,048,576 bytes.	The memory in 8088 does not divide in to two banks as 8086.
The data bus of 8086 is 16-bit wide	The data bus of 8088 is 8-bit wide.
It has BHE (bar) signal on pin no. 34 & there is no SSO (bar) signal.	It does not has BHE (bar) signal on pin no. 34 & has only SSO (bar) signal. It has no S7 pin.
The output signal is used to select memory or I/O at	The output signal is used to select memory or I/O at

M/IO(bar) but if IO(bar)/M low or logic '0' it selects I/O devices and if IO(bar)/M is high or logic '1'it selects memory.	M(bar)/IO but if IO/M(bar) is low or at logic '0',it selects Memory devices and if IO/M(bar) is high or at logic '1'it selects I/O.
It needs one machine cycle to R/W signal if it is at even location otherwise it needs two.	It needs one machine cycle to R/W signal if it is at even location otherwise it needs two.
In 8086, all address & data Buses are multiplexed.	In 8088, address bus; AD_7- AD_0 buses are multiplexed.
It needs two IC 74343 for de-multiplexing AD_0-AD_{19}.	It needs one IC 74343 for de-multiplexing AD_0-AD_7.
Maximum supply current 360mA.	Maximum supply current 340mA.
Three clock speed: 5, 8, 10 MHz	Two clock speed: 5, 8 MHz

Addressing modes of 8086

Implied (or implicit or Inherent) Addressing mode: The data value/data address is implicitly associated with the instruction. Example

STC	set carry flag
CLC	clear (reset) carry flag

Immediate Addressing mode: Operand (one of the operand say source) is immediate data in the instruction. Example
MOV CX, 234AH
ADI 23H

Register Addressing mode: Transfers a copy of byte or word from source register or memory location to destination register or memory location. Example
MOV CX, DX ; transfers the content of DX to CX register pair

Direct Addressing mode: One operand should be memory location i.e. moves byte or word between memory location and the register. Example
MOV BX, [4371H] ; copies 16-bit word into a memory pointed by the displacement address 4371H to the BX register.

Register Indirect Addressing: Transfers a word or byte between register and memory addressed by index or base register. Index or base register are BP, BX, DI, and SI. Example
MOV AX, [CX] ; copies the word-sized data from data segment offset address indexed by BX into AX

Register Relative Addressing Mode: Transfers a byte or word between register and memory location addressed by an index or base register plus displacement. Example
MOV AX, [CX+4]

Base Addressing mode: 8-bit or 16-bit instruction operand is added to the contents of a base register (BX or BP), the resulting value is a pointer to location where data resides. Example
MOV AX, [BP]

Index Addressing mode: 8-bit or 16-bit instruction operand is added to the contents of an index register (SI or DI), the resulting value is a pointer to location where data resides. Example
MOV AX, [DI]

Base-Plus-Index Addressing Mode: Transfers a word or byte between register and memory location addressed by base register (BP or BX) plus index register (DI or SI). Example
MOV [BX+DI], CL Used for locating array data type.

Based-Plus-Index with Displacement Addressing mode: 8-bit or 16-bit instruction operand is added to the contents of a base register (BX or BP) and index register (SI or DI), the resulting value is a pointer to location where data resides. Example

MOV AL, [BX+DI+4]
MOV [BX+DI+4], CL

Bus structure

A microprocessor unit performs basically four operations: memory read, memory write, I/O read, I/O write. These operations are part of communication between MPU & peripheral devices.

A communication includes identifying peripheral or memory location, transfer of data & control functions. These are carried out using address bus, data bus and control bus respectively. All the buses together are called the system bus.

In case of 8085 MPU we have

- 8 unidirectional address pins
- 8 bi directional multiplexed address/ data pins
- 11 control output pins
- 11 control input pins

Data bus:

The data bus provides a path for data flow between the system modules. It consists of a number of separate lines, generally 8, 16, 32 or 64. The number of lines is referred to as width of the data bus. A single line can only carry one bit at a time, the number of lines determine how many bits can be transmitted at a time. The width also determines the overall system performance.

An 8 bit data bus would require twice the time required by 16 bit data bus to transmit 16 bit data

8085 – 8 bit data bus
8086 – 16 bit data bus

Address bus:

The address bus is used to designate the source and destination of data in data bus. In a computer system, each peripheral or memory location is identified by a binary number called an address.

The width of the address bus determines the maximum possible memory capacity of the system. The address bus is also used to address I/O ports. Usually higher order bits are used to select particular modules and lower order bit select a memory location or I/O port within a module.

8085 – 16 bit address bus

Thus, maximum amount of memory locations it can address 2^{16} = 65, 536 or 64 Kb

Control bus:

These are group of lines used to control the data and address bus. Since this bus is shared by all the component of the microcomputer system; there must be some control mechanism to distinguish between data and address. The timing signals indicate the validity of data and address information; while command signal specify operations to be performed. Some of the control signals are:

- Memory write
- Memory Read
- I/O write
- I/O read
- ALE
- Interrupt Request
- Interrupt Acknowledge

Unit 5
Assembly Language Programming

Programming with Intel 8085 microprocessor

Instruction Format

On the basis of size they occupy, the instruction formats are:

i) One byte instruction
It includes the op-code and operand in the same byte (all register transfer). E.g. MOV A, B

ii) Two byte instruction
The first byte specifies op-code and second byte specifies the operand. E.g. MOV A, 32H

iii) Three byte instruction
The first byte specifies op-code and following two byte specifies 16-bit operand. E.g. LXI B, 2032H

Instruction Types

According to their functions, 8085 instructions can be classified as:

1) Data transfer
2) Arithmetic
3) Logical
4) Branching
5) Miscellaneous

1) Data transfer instructions

i) MOV
Syntax: MOV Rd, Rs
MOV M, Rs
MOV Rd, M

ii) MVI
Syntax: MVI Rd/M, 8-bit data

iii) LXI
Syntax: LXI Reg. pair, 16-bit data
e.g. LXI B, 8085

iv) LDA
Syntax: LDA 16-bit address
e.g. LDA 8085
A←M[8085]

v) STA
Syntax: STA 16-bit address
e.g. STA 8085
M [8085]←A

vi) LDAX
Syntax: LDAX B/D register pair
e.g. LDAX B
A←M [8050] (if B=80, C=50)

vii) STAX
Syntax: STAX B/D register pair
e.g. STAX B
M [8050]←A (if B=80, C=50)

viii) LHLD
Syntax: LHLD 16-bit address
e.g. LHLD 8085
L←M [8085]
H←M [8086]

ix) SHLD
Syntax: SHLD 16-bit address
e.g. SHLD 8085
M [8085]←L
M [8086]←H

x) XCHG
Syntax: XCHG ; interchanges between D and H; and E and L
H←D and L←E
D←H and E←L

xi) IN
Syntax: IN 8-bit port address
e.g. IN 15 ; data received from port address 15 is transferred into Accumulator.

xii) OUT
Syntax: OUT 8-bit port address
e.g. OUT 15 ; content of Accumulator is transferred to port address 15.

2) Arithmetic Instructions

i) ADD
Syntax: ADD R/M
e.g. ADD B ; A←A+B

ii) ADC
Syntax: ADC R/M
e.g. ADC B ; A←A+B+CY

iii) ADI
Syntax: ADI 8-bit data
e.g. ADI 45 ; A←A+45

iv) ACI
Syntax: ACI 8-bit data
e.g. ACI 45 ; A←A+45+CY

v) SUB
Syntax: SUB R/M
e.g. SUB B ; A←A-B

vi) SBB
Syntax: SBB R/M
e.g. SBB B ; A←A-B-CY

vii) SUI
Syntax: SUI 8-bit data
e.g. SUI 45 ; A←A-45

viii) SBI
Syntax: SBI 8-bit data
e.g. SBI 45 ; A←A-45-CY

ix) DAD
Syntax: DAD register pair ; contents of register pair is added to HL and the sum is saved in HL pair.

e.g. DAD B ; HL←HL+BC

x) INR

Syntax: INR R/M

e.g. INR B ; B←B-1

xi) DCR

Syntax: DCR R/M

e.g. DCR C ; C←C-1

xii) INX

Syntax: INX register pair

e.g. INX H ; HL←HL+1

xiii) DCX

Syntax: DCX register pair

e.g. DCX B ; BC←BC-1

3) Logical instructions

i) ANA

e.g. ANA reg./M

ii) ANI

e.g. ANI 8-bit data

iii) ORA

e.g. ORA Reg./M

iv) ORI

e.g. ORI 8-bit data

v) XRA

e.g. XRA Reg./M

vi) XRI

e.g. XRI 8-bit data

vii) CMA

e.g. CMA ; complements the content of accumulator

viii) CMP

Syntax: CMP Reg./M

The contents of the operand (register/memory) are compared with the content of accumulator and both contents are preserved and the comparison is shown by setting the flags.

If A>Reg./M, CY=0, Z=0

If A=Reg./M, CY=0, Z=1

If A<Reg./M, CY=1, Z=0

ix) CPI

Syntax: CPI 8-bit data

If A>data, CY=0, Z=0

If A=data, CY=0, Z=1

If A<data, CY=1, Z=0

x) RLC

Each binary bit of the accumulator is rotated left by one position. Bit D7 is placed in the position of D0 as well as in the carry flag (CY). Other flags are not affected.

Syntax: RLC

Before RLC: 10100111, CY=0

After RLC: 01001111; CY=1

xi) RRC

Syntax: RRC

Before RLC: 10100111, CY=0

After RLC: 11010011; CY=1

xii) RAL

Each binary bit of the accumulator is rotated left by one position through the carry flag (CY). Other flags are not affected.

Syntax: RAL

Before RAL: 10100111, CY=0

After RAL: 01001110; CY=1

xiii) RAR

Each binary bit of the accumulator is rotated right by one position through the carry flag. Other flags are not affected.

Syntax: RAL

Before RAL: 10100111, CY=0

After RAL: 01010011; CY=1

xiv) DAA

Decimal Adjust Accumulator

The contents of accumulator are changed from binary value to two binary-coded decimal (BCD) digits. This is the only instruction that uses the AC flag to perform binary to BCD conversion. All flags are affected.

4) Branching instructions

a) Jump instructions

i) Unconditional jump

The program sequence is transferred to the memory location specified by the 16-bit address without checking any condition.

Syntax: JMP 16-bit address
e.g. JMP 8085

ii) Conditional jump

Op-code	**Description**	**Flag status**
JC	Jump on carry	CY=1
JNC	Jump on no carry	CY=0
JP	Jump on positive	S=0
JM	Jump on minus	S=1
JPE	Jump on even parity	P=1
JPO	Jump on odd parity	P=0
JZ	Jump on zero	Z=1
JNZ	Jump on non-zero	Z=0

b) Call instructions / Return instructions

i) Unconditional call/return

```
Syntax:     CALL 16-bit address
            :
            :
            :
            RET
```

ii) Conditional Call/return

Op-code	**Description**	**Flag status**
CC/RC	Call/Return on carry	CY=1

CNC/RNC	Call/Return on no carry	CY=0
CP/RP	Call/Return on positive	S=0
CM/RM	Call/Return on minus	S=1
CPE/RPE	Call/Return on even parity	P=1
CPO/RPO	Call/Return on odd parity	P=0
CZ/RZ	Call/Return on zero	Z=1
CNZ/RNZ	Call/Return on non-zero	Z=0

c) Restart instructions

The RST instructions are equivalent to 1-byte call instructions to one of the eight memory locations on page 0. The instructions are generally used in conjunction with interrupts and inserted using external hardware. However, these can be used as software instructions in a program to transfer program execution to one of the eight locations.

Op-code	**Restart Address (H)**
RST 0	0000
RST 1	0008
RST 2	0010
RST 3	0018
RST 4	0020
RST 5	0028
RST 6	0030
RST 7	0038

5) Miscellaneous instructions

i) PUSH

Syntax: PUSH reg. pair

e.g. PUSH B

Before instruction	After instruction
B=32, C=57	M [2097]=57
SP=2099	M [2098]=32
	SP=2097

ii) POP

Syntax: POP reg. pair

e.g. PUSH H

Before instruction	After instruction
SP=2090	H=01
M [2090]=F5	L=F5
M [2091]=01	SP=2092

iii) EI

Enabling interrupts

iv) DI

Disabling interrupts

v) PCHL

The contents of registers H and L are copied to the program counter.

vi) SPHL

The contents of registers H and L are copied to the stack pointer register.

vii) XTHL

Exchange H and L with Top of Stack

6) Machine control instructions

i) HLT

Halt and enter wait state. The contents of the registers are unaffected during the HLT state.

ii) NOP

No operation is performed. The instruction is fetched and decoded; however, no operation is executed. The instruction is used to fill in time delays or to delete and insert instructions while troubleshooting.

iii) RIM

Read Interrupt Mask. This is a multipurpose instruction used to read the status of interrupts 7.5, 6.5, 5.5 and to read serial data input bit. The instruction loads 8 bits in the accumulator with the following interpretations:

D7	D6	D5	D4	D3	D2	D1	D0
SID	I7	I6	I5	IE	7.5	6.5	5.5

D7=serial input data bit

D6, D5, D4=interrupts pending if bit=1

D3=interrupt enable; flip-flop is set if bit=1

D2, D1, D0=interrupt masked if bit=1

e.g. After the execution of instruction RIM, the accumulator contained 49H. Explain

(A): 49H = 0 1 0 0 1 0 0 1

RST 7.5 is pending.

Interrupt enable flip-flop is set.

RST 7.5 and 6.5 are enabled. RST 5.5 masked.

iv) SIM

Set Interrupt Mask. This is a multipurpose instruction and used to implement the 8085 interrupts (RST 7.5, 6.5, and 5.5) and serial data output.

The instruction interprets the accumulator contents as follows:

D7	D6	D5	D4	D3	D2	D1	D0
SOD	SDE	xxx	R7.5	MSE	M7.5	M6.5	M5.5

D7=serial output data

D6=serial data enable (1=enable and 0=disable)

D4=if 1, reset RST 7.5 flip-flop

D3=if 1, mask set enable

D2, D1, D0=masks interrupts if bits=1

- **SOD** – Serial Output Data: Bit D7 of the accumulator is latched into the SOD output line and made available to a serial peripheral if bit D6=1.
- SDE – Serial Data Enable: If this bit = 1, it enables the serial output. To implement serial output, this bit needs to be enabled.
- **XXX** – Don't care.
- **R7.5** – Reset RST 7.5: If this bit = 1, RST 7.5 flip-flop is reset. This is an additional control to reset RST 7.5.
- **MSE** – Mask Set Enable: If this bit is high, it enables the functions of bits D2, D1, D0. This is a master control over all the interrupt masking bits. If this bit is low, bits D2, D1, and D0 do not have any effect on the masks.
- **M7.5** – D2 = 0, RST 7.5 is enabled.
  = 1, RST 7.5 is masked or disabled.
- **M6.5** – D1 = 0, RST 6.5 is enabled.
  = 1, RST 6.5 is masked or disabled.
- **M5.5** – D0 = 0, RST 5.5 is enabled.
  = 1, RST 5.5 is masked or disabled.

Simple Sequence Programs, Branching, Looping

- Explained inside the class and lab.

Programming with Intel 8086 microprocessor

Assembly Instruction Format: Op-codes, mnemonics and operands

Machine Language:

There is only one programming language that any computer can actually understand and execute: its own native binary machine code. This is the lowest possible level of language in which it is possible to write a computer program. However, binary code is not native to humans and it is very easy for error to occur in the program. These bugs are not easy to determine in a pool of binary digits. Also for CPU like 8086 it is tedious and virtually impossible to memorize thousands of binary instructions codes.

Example

Program memory address	Content (binary)	Content (Hex)
00100H	0000 0100	04h

Assembly Language:

Programs in assembly language are represented by certain words representing the operation of instruction. Thus programming gets easier. Assembly language statements are generally written in a standard form that has four fields.

- Label field
- Op-code field (Instruction or Mnemonic)
- Operand field
- Comment field

Let us consider a simple example to add two numbers

```
Label  Mnemonic  Operand   Comment
Start: MVI       A, 10h    ; Move 10h into accumulator
       MVI       B, 20h    ; Move 20h into register B
       ADD       B         ; Add the contents of register B with accumulator
```

Thus, we see the ease with the assembly language rather than machine language.

Mnemonic is a short alphabetic code used in assembly language for microprocessor operation. These are words (usually two-to-four letter) used to represent each instruction.

Assembler is software (program module) which converts assembly language code (source module) into a machine language code.

Types of Assemblers

One Pass Assembler:

- Goes (scans) through the program once
- Can resolve backward reference
- Cannot resolve forward reference

Two Pass Assembler:

- Goes (scans) through the assembly language program twice
- First pass generates the table of the symbol, which consists of labels with addresses assigned to them
- Second pass uses the symbol table generated in the first pass to complete the object code for each instruction thus producing machine code
- It can resolve forward and backward reference both.

Backward reference	Forward reference
L1: ….. ….. ….. ….. JMP L1	JMP L1 ….. ….. ….. L1: …..

Linker is a program that links object file created by assembler into a single executable file.

However, it must be remembered that for execution these codes are converted to machine codes. This is done by assembler. An assembler translates a program written in assembly language into machine language program (object code). Assembly language program are called “source codes”. Machine language programs are known as object codes. A translator converts source codes to object codes and then into executable formats. The process of converting source code into object code is called compilation and assembler does it. The process of converting object codes into executable formats is called linking and ***linker*** does it.

*.asm ~~assembler~~ *.obj ~~linker~~ *.exe

Macro assembler

Macro assembler is an assembly language that allows macros to be defined and used. The Microsoft Macro Assembler (MASM) is an x86 assembler that uses the Intel syntax for MS-DOS and Microsoft Windows.
It translates a program written in macro language into the machine language. A macro language is the one in which all the instruction sequence can be defined in macro block. A macro is an instruction sequence that appears repeatedly in the program assigned with specific name. The MASM replaces a macro name with appropriate instruction sequence whenever it encounters a macro name. E.g.

```
Initiz macro
        Mov ax, @dataseg
        Mov ds, ax
        Mov es, ax
Endm
```

```
OR,     Initiz { Mov ax, @dataseg
                 Mov ds, ax
                 Mov es, ax }
```

There exists little difference between macro program and subroutine program.
When a subroutine program occurs in program, execution jumps out of main program and executes subroutine and control returns to the main program after RET instruction. A macro, in the other hand, does not cause program execution to branch out of main program. Each time a macro occurs, it is replaced with appropriate sequence in the main program.

Advantages of using Macro

- To simplify and reduce the repetitive coding.
- To reduce errors caused by repetitive coding.
- To make assembly language program more readable.

Assembler Directives

The instruction that will give the information to assembler for performing assembling are called assembler directives. They are not executed by MP, so they are called dummy or pseudo instructions. It gives direction to the assembler.

Following are some commonly used directives:

1) **Segment and Ends directive**

```
Segment_name SEGMENT
…………
…………
…………
Segment_name ENDS

e.g.    dataseg segment
        a dw 1234h
        b dw 4321h
        c dw ?
        dataseg ends
```

2) **Proc and Endp directive**

```
Procedure_name proc
……...
………
………
Procedure_name endp
```

3) **END directive**
    - Ends the entire program and appears as the last statement.

    END procedure_name

4) **ASSUME directive**

    Assume CS: CODE_HERE, DS: DATA_HERE

5) **PAGE directive**
    - Specifies the maximum number of lines the assembler is to list on a page and the maximum number of characters on a line.

    PAGE [length], [width]   ; maximum is page 10 to 255, 60 to 132
    e.g. page 60, 132
    - length is 60 lines per page and width is 132 characters per line.

6) **TITLE directive**

    Title text

7) **EQU directive**
    - Is used to assign name to constant used in your program.

    pi equ 3.1417 ; pi=3.1417

8) **DUP directive**

    Price dw 4 dup(0)
    Weight db 100 dup(?)

9) **Data Definition directive**

Directive	**Description**	**Size (byte)**	**Attribute**
DB	Define byte	1	Byte
DW	Define word	2	Word
DD	Define double word	4	Double word
DQ	Define quad word	8	Quad word
DT	Define ten byte	10	Ten byte

10) **ORG directive**

It is used to assign the starting memory location to the program.
Org 3000H

11) **Macro and Endm directive**

- Used to define macro module.

12) **OFFSET directive**

It is an operator which informs the assembler to determine the displacement of named data or variable from the start of the segment, which contains it.

Simplified Segment Directives

.model (memory model)

Memory model	**Description**
Tiny	Code and data together may not be greater than 64K
Small	Neither code nor data may be greater than 64K
Medium	Only the code may be greater than 64K
Compact	Only the data may be greater than 64K
Large	Both code and data may be greater than 64K
Huge	All available memory may be used for code and data

.stack

The .stack directive sets the size of the program stack, which may be any size up to 64K. This segment is addressed by SS and SP registers.

.code

The .code directive identifies the part of the program that contains instructions. This segment is addressed by CS and IP registers.

.data

All variables are defined in this segment area.

Instruction sets

- Provided in the photocopy. Study the following.

1) Data transfer: MOV, IN, OUT, LEA

i) MOV R/M, R/M/Imm
Copy byte or word from specified source to specified destination

ii) IN AL/AX, port no/DX
Copy a byte or word from specified port number to accumulator
Second operand is a port number. If required to access port number over 255 - **DX** register should be used.

iii) OUT port no/DX, AL/AX

Copy a byte or word from accumulator to specified port number

First operand is a port number. If required to access port number over 255 - **DX** register should be used.

iv) LEA R, M

Load effective address of operand into specified register

2) Arithmetic: ADD, SUB, INC, DEC, MUL, DIV, CMP, DAA, AAA

i) ADD R/M, R/M/Imm

Add specified byte to byte or specified word to word

ii) SUB R/M, R/M/Imm

Subtract specified byte from byte or specified word from word

iii) INC R/M

Increment specified byte or word by 1

iv) DEC R/M

Decrement 1 from specified byte or word

v) MUL R/M

Multiplies an unsigned multiplicand by an unsigned multiplier

AL or AX is assumed as multiplicand.

vi) DIV R/M

Divides an unsigned dividend (accumulator) by an unsigned divisor (register)

vii) CMP R/M, R/M/Imm

Compare two specified bytes or two specified words

	CF	SF	ZF
Operand1>Operand2	0	0	0
Operand1=Operand2	0	0	1
Operand1<Operand2	1	1	0

viii) DAA

Decimal adjust After Addition.

Corrects the result of addition of two packed BCD values

Algorithm:

if low nibble of AL > 9 or AF = 1 then:

- AL = AL + 6
- AF = 1

If AL > 9Fh or CF = 1 then:

- AL = AL + 60h
- CF = 1

Example:

```
MOV AL, 0Fh   ; AL = 0Fh (15)
DAA           ; AL = 15h
RET
```

ix) AAA

ASCII Adjust after Addition.

Corrects result in AH and AL after addition when working with BCD values.

It works according to the following Algorithm:

if low nibble of AL > 9 or AF = 1 then:

- AL = AL + 6
- AH = AH + 1
- AF = 1
- CF = 1

Else

- AF = 0
- CF = 0

in both cases:
clear the high nibble of AL.

Example:

```
      MOV AX, 15   ; AH = 00, AL = 0Fh
      AAA          ; AH = 01, AL = 05
      RET
```

3) Logic: AND, OR, XOR, NOT, ROR, RCR, ROL, RCL, SHL, SHR

i) AND/OR/XOR R/M, R/M/Imm

ii) NOT R/M
Invert each bit of a byte or word

iii) RCL/RCR
Syntax: RCL/RCR R/M, CL/Imm

RCL	**RCR**
Rotate operand1 left through Carry Flag. The number of rotates is set by operand2.	Rotate operand1 right through Carry Flag. The number of rotates is set by operand2.
Algorithm: shift all bits left, the bit that goes off is set to CF and previous value of CF is inserted to the right-most position.	Algorithm: shift all bits right, the bit that goes off is set to CF and previous value of CF is inserted to the left-most position.
Example: STC ; set carry (CF=1). MOV AL, 1Ch ; AL = 00011100b RCL AL, 1 ; AL = 00111001b, CF=0. RET	Example: STC ; set carry (CF=1). MOV AL, 1Ch ; AL = 00011100b RCR AL, 1 ; AL = 10001110b, CF=0. RET

iv) ROL/ROR
Syntax: ROL/ROR R/M, CL/Imm

ROL	**ROR**
Rotate operand1 left. The number of rotates is set by operand2.	Rotate operand1 right. The number of rotates is set by operand2.
Algorithm: shift all bits left, the bit that goes off is set to CF and the same bit is inserted to the right-most position.	Algorithm: shift all bits right, the bit that goes off is set to CF and the same bit is inserted to the left-most position.
Example: MOV AL, 1Ch ; AL = 00011100b ROL AL, 1 ; AL = 00111000b, CF=0. RET	Example: MOV AL, 1Ch ; AL = 00011100b ROR AL, 1 ; AL = 00001110b, CF=0. RET

v) SHL/SHR
Shift logical left / Shift logical right
Syntax: SHL/SHR R/M, CL/Imm

SHL	**SHR**
Shift operand1 Left. The number of shifts is set by operand2.	Shift operand1 Right. The number of shifts is set by operand2.
Algorithm: • Shift all bits left, the bit that goes off is set to CF. • Zero bit is inserted to the right-most position.	Algorithm: • Shift all bits right, the bit that goes off is set to CF. • Zero bit is inserted to the left-most position.
Example: MOV AL, 11100000b SHL AL, 1 ; AL = 11000000b, CF=1.	Example: MOV AL, 00000111b SHR AL, 1 ; AL = 00000011b, CF=1.

4) Branching: JMP, CALL, RET, LOOP
   i) JMP label
      Jumps to a designated address
   ii) CALL label
      Call a procedure (subprogram), save return address on stack
   iii) RET
      Returns from a procedure previously entered by a call
   iv) LOOP
      Loop through a sequence of instruction until CX=0

```
Label: …..
       …..
       …..
       Loop Label
```

5) Stack: PUSH, POP
   - Like in 8085
   - E.g. PUSH BX
   - POP BX

INT 21h functions

- Provided in the photocopy
- DOS services
- 01h, 02h, 09h, 0Ah, 4Ch

Function number	**Description**
01h e.g. mov ah,01h int 21h	**Keyboard input with echo:** This operation accepts a character from the keyboard buffer. If none is present, waits for keyboard entry. It returns the character in AL.
02h e.g. mov ah,02h int 21h	**Display character:** Send the character in DL to the standard output device console.
09h e.g. mov ah,09h int 21h	**String output:** Send a string of characters to the standard output. DX contains the offset address of string. The string must be terminated with a '$' sign.
0Ah	**String input**
4Ch e.g. mov ax,4C00h int 21h	**Terminate the current program** (mov ah,4Ch int 21h is also used.)

INT 10h functions

- Provided in the photocopy
- Video display services (BIOS services)
- 00h, 01h, 02h, 06h, 07h, 08h, 09h, 0Ah

Function number	**Description**
00h	Set video mode
01h	Set cursor size
02h	Set cursor position
06h	Scroll window up
07h	Scroll window down
08h	Read character and attribute of cursor

09h	Display character and attribute at cursor
0Ah	Display character at cursor

Simple sequence programs, Branching, Looping

- Discussed in the class and lab.

Unit 5
Computer Animation

Introduction

Although we tend to think of **animation** as implying object motions, the term computer animation generally refers to any time sequence of visual changes in a scene. In addition to changing object position with translations or rotations, a computer-generated animation could display time variations in object size, color, transparency, or surface texture.

Some typical applications of computer-generated animation are entertainment (motion pictures and cartoons), advertising, scientific and engineering studies, and training and education. Advertising animations often transition one object shape into another: for example, transforming a can of motor oil into an automobile engine.

Computer animations can also be generated by changing camera parameters, such as position, orientation, and focal length. And we can produce computer animations by charging lighting effects or other parameters and procedures associated with illumination and rendering.

Design of animation sequences

In general, an animation sequence is designed with the following steps:

1. Storyboard layout
2. Object definitions
3. Key-frame specifications
4. Generation of in-between frames

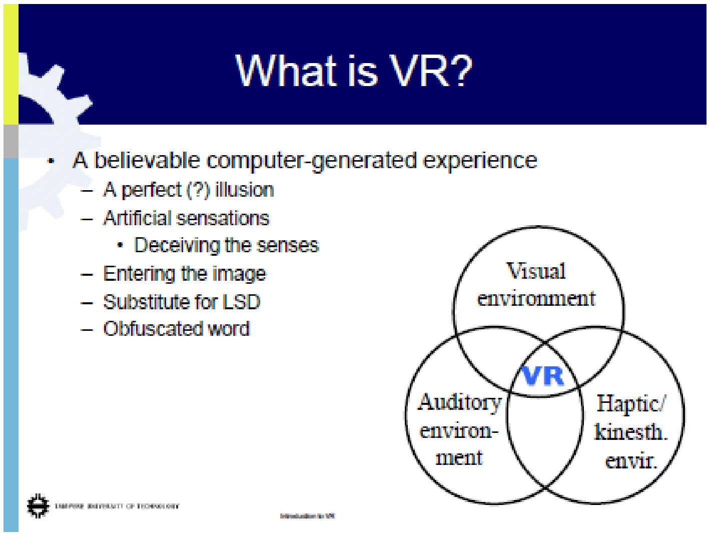
This standard approach for animated cartoons is applied to other animation applications as well, although there are many special applications that do not follow this sequence. Real-time computer animations produced by flight simulators, for instance, display motion sequences in response to settings on the aircraft controls. For frame-by-frame animation, each frame of the scene is separately generated and stored. Later, the frames can be recorded on film or they can be consecutively displayed in "real-time playback" mode.

1. **Storyboard** is an outline of the action. It defines the motion sequence as a set of basic events that are to take place. Depending on the type of animation to be produced, the storyboard could consist of a set of rough sketches or it could be a list of the basic ideas for the motion.
2. An **object definition** is given for each participant in the action. Objects can be defined in terms of basic shapes, such as polygons or splines. In addition, the associated movements for each object are specified along with the shape.
3. A **key frame** is a detailed drawing of the scene at a certain time in the animation sequence. Within each key frame, each object is positioned according to the time for that frame. Some key frames are chosen at extreme positions in the action; others are spaced so that the time interval between key frames is not too great. More key frames are specified for intricate motions than for simple, slowly varying motions.
4. **In-betweens** are the intermediate frames between the key frames. The number of in-betweens needed is determined by the media to be used to display the animation. Film requires 24 frames per second, and graphics terminals are refreshed at the rate of 30 to 60 frames per second.

Typically, time intervals for the motion are set up so that there are from three to five in-betweens for each pair of key frames. Depending on the speed specified for the motion, some key frames can be duplicated. For a 1-minute film sequence with no duplication, we would need 1440 frames. With five in-betweens for each pair of key frames, we would need 288 key frames. If the motion is not too complicated, we could space the key frames a little farther apart.

There are several other tasks that may be required, depending on the application. They include motion verification, editing, and production and synchronization of a soundtrack. Many of the functions needed to produce general animations are now computer-generated.

Virtual Reality

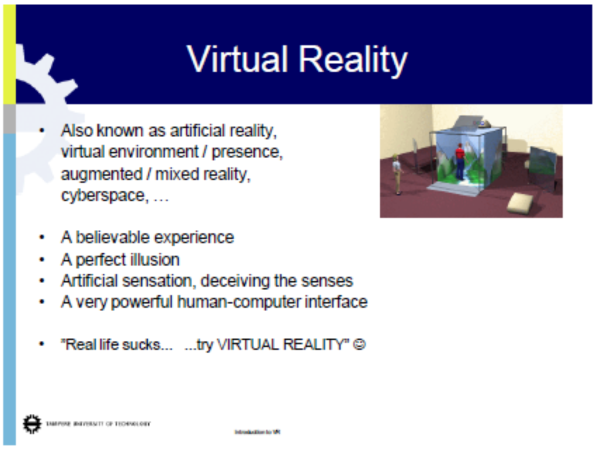


What is VR?

- A believable computer-generated experience
  - A perfect (?) illusion
  - Artificial sensations
    - Deceiving the senses
  - Entering the image
  - Substitute for LSD
  - Obfuscated word

TAMPERE UNIVERSITY OF TECHNOLOGY

Introduction to VR

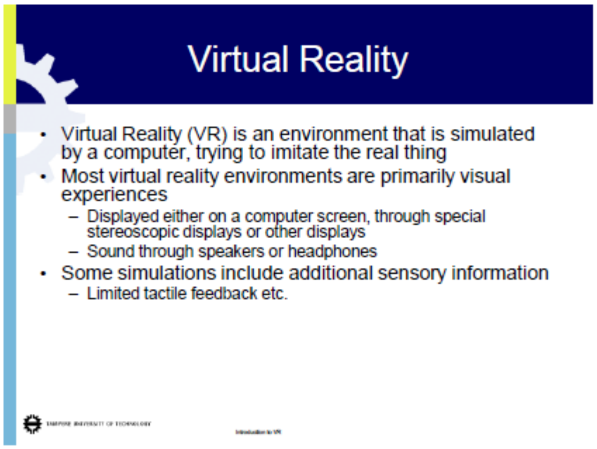


Virtual Reality

- Also known as artificial reality, virtual environment / presence, augmented / mixed reality, cyberspace, …
- A believable experience
- A perfect illusion
- Artificial sensation, deceiving the senses
- A very powerful human-computer interface
- "Real life sucks… …try VIRTUAL REALITY" ☺

TAMPERE UNIVERSITY OF TECHNOLOGY

Introduction to VR

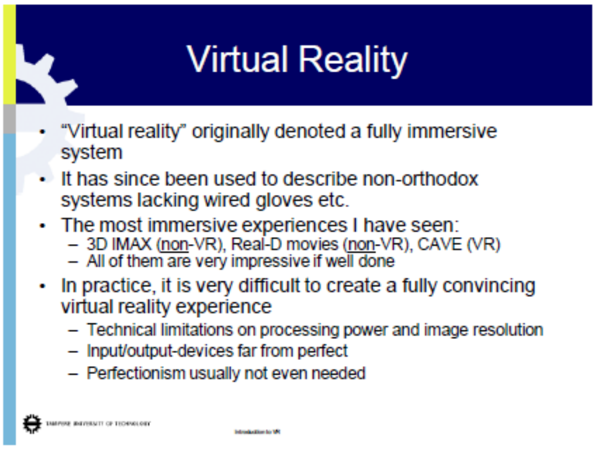


Virtual Reality

- Virtual Reality (VR) is an environment that is simulated by a computer, trying to imitate the real thing
- Most virtual reality environments are primarily visual experiences
  - Displayed either on a computer screen, through special stereoscopic displays or other displays
  - Sound through speakers or headphones
- Some simulations include additional sensory information
  - Limited tactile feedback etc.

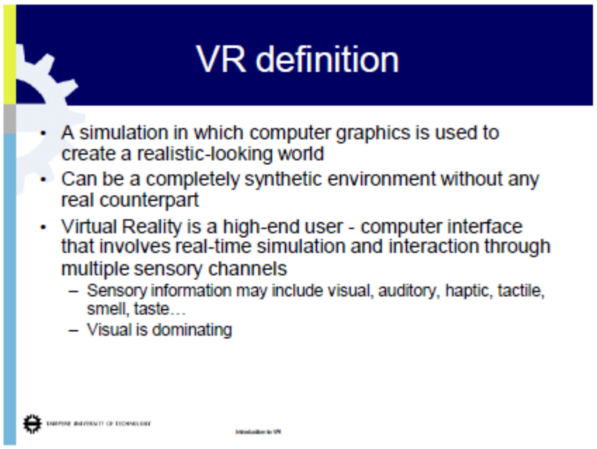
TAMPERE UNIVERSITY OF TECHNOLOGY

Introduction to VR



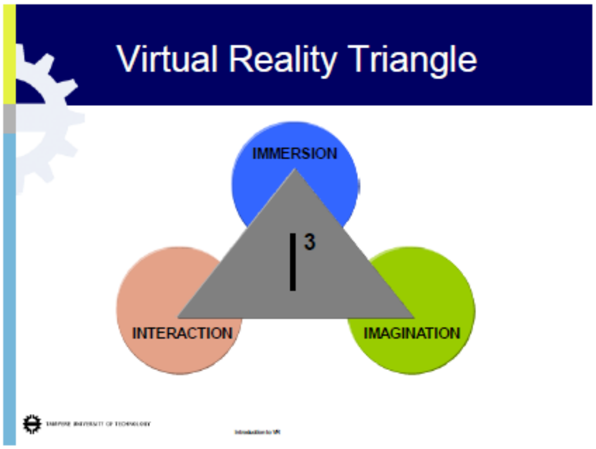
Virtual Reality

- "Virtual reality" originally denoted a fully immersive system
- It has since been used to describe non-orthodox systems lacking wired gloves etc.
- The most immersive experiences I have seen:
  - 3D IMAX (non-VR), Real-D movies (non-VR), CAVE (VR)
  - All of them are very impressive if well done
- In practice, it is very difficult to create a fully convincing virtual reality experience
  - Technical limitations on processing power and image resolution
  - Input/output-devices far from perfect
  - Perfectionism usually not even needed

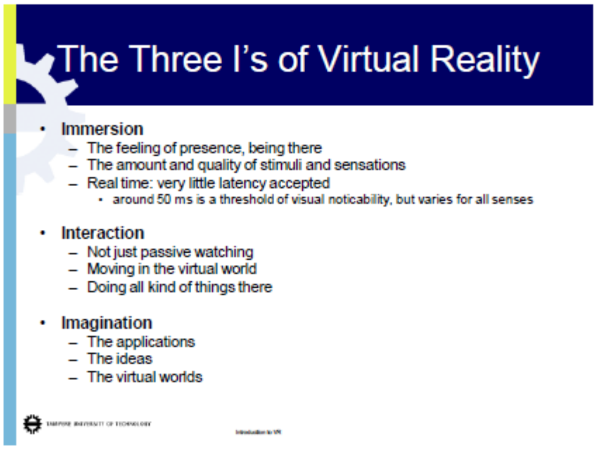


VR definition

- A simulation in which computer graphics is used to create a realistic-looking world
- Can be a completely synthetic environment without any real counterpart
- Virtual Reality is a high-end user - computer interface that involves real-time simulation and interaction through multiple sensory channels
  - Sensory information may include visual, auditory, haptic, tactile, smell, taste…
  - Visual is dominating

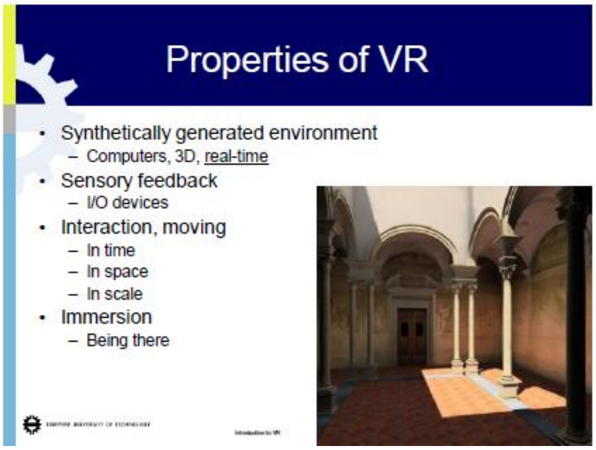


Virtual Reality Triangle



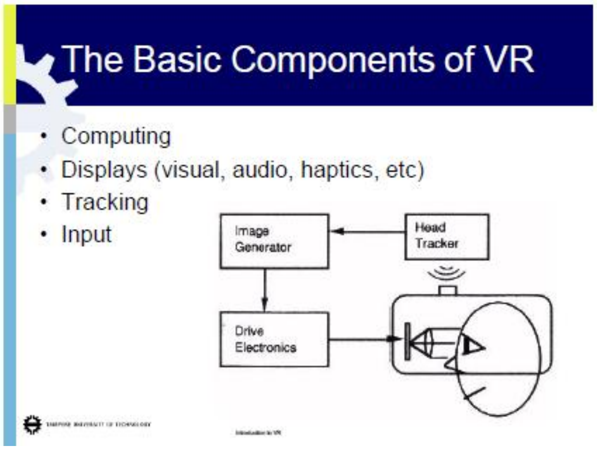
The Three I's of Virtual Reality

- **Immersion**
  - The feeling of presence, being there
  - The amount and quality of stimuli and sensations
  - Real time: very little latency accepted
    - around 50 ms is a threshold of visual noticability, but varies for all senses
- **Interaction**
  - Not just passive watching
  - Moving in the virtual world
  - Doing all kind of things there
- **Imagination**
  - The applications
  - The ideas
  - The virtual worlds



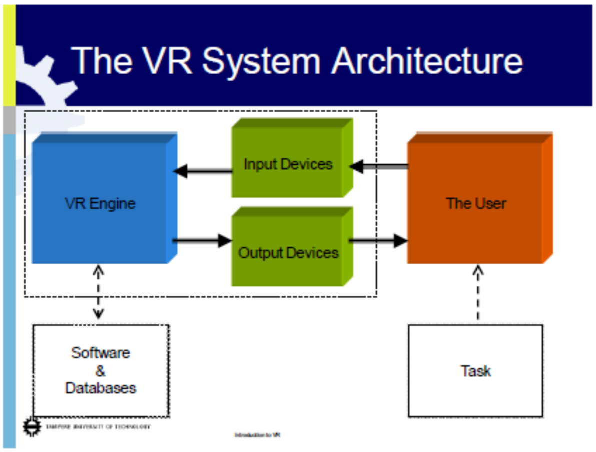
Properties of VR

- Synthetically generated environment
  - Computers, 3D, real-time
- Sensory feedback
  - I/O devices
- Interaction, moving
  - In time
  - In space
  - In scale
- Immersion
  - Being there



The Basic Components of VR

- Computing
- Displays (visual, audio, haptics, etc)
- Tracking
- Input



The VR System Architecture

VR Engine

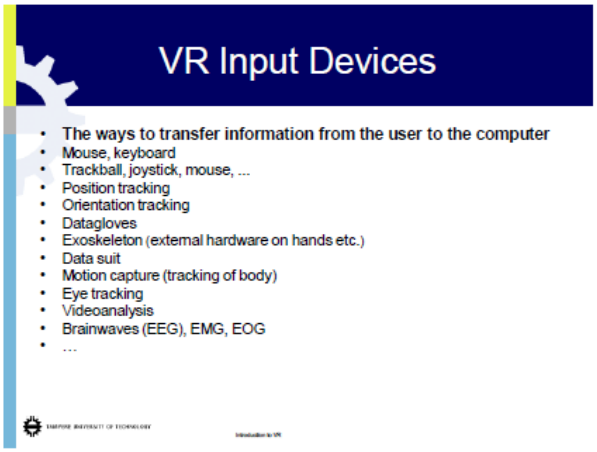
Input Devices

Output Devices

The User

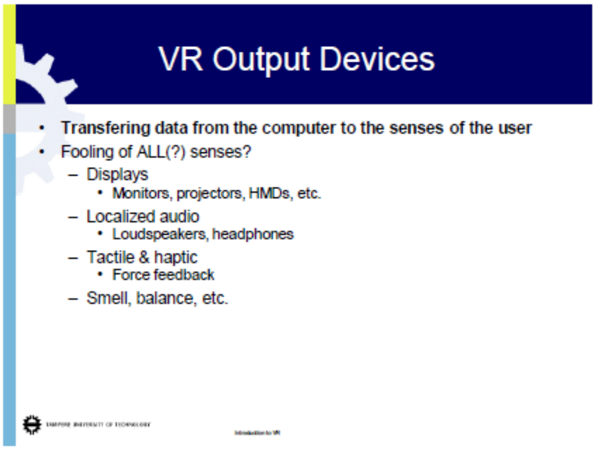
Software
&
Databases

Task



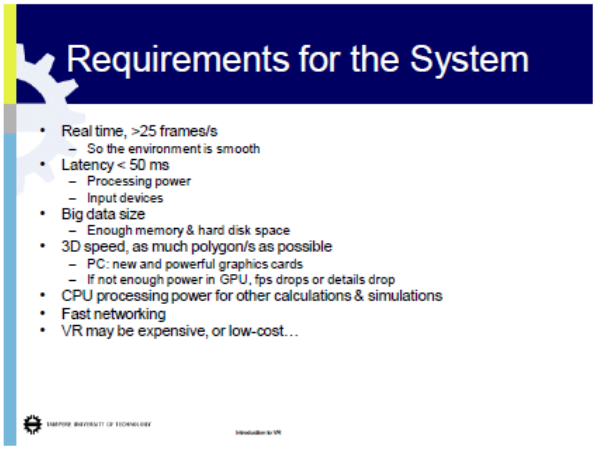
VR Input Devices

- **The ways to transfer information from the user to the computer**
- Mouse, keyboard
- Trackball, joystick, mouse, ...
- Position tracking
- Orientation tracking
- Datagloves
- Exoskeleton (external hardware on hands etc.)
- Data suit
- Motion capture (tracking of body)
- Eye tracking
- Videoanalysis
- Brainwaves (EEG), EMG, EOG
- ...



VR Output Devices

- **Transfering data from the computer to the senses of the user**
- Fooling of ALL(?) senses?
  - Displays
    - Monitors, projectors, HMDs, etc.
  - Localized audio
    - Loudspeakers, headphones
  - Tactile & haptic
    - Force feedback
  - Smell, balance, etc.



Requirements for the System

- Real time, >25 frames/s
  - So the environment is smooth
- Latency < 50 ms
  - Processing power
  - Input devices
- Big data size
  - Enough memory & hard disk space
- 3D speed, as much polygon/s as possible
  - PC: new and powerful graphics cards
  - If not enough power in GPU, fps drops or details drop
- CPU processing power for other calculations & simulations
- Fast networking
- VR may be expensive, or low-cost…

Unit 6
Basic I/O, Memory R/W and Interrupt Operations

Memory Read/Write Operation

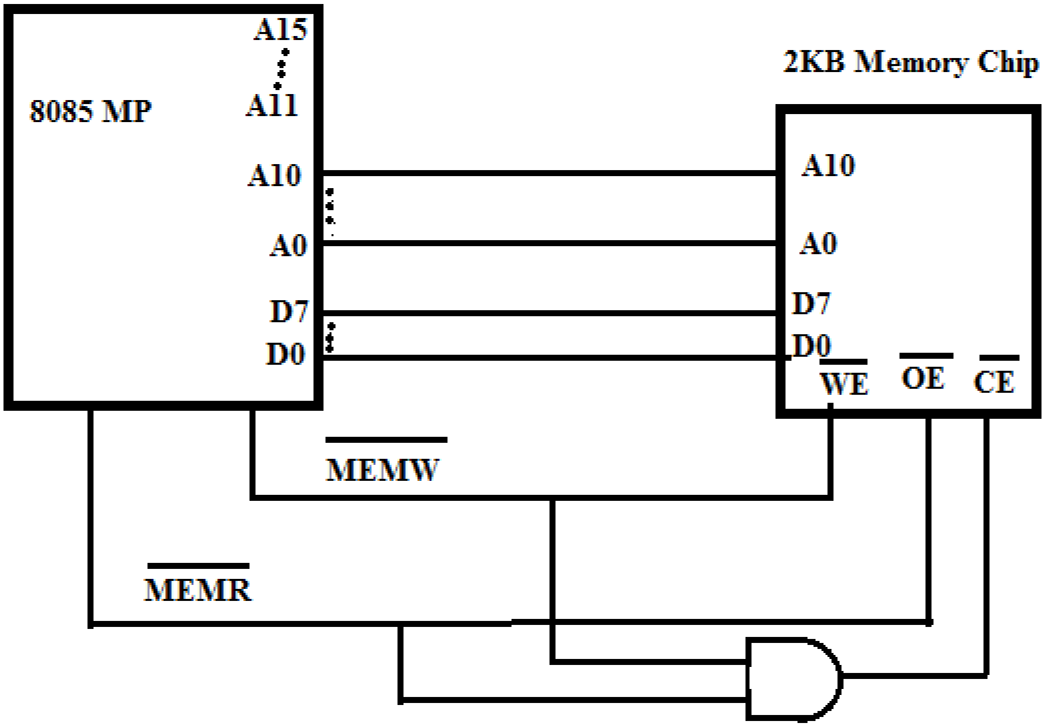


Fig: memory chip interfacing with microprocessor

Fig: memory chip interfacing with microprocessor

- → In the above figure, the lower address lines A0 – A10 of MP are connected to A0 – A10 pins of memory (2K) and D0 – D7 data lines of MP are connected to D0 – D7 of memory.
- → The memory read control signal MEMR(bar) and memory write control signal MEMW(bar) are connected to OE(bar) and WE(bar) pins of memory for read and write operation respectively.
- → When MP issued MEMW(bar) control signal to memory, then memory stores the data available in data lines into the addressed location. This process is known as memory write operation.
- → When MP issued MEMR(bar) control signal to memory, then memory places the data stored in addressed location towards the processor. This process is known as memory read operation.

Direct Memory Access (DMA)

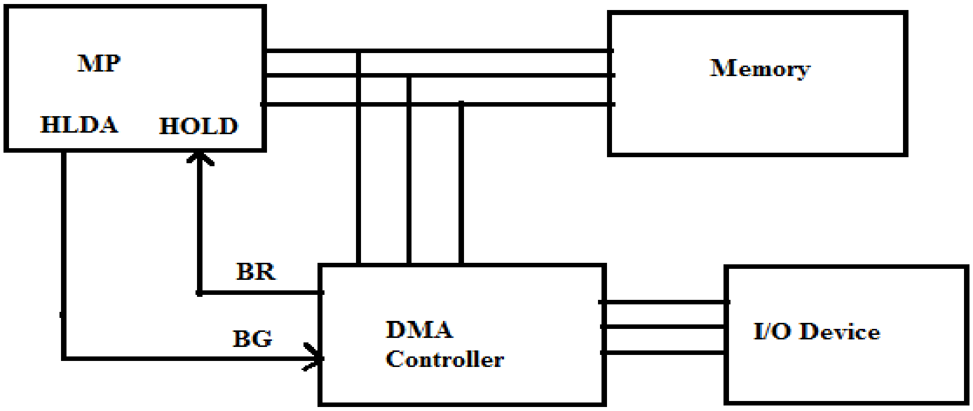


Fig: DMA

- → DMA is a process of communication for data transfer between memory and input/output, controlled by an external circuit called DMA controller, without involvement of CPU.
- → 8085 MP has two pins HOLD and HLDA which are used for DMA operation.
- → First, DMA controller sends a request by making Bus Request (BR) control line high. When MP receives high signal to HOLD pin, it first completes the execution of current machine cycle, it takes few clocks and sends HLDA signal to the DMA controller.
- → After receiving HLDA through Bus Grant (BG) pin of DMA controller, the DMA controller takes control over system bus and transfers data directly between memory and I/O without involvement of CPU. During DMA operation, the processor is free to perform next job which does not need system bus.
- → At the end of data transfer, the DMA controller terminates the request by sending low signal to HOLD pin and MP regains control of system bus by making HLDA low.

DMA controller 8237 Interfacing

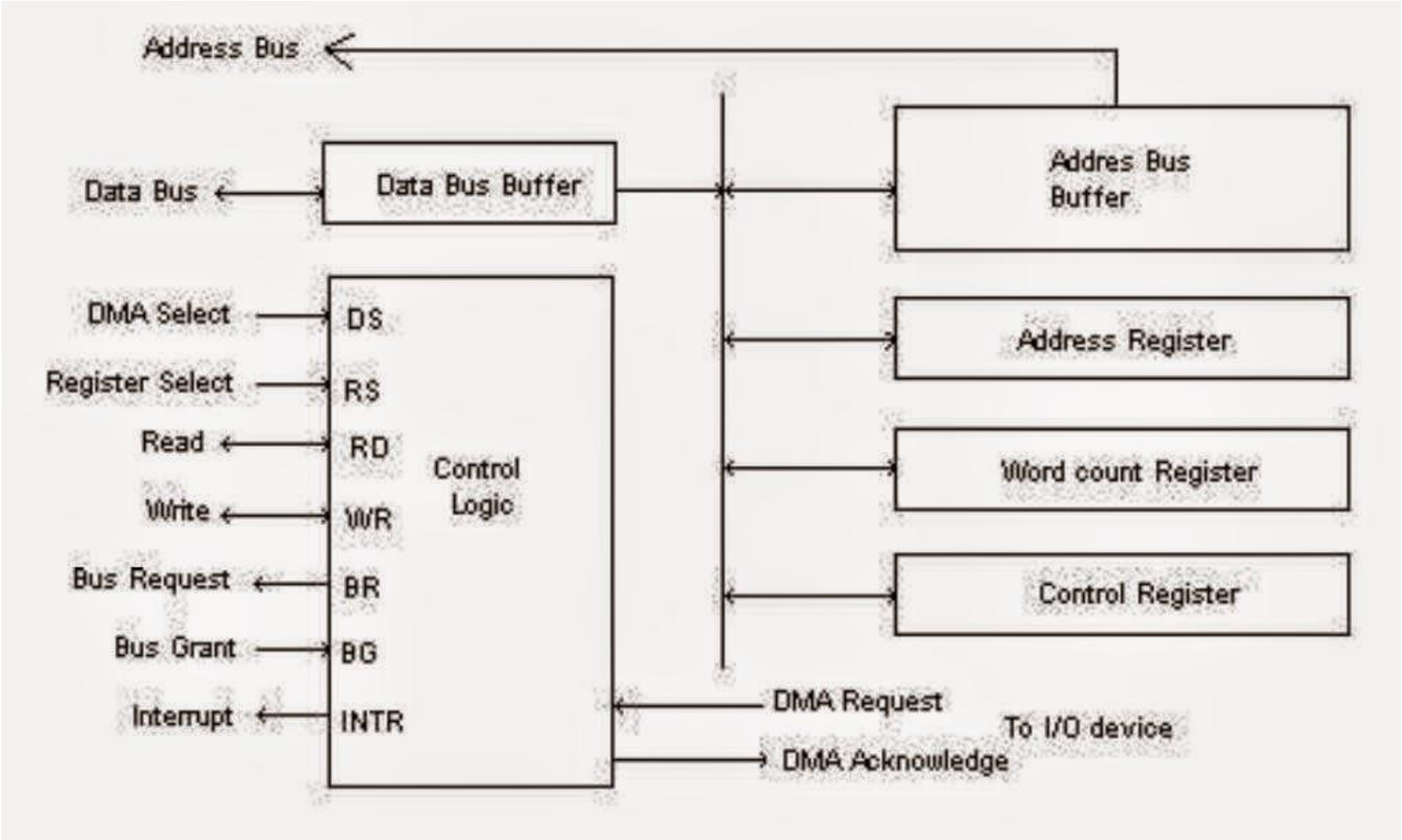


Fig: DMA Controller

- → Figure shows the block diagram of a typical DMA controller. The unit communicates with the MP via the data bus and control lines.
- → The registers in the DMA are selected by the MP through the address bus by enabling the DS (DMA select) and RS (Register Select) inputs. The RD (read) and WR (write) inputs are bidirectional.
- → When the bus grant (BG) input is 0, the MP can communicate with the DMA registers through the data bus to read from or write to the DMA registers. When BG=1, the processor does not have control over the system buses and the DMA can communicate directly with the memory by specifying an address in the address bus and activating the RD or WR control.
- → The DMA controller has three registers: an address register, a word count register and a control register.

→ The address register contains an address to specify the desired location in memory. The address bits go though bus buffers into the address bus. The address register is incremented after each word that is transferred to memory.

→ The word count register holds the number of words to be transferred. The register is decremented by one after each word transfer and internally tested for zero.

→ The control register specifies the mode of transfer.

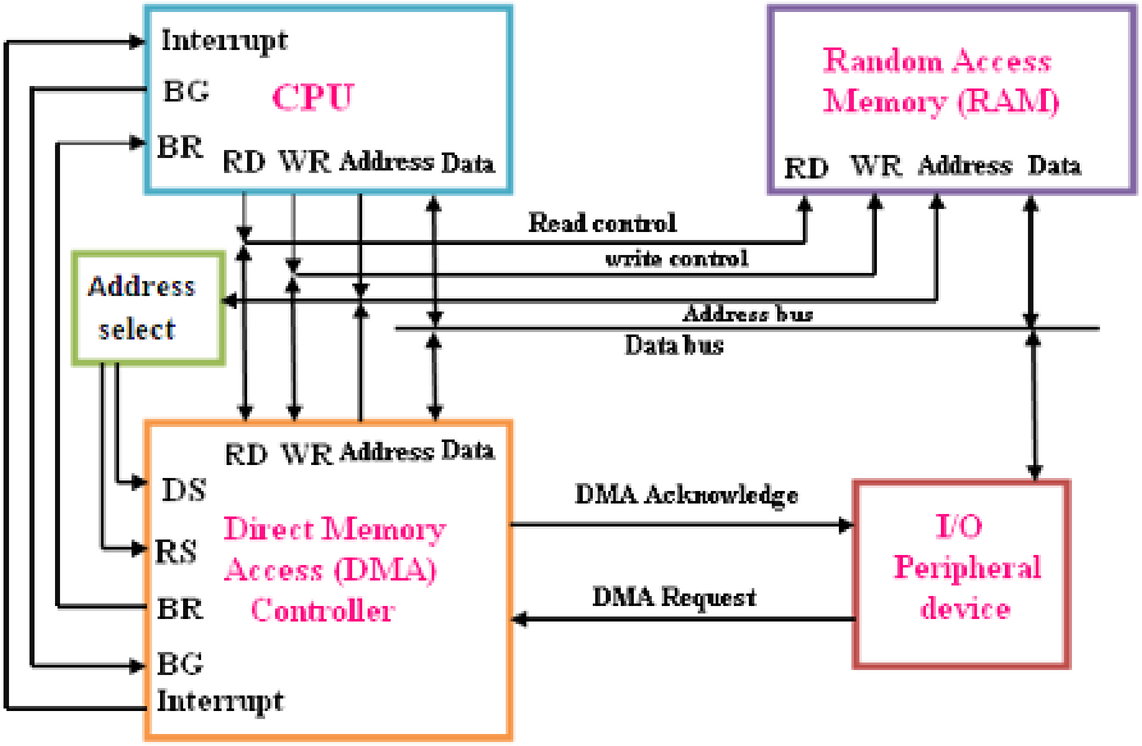


Fig: DMA Transfer

Interrupt

→ Interrupt is a process where an external device can get the attention of the microprocessor. The process starts from the I/O device. An interrupt is considered to be an emergency signal that may be serviced. The Microprocessor may respond to it as soon as possible.

→ 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 most probably have its own ISR.

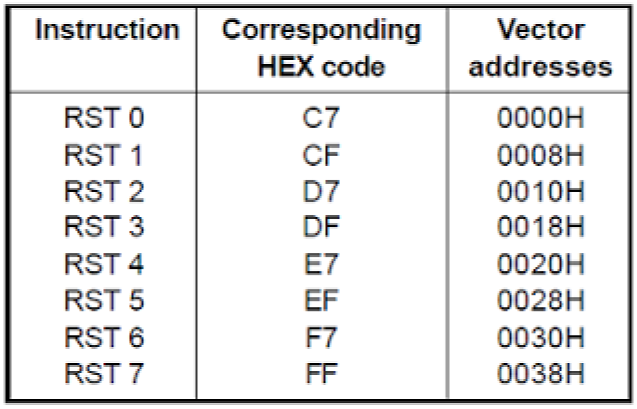
→ Responding to an interrupt may be immediate or delayed depending on whether the interrupt is maskable or non-maskable and whether interrupts are being masked or not. There are two ways of redirecting the execution to the ISR depending on whether the interrupt is vectored or non-vectored.

- **Vector Interrupt:** In this type of interrupt, Processor knows the address of Interrupt. In other word processor knows the address of interrupt service routine.
  The examples of vector interrupt are RST 7.5, RST 6.5, RST 5.5, TRAP.
- **Non-Vector Interrupt:** In this type of interrupt, Processor cannot know the address of Interrupt. It should give externally. In the device will have to send the address of interrupt service routine to processor for performing Interrupt.
  The example of Non-vector interrupt is INTR.

→ When a device interrupts, it actually wants the MP to give a service which is equivalent to asking the MP to call a subroutine. This subroutine is called ISR (Interrupt Service Routine).

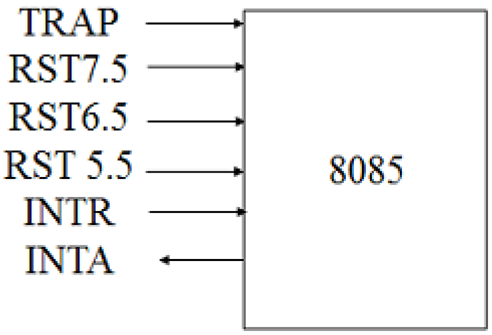
Software Interrupt: It is an instruction based Interrupt which is completely controlled by software. That means programmer can use this instruction to execute interrupt in main program.

There are eight software interrupt available in 8085 microprocessor. See the example with their hex code and vector address.



Instruction	Corresponding HEX code	Vector addresses
RST 0	C7	0000H
RST 1	CF	0008H
RST 2	D7	0010H
RST 3	DF	0018H
RST 4	E7	0020H
RST 5	EF	0028H
RST 6	F7	0030H
RST 7	FF	0038H

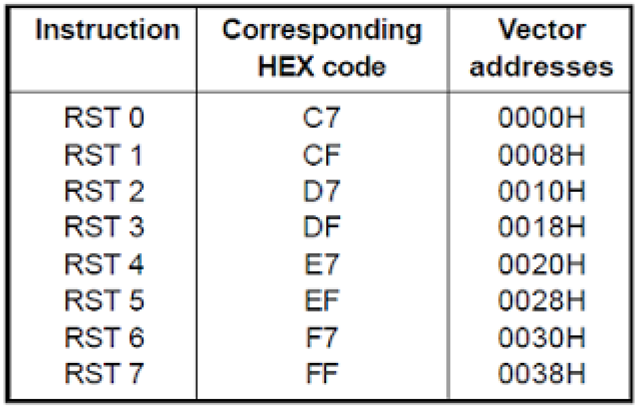
Hardware Interrupt: As name suggests it is interrupt which can get the interrupt request in hardware pin of microprocessor 8085. There are mainly six dedicated pins available for interrupt purpose.



Those are TRAP, RST 7.5, RST 6.5, RST 5.5, INTR, INTA (It is not an Interrupt pin but it is used to send acknowledgement of the Interrupt request getting from other interrupt pin.)

8085 Interrupt Pins and Interrupt Priority

There are five interrupt pins in 8085 and one interrupt acknowledge (INTA) pin.

<table>
<tr><td>Pin No.</td><td>Name</td><td>Type</td><td>Priority</td><td>ISR Location</td></tr>
<tr><td>6</td><td>TRAP</td><td>Vectored</td><td>Highest</td><td>CALL 0024H (3-byte call)</td></tr>
<tr><td>7</td><td>RST 7.5</td><td>Vectored</td><td>↓</td><td>CALL 003CH (3-byte call)</td></tr>
<tr><td>8</td><td>RST 6.5</td><td>Vectored</td><td></td><td>CALL 0034H (3-byte call)</td></tr>
<tr><td>9</td><td>RST 5.5</td><td>Vectored</td><td></td><td>CALL 002CH (3-byte call)</td></tr>
<tr><td>10</td><td>INTR</td><td>Non-Vectored</td><td>Lowest</td><td>RST (Restart instructions) – 1 byte call

<table>
<tr><th>Instruction</th><th>Corresponding HEX code</th><th>Vector addresses</th></tr>
<tr><td>RST 0</td><td>C7</td><td>0000H</td></tr>
<tr><td>RST 1</td><td>CF</td><td>0008H</td></tr>
<tr><td>RST 2</td><td>D7</td><td>0010H</td></tr>
<tr><td>RST 3</td><td>DF</td><td>0018H</td></tr>
<tr><td>RST 4</td><td>E7</td><td>0020H</td></tr>
<tr><td>RST 5</td><td>EF</td><td>0028H</td></tr>
<tr><td>RST 6</td><td>F7</td><td>0030H</td></tr>
<tr><td>RST 7</td><td>FF</td><td>0038H</td></tr>
</table></td></tr>
</table>

- → Pin 6 to pin 10 interrupts have the priorities from highest to lowest in decreasing order.
- → Priority means which interrupt gets the acknowledgement first if more than one are interrupting the microprocessor.

Maskable and Non-Maskable Interrupt

Maskable interrupts: An interrupt which can be disabled by software that means we can disable the interrupt by sending appropriate instruction, is called a maskable interrupt.
RST 7.5, RST 6.5, and RST 5.5 are the examples of Maskable Interrupt.

Non-Maskable interrupts: As name suggests we cannot disable the interrupt by sending any instruction is called Non Maskable Interrupt.
TRAP interrupt is the non-maskable interrupt for 8085. It means that if an interrupt comes via TRAP, 8085 will have to recognize the interrupt we cannot mask it.

Vectored and Polled Interrupt

Vectored Interrupt

In a computer, a vectored interrupt is an I/O interrupt that tells the part of the computer that handles I/O interrupts at the hardware level that a request for attention from an I/O device has been received and also identifies the device that sent the request.

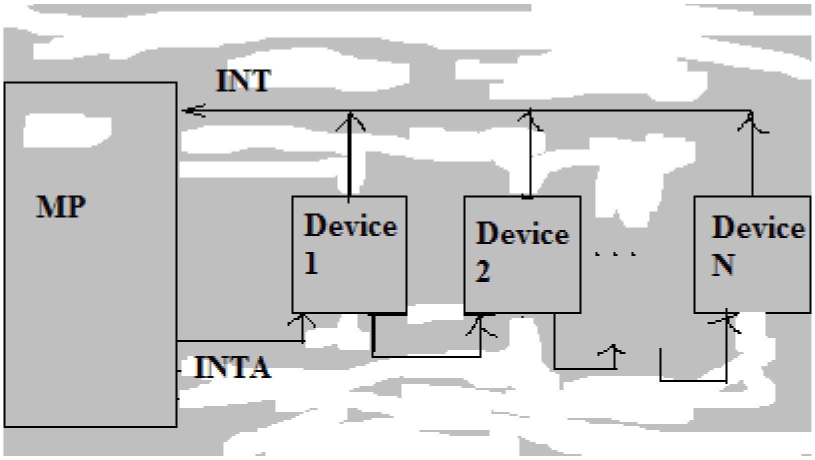


Fig: Vectored Interrupt

The device is connected in a chain as shown in figure above for setting up the priority systems. Suppose that one or more devices interrupt the processor at a time. In response, the processor saves its current status and then generates an interrupt acknowledge (INTA) signal to the highest priority device, which is device1, in this case. If this device has generated the interrupt, it will accept the INTA signal from the processor; otherwise, it will pass INTA on to the next device until INTA is accepted by the interrupting device.

Polled Interrupt

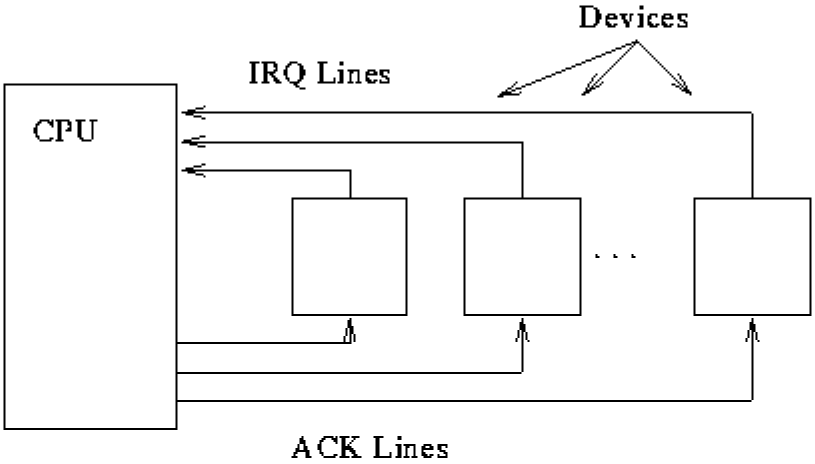


Fig: Polled Interrupt

In a computer, a polled interrupt is a specific type of I/O interrupt that notifies the part of the computer containing the I/O interface that a device is ready to be read or otherwise handled but does not indicate which device. The interrupt controller must poll (send a signal out to) each device to determine which one made the request.

Polled interrupts are handled using mostly software and are therefore slower compared to vectored (hardware) interrupts. The processor responds to an interrupt by executing one general service routine for all devices. The priority of these devices is determined by the order in which the routine polls each device. Once the processor determines the source of interrupt, it branches to the service routine for that device. The typical configuration of the polled interrupt is shown in figure above.

As shown in figure, several external devices (Device1, Device2,…….., Device N) are connected to a single interrupt line (INT) of the processor. When one or more devices activate the INT line high, the processor saves the content of the PC and other registers and then branches to an address defined by the manufacturer of the processor. The user can write a program at this address in order to poll each device starting with highest priority device in order to find the source of the interrupt.

Polled interrupts are very simple. But for a large number of devices, the time required to poll each device may exceed the time to service the device.

Programmable Interrupt Controller: The 8259A

The 8259A is a programmable interrupt-managing device, specifically designed for use with the interrupt signals (INTR/INT) of the 8085 MP.

The 8259A block diagram includes control logic, registers for interrupt requests, priority resolver, cascade logic, and data bus. The registers manage interrupt requests; the priority resolver determines their priority. The cascade logic is used to connect additional 8259A devices.

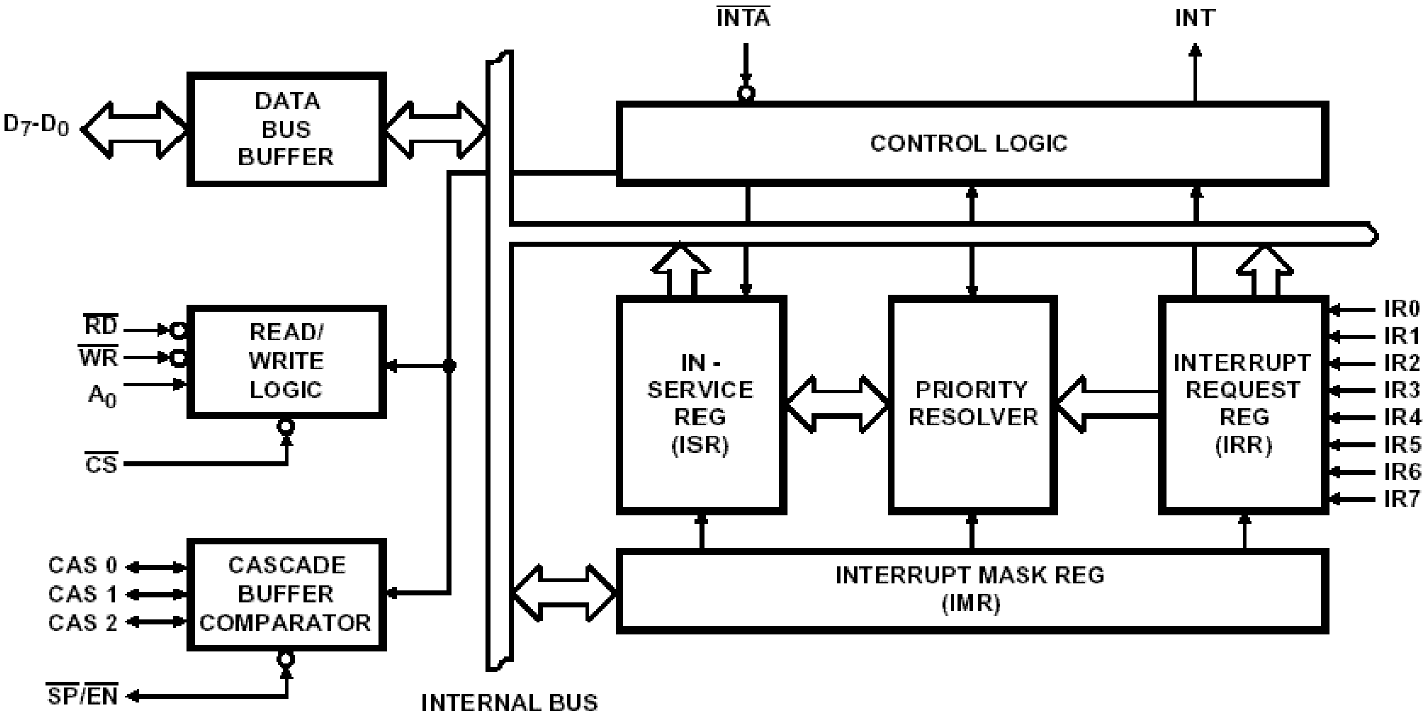


Fig: Block Diagram of 8259A PIC

The following steps take place during the operation of 8259A:

i) One or more interrupt request lines go high requesting the service.
ii) The 8259A resolves the priorities and sends an INT signal to the MP.
iii) The MP acknowledges the interrupt by sending INTA(bar).
iv) After the INTA(bar) has been received, the op-code for the call instruction (CDH) is placed on the data bus.
v) Because of the CALL instruction, the MP sends two more INTA(bar) signals.
vi) At the first INTA(bar), the 8259A places the low-order 8-bit address on the data bus and at the second INTA(bar), it places the high-order 8-bit address of the interrupt vector. This completes the 3-byte CALL instruction.
vii) The program sequence of the MP is transferred to the memory location specified by the CALL instruction.

Priority modes

i) Fixed-priority mode

→ IR0 has the highest priority and the following IR1, IR2, IR3…… etc. have the decreasing priorities.

ii) Automatic rotation mode

→ First priority changes to the last after its service.

iii) Specific rotation mode

→ This is user selectable or programmable, which means priority can be selected by programming.

Features

i) It manages 8 interrupt requests.

ii) It can vector an interrupt request anywhere in the memory map through program control without additional hardware for restart instructions. However, all 8 requests are spaced at the interval of either 4 locations or 8 locations.

iii) It can solve 8 levels of interrupt priorities in a variety of modes.

iv) With 8259A devices, the priority scheme can be expanded to 64 levels.

v) The 8259A has the abilities such as reading the status and changing the interrupt mode during a program execution.