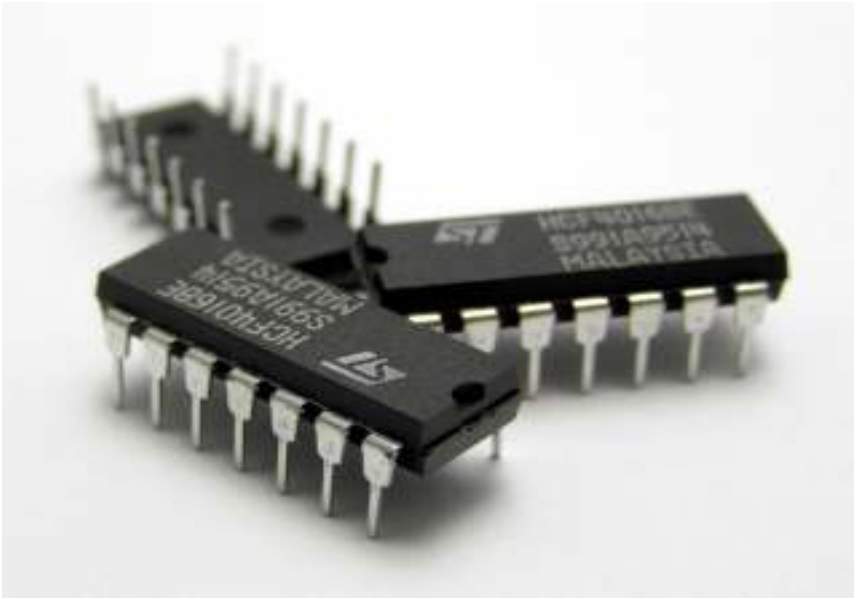
MPES
Module 1_1

Microprocessor

- Microprocessors are multipurpose devices that can be designed for generic or specialized functions.
- It is an integrated circuit contained on a single silicon chip, a microprocessor consists of the arithmetic logic unit, control unit, internal memory registers, and other vital circuitry of a computer's central processing unit (CPU).
- A central processing unit (CPU) contained within a single chip (integrated circuit). The term originated in the 1970s when processors were first miniaturized. Today, all CPUs are microprocessors, and server, desktop, laptop, smartphone and tablet microprocessors have more than one processing unit (dual core and multicore)

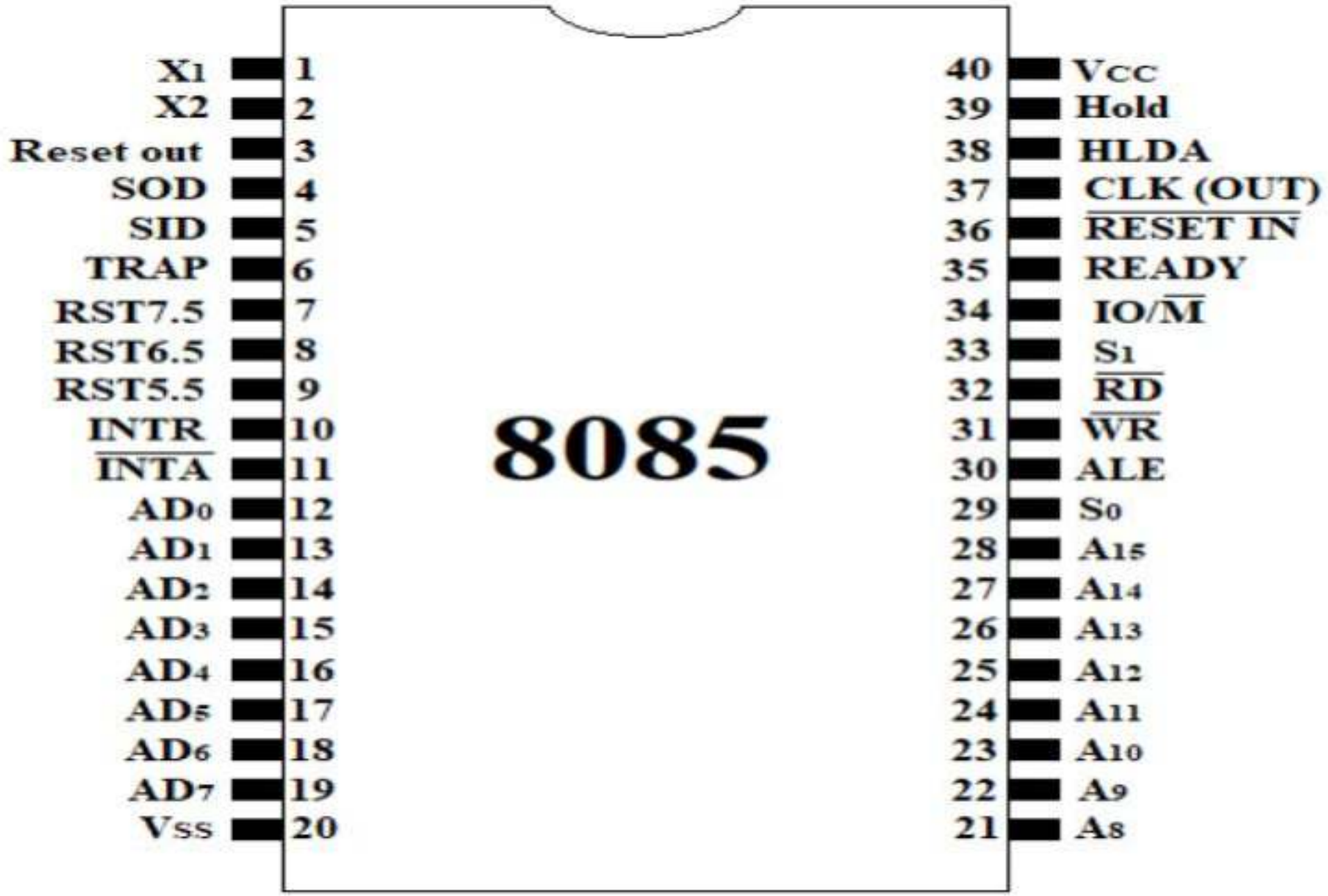
8085 Microprocessor

- 8 bit Microprocessor
- 40 pin DIP (Dual Inline package)

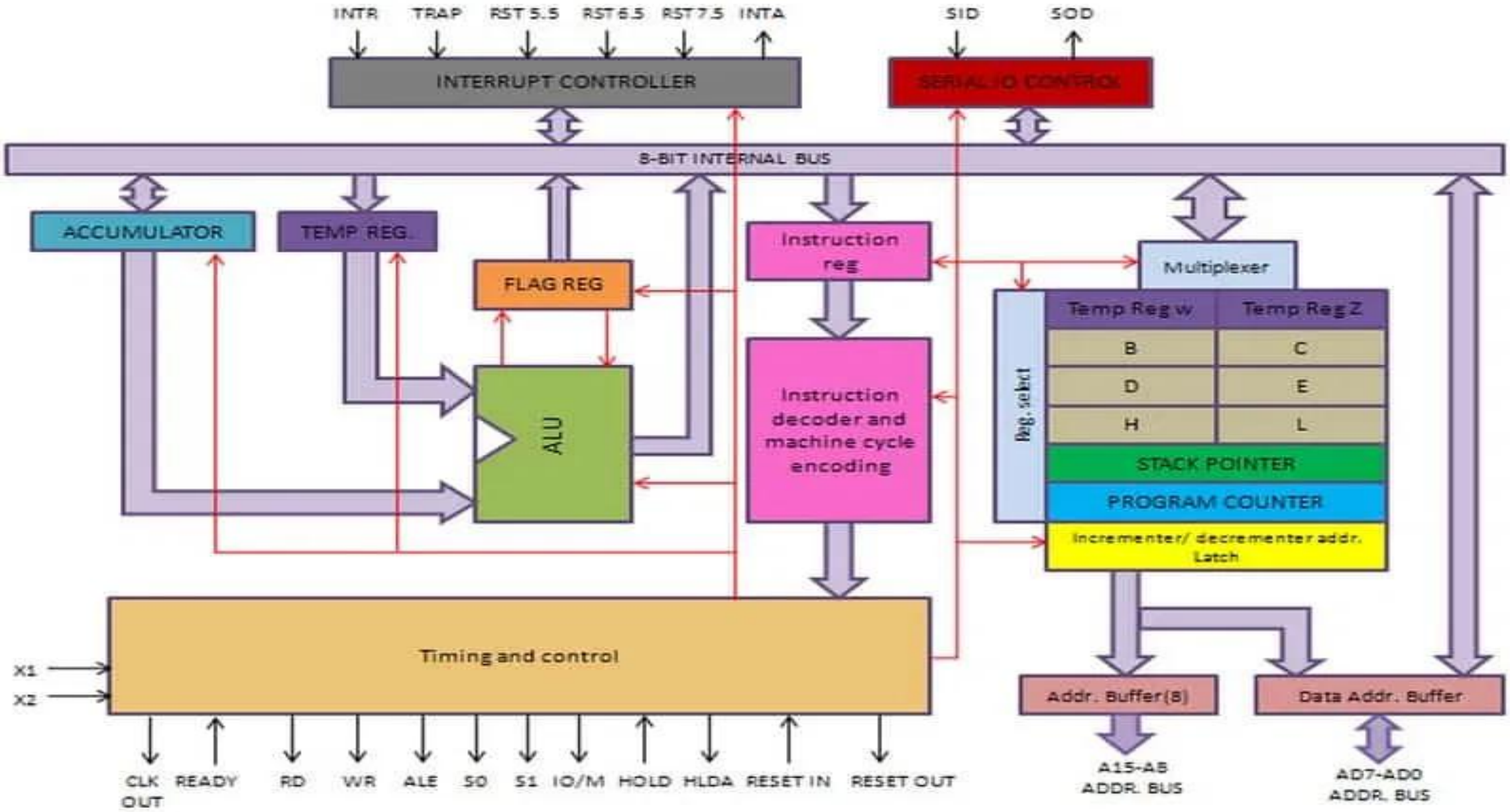


- Based on Von-Neumann architecture in which the data and instructions are in the same memory space without any distinction between them. That means it uses the same bus for data and address, so speed of operation is less compared to harvard architecture where separate address and data buses are there to access address and data from from the respective memory space.
- 5V, 3 MHz (internal) processor, 8 bit data and upto 16 bit address by multiplexing.

Pin Diagram of 8085



8085 Architecture



Arithmetic & Logic unit:

Performs 8 bit arithmetic and logic operations.

Accumulator:

8 bit register, Also known as register A.

Most important general purpose register in 8085.

During arithmetic and logical operations, one of the operands will be in the accumulator, also the results will be stored in accumulator.

Temperory register:

8 bit register to hold the second operand and intermediate results during arithmetic and logical operations.

Not accessible to the programmer.

W and Z registers:

8 bit temperory registers, can be used as WZ register pair to hold the 16bit address while executing certain instructions such as LDA C234H

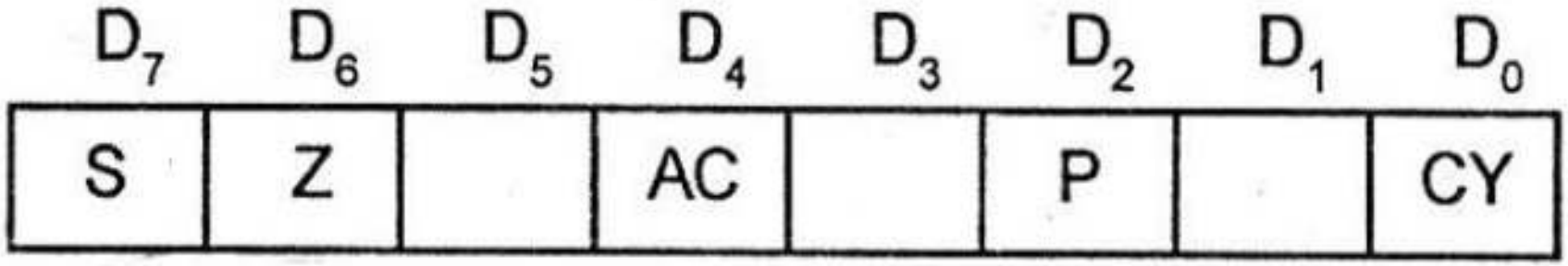
B,C,D,E, H and L Registers:

All are 8 bit general purpose registers. But if needed we can use register pairs as **BC, DE, and HL** to store the 16 bit information where left side register in each pair contain the most significant byte and the right side register contains the least significant byte.

HL register pair is more important as it has more functions such as it can be used as a memory pointer while accessing memory location with 16 bit address , also one operand during a 16bit addition etc should be in the HL register pair.

Also there are more ways to address the HL register pair compared to other register pairs.

FLAG Register in 8085:



Flag is an 8-bit register containing 5 1-bit flags:

Sign - set if the most significant bit of the result is set.

Zero - set if the result is zero.

Auxiliary carry - set if there was a carry out from bit 3 to bit 4 of the result.

Parity - set if the parity (the number of set bits in the result) is even.

Carry - set if there was a carry during addition, or borrow during subtraction/comparison.

Program Counter:

Program counter (PC) is a 16 bit register.

It contains the address of the next instruction to be executed.

After fetching the address from PC, it increaments by one.

Instruction Register:

Instruction register is a special purpose register used to receive the 8 bit opcode portion of an instruction.

not accessible to the programmer.

Instruction Decoder:

Instruction decoder decodes the information present in the Instruction register. Based on that the timing and control unit generates the timing sequences for executing that instruction.

Stack Pointer:

Stack pointer is a special purpose 16-bit register.

holds the address of the top of the stack.

Timing and Control Unit:

Responsible for generating timing and control signals to coordinate all the activities inside and outside the 8085 microprocessor to obtain its
desired output.

The extrnal oscillator frequency is divided by 2 internally and the approximate maximum internal frequency of operation of 8085 microprocessor is 3 MHz.

For an internal frequency of 3 MHz, the clock cycle or T state of 8085 is 333nS.

Also the minimum internal frequency of operation of 8085 is 500KHz.

Multiplexer / Demultiplexer

The registers B,C, D,E, H and L are connected to the internal data bus. So to choose a particular register during a register write or read operation,we need to select the specific register as given in the instruction.

MUX for register read and DeMUX for register write.

The **Register select unit** provides appropriate code to MUX or DeMUX to select the specific register in the instruction.

Address/Data Buffers:

The purpose of the buffer is to provide a means for electrically disconnecting the output pin from the external bus when not actually outputting data and the bus is being driven from a different device.

The buffers are bidirectional when used for data and unidirectional when used for address.

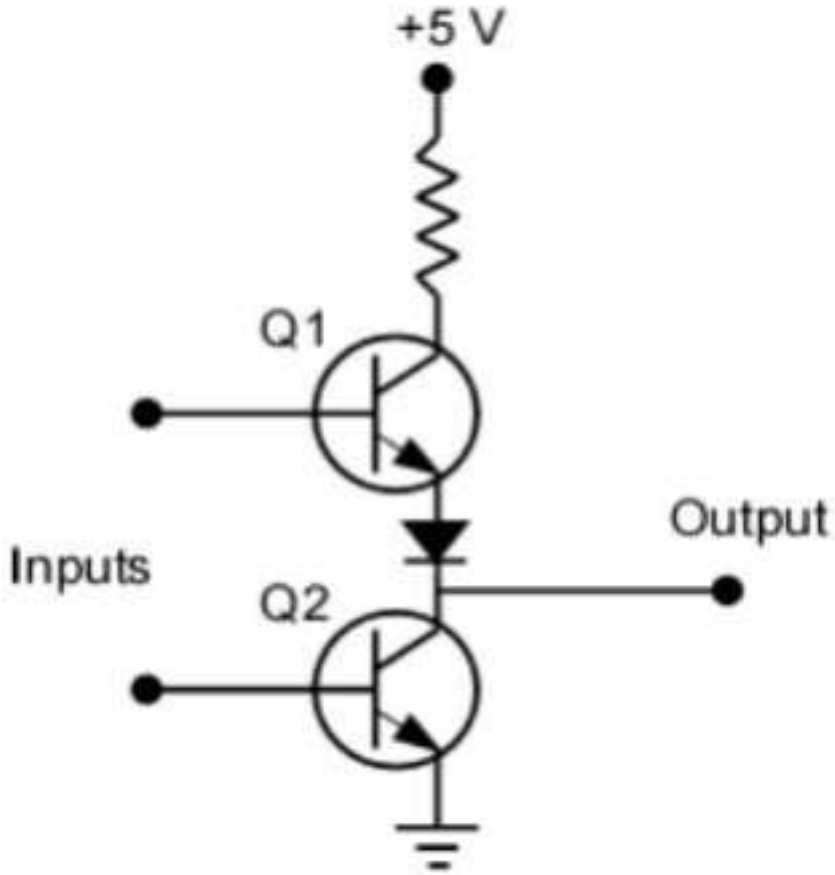


Fig. Example for a Tristate buffer

Interrupts :

Interrupts are the signals generated by the external devices to request the microprocessor to perform a task. There are 5 hardware interrupt signals, i.e. TRAP, RST 7.5, RST 6.5, RST 5.5, and INTR.

An interrupt controller is an integrated circuit that helps microprocessor to handle interrupt requests coming from multiple different sources (like external I/O devices) which may come simultaneously.

The TRAP has the highest priority followed by RST 7.5, RST 6.5, RST 5.5.

Serial I/O Controller:

8085 Microprocessor has two Serial Input/Output pins that are used to read/write one bit data to and from peripheral devices.

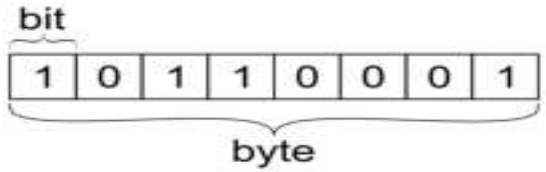
SOD and SID pins.

Thank You

MPES
Module 1_2

Introduction to Microprocessor languages:

- ✓ Microprocessors can recognise and operate with binary numbers alone.
- ✓ Each microprocessor have its own binary words, meanings and languages for its operation.
- ✓ The **Word** or Word length is the number of bits the microprocessor can recognise and process at a time. It can range from 4 bits (low speed) to 64 bits (high speed) or even more.
- ✓ A byte is a group of 8 bits, or it consists of two nibbles (lower and upper 4 bits in a group of 8 bits)



- ✓ To communicate with a micr n **Binary language** (**Machine language**) which the specific processor can understand.
- ✓ To overcome the difficulty in writing instructions with 0's and 1's, microprocessor manufacturers have formulated English like words to represent the binary instructions **specific for a processor**, called **Assembly language**.
- ✓ The programs written in assembly language is not transferrable from one processor to another (because processor specific).
- ✓ To overcome this , general purpose languages such as FORTRAN, 'C', Python etc are formed, which are processor independent, called **High level languages**

Contd…

- ✓An **Instruction** is a binary pattern used to command the microprocessor to perform a specific function.
- ✓The microprocessor developer selects combinations of bit patterns and gives specific meaning to each combination by using electronic logic circuits, and is called an Instruction. For example an 8 bit processor can have 2^8 = 256 combinations of eight bits, or 256 words. Instructions are made up of one or several words.
- ✓The set of instructions made for a processor makes up its **specific Binary or Machine Language.**
- ✓8085 is an 8 bit processor, with 74 different instructions.
- ✓For convenience, we can use hexadecimal code corresponding to a binary code, ie, 00110010 is a binary number can be written as 32_{16} or 32 H. But it is still difficult to write and understand programs using hex values.
- ✓To overcome this, manufacturers developed a symbolic code for each instruction called **mnemonic,** which are processor specific. As an example binary code 00111100 or 3C H is represented by INR A, instruction to increment accumulator register by one.
- ✓The complete set of 8085 mnemonics is called 8085 assembly language.
- ✓Machine Language and Assembly language are processor specific and are called **Low Level Languages.**

Contd...

- ✓Since the microprocessor can understand only binary values, the programs written in assembly language has to be translated to binary equivalent either manually (Hand assembly) or with the help of **Assembler.**
- ✓**Assembler** is a program which translates the mnemonics into its corresponding binary machine codes.
- ✓Similarly processor or machine independent high level languages are translated to binary equivalent (object code) with the help of **Compiler or Interpreter.**
- ✓**Compiler** is a program which translates high level languages in to machine language . A compiler reads a given program (source code) entirely and translates to machine code (object code).
- ✓**Interpreter** is a program which translates high level to machine language ,one statement at a time.

Thank You

MPES
Module 1_3

Instruction Format of 8085:

- An instruction is a command to the microprocessor to perform a given task on a specified data.
- Each instruction has two parts: one is task to be performed, called the operation code (opcode), and the second is the data to be operated on, called the operand. The operand (or data) can be specified in various ways. It may include 8-bit (or 16-bit) data, an internal register, a memory location, or 8-bit (or 16-bit) address. In some instructions, the operand is implicit.
- The 8085 has 74 basic instructions and 246 total instructions. The instruction set of 8085 is defined by the manufacturer INTEL Corporation.
- The size of 8085 instruction can be one-byte, two bytes or three bytes.
- A **1-byte instruction** includes the opcode and operand in the same byte. These instructions are stored in 8- bit binary format in memory; each requires one memory location.

Example:

Task	Op code	Operand	Binary Code	Hex Code
Copy the contents of the accumulator in the register C.	MOV	C,A	0100 1111	4FH

Contd….

- In a **two-byte instruction**, the first byte specifies the operation code and the second byte specifies the operand. Source operand is a data byte immediately following the opcode.

Example:

Task	Opcode	Operand	Binary Code	Hex Code	
Load an 8-bit data byte in the accumulator.	MVI	A, Data	0011 1110 DATA	3E Data	First Byte Second Byte

- In a **three-byte instruction**, the first byte specifies the opcode, and the following two bytes specify the 16-bit address. Note that the second byte is the low-order address and the third byte is the high-order address.

Example:

Task	Opcode	Operand	Binary code	Hex Code	
Transfer the program sequence to the memory location 2085H.	JMP	2085H	1100 0011 1000 0101 0010 0000	C3 85 20	First byte Second Byte Third Byte

Addressing Modes in 8085:

Addressing modes refers to the way in which the operand of an instruction is specified. In 8085 there are five addressing modes.

- **Immediate Addressing**
- **Direct Addressing**
- **Register Addressing**
- **Register Indirect Addressing**
- **Implied Addressing**

1. Immediate Addressing:

The data is specified in the instruction itself.

Example: MVI B, 3EH

Moves the data 3EH given in the instruction to B-register.

2. Direct Addressing:

The address of the data is specified in the instruction. The data will be in the memory.

Example: LDA 1050H

Load the data available in memory location 1050H in accumulator

3. Register Addressing :

The instruction specifies the name of the register in which the data is available.

Example: MOV A, B

Moves the content of B-register to A-register.

4. Register Indirect Addressing:

The instruction specifies the name of the register in which the address of the data is available. Here the data will be in memory and the address will be in a register pair.

Example: MOV A, M

The memory data addressed by HL pair is moved to A-register.

5. Implied Addressing:

In implied addressing mode, the instruction itself specifies the data to be operated.

Example: CMA

Complements the content of accumulator.

Thank You

MPES
Module 1_4

Instruction Cycle in 8085:

- The Program and data which are stored in the memory, external to the microprocessor are used for executing the complete instruction.
- The sequence of operations that a processor has to carry out while executing an instruction is called instruction cycle.
- Each instruction cycle of a processor in turn consists of a number of machine cycles. Thus to execute a complete instruction of the program, the following steps should be performed by the 8085 microprocessor.
    - Fetching the opcode from the memory;
    - Decoding the opcode to identify the specific set of instructions;
    - Fetching the remaining Bytes left for the instruction, if the instruction length is of 2 Bytes or 3 Bytes;
    - Executing the complete instruction procedure.

The given steps altogether constitute the complete **instruction cycle.**

Machine Cycle in 8085:

- The time required to access the memory or input/output devices is called Machine cycle.
- To execute an instruction, the processor will run one or more machine cycles in a particular order.
- The seven Machine Cycle in 8085 Microprocessor are :
  - **Opcode Fetch Cycle**
  - **Memory Read**
  - **Memory Write**
  - **I/O Read**
  - **I/O Write**
  - **Interrupt Acknowledge**
  - **Bus Idle**

T-State:

- The machine cycle and instruction cycle takes multiple clock periods.
- The T-state is the time period of the internal clock signal of the processor.
- A portion of an operation carried out in one system clock period is called as T- state.
- The time taken by the processor to execute a machine cycle is expressed in T-state

Instruction Set in 8085:

Classified as,

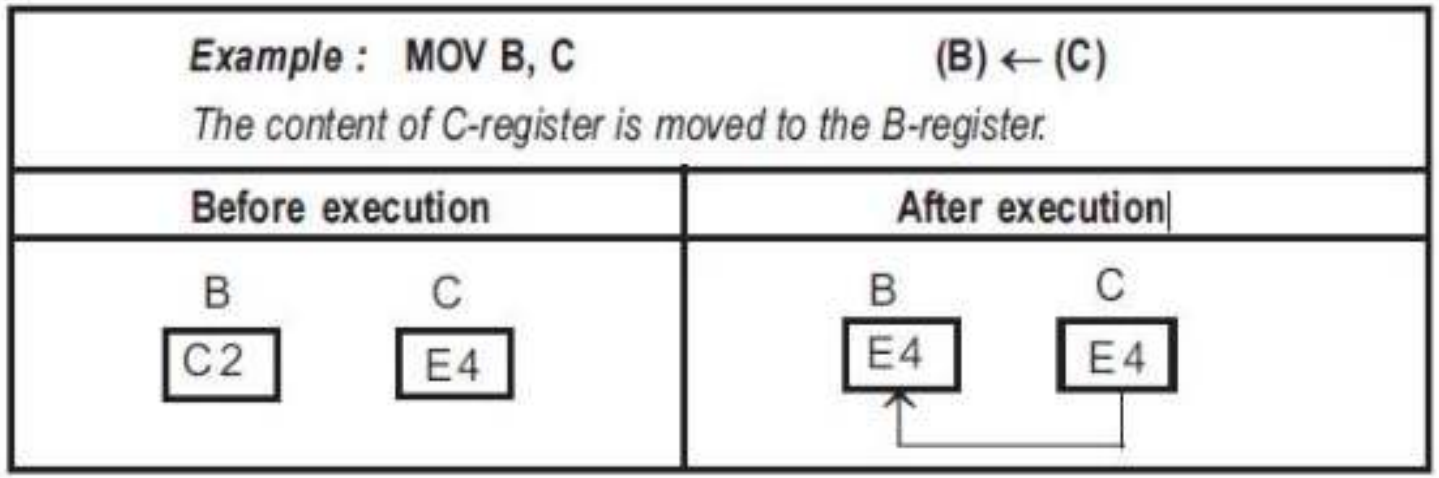
1. Data Transfer Instructions
2. Arithmetic Instructions
3. Logical Instructions
4. Branching Instructions
5. Machine Control Instructions

Data Transfer Instructions:

- The instructions that moves (copies) data between registers or between memory location and register.
- In all data transfer operations, the content of source register or memory is not altered. Hence the data transfer is copying operation.

Data transfer instructions contd…:

- MOV Rd, Rs (Rd) ← (Rs)
- The content of source register (Rs) is copied to the destination register (Rd). The registers Rd and Rs can be any one of the general purpose registers A, B, C, D, E, H or L.
- No flags are affected.



- One byte instruction
- One machine cycle : Opcode fetch - 4T
- Register addressing
- Total number of instructions = 49

Contd…

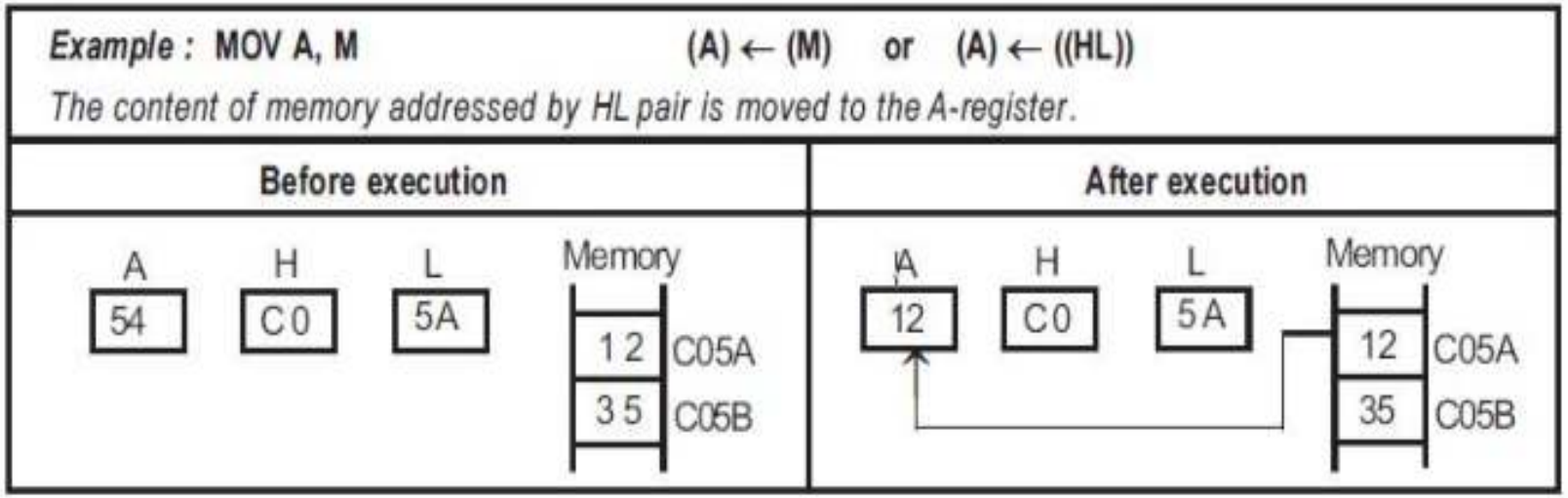
- MOV Rd, M (Rd) ← (M) or (Rd) ← ((HL))
- The content of memory (M) addressed by the HL pair is moved to the destination register (Rd). The register Rd can be any one of the general purpose registers A, B, C, D, E, H or L.
- No flags are affected.
- One byte instruction
- Two machine cycles:

  Opcode fetch - 4T

  Memory read - 3T

  Total - 7T
- Register indirect addressing
- Total number of instructions = 7

  MOV A, M MOV B, M MOV C, M MOV D, M MOV E, M MOV H, M MOV L, M

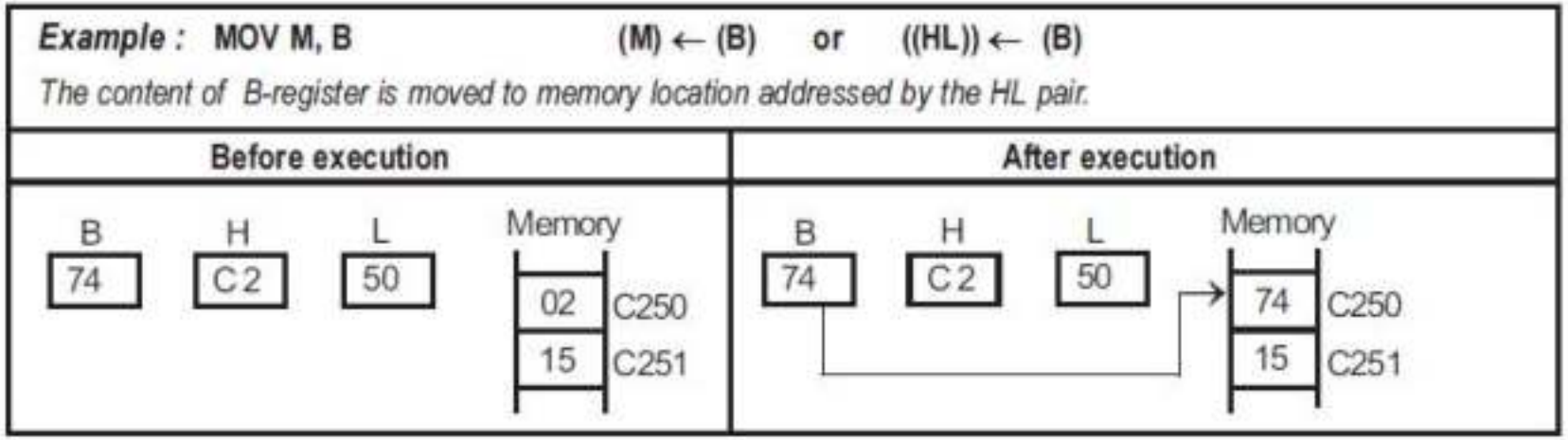


Contd…

- MOV M, Rs (M) ← (Rs) or ((HL)) ← (Rs)
- The content of source register (Rs) is moved to the memory location addressed by HL pair. The register Rs can be any one of the general purpose registers A, B, C, D, E, H or L.
- No flags are affected.
- One byte instruction
- Two machine cycles :

  Opcode fetch - 4T

  Memory write - 3T
- Register indirect addressing
- Total number of instructions = 7



MOV M,A MOV M,B MOV M, C MOV M, D MOV M, E MOV M, H MOV M, L

Thank You

MPES
Module 1_5

Data transfer Instructions Contd….

- **MVI Rd, d8 (Rd) ← d8**
- The 8-bit data (d8) given in the instruction is moved to the destination register (Rd). The register Rd can be any one of the general purpose registers A, B, C, D, E, H or L.
- No flags are affected.
- Two byte instruction
- Two machine cycles:

  Opcode fetch - 4T

  Memory read - 3T
- Immediate addressing mode
- Total number of instructions = 7

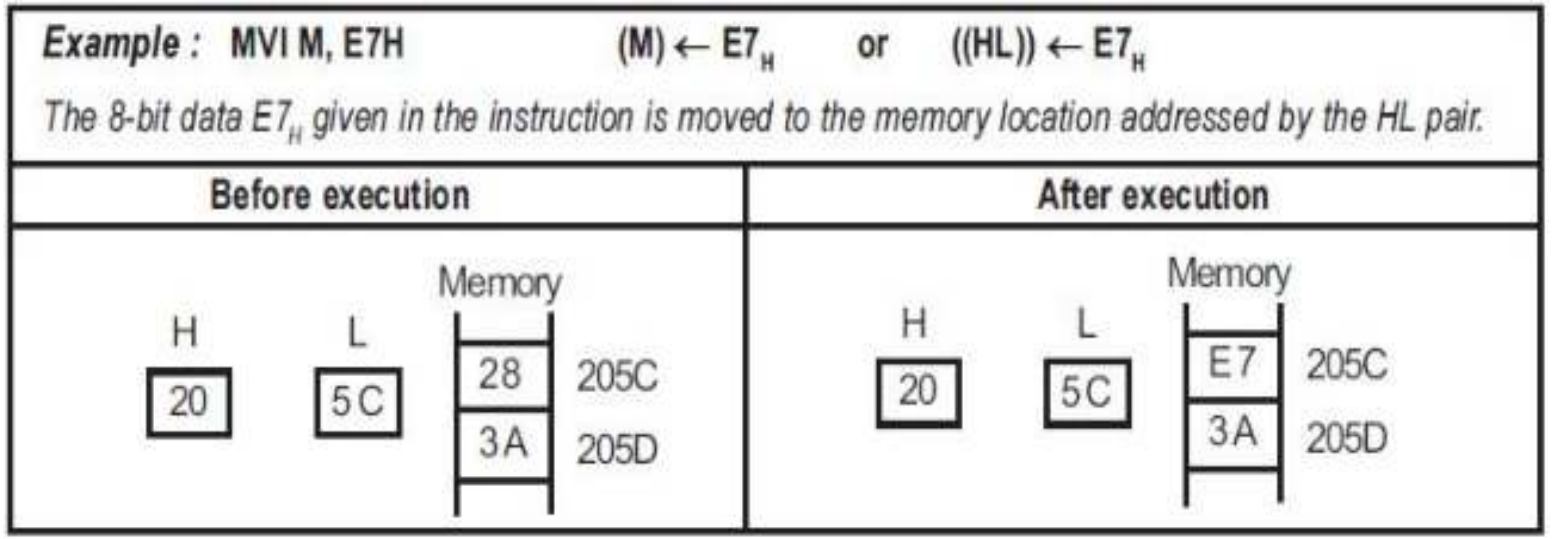
  MVI A, d8 MVI B, d8 MVI C, d8 MVI D, d8 MVI E, d8 MVI H, d8 MVI L, d8

Example : **MVI D,09H** **(D) ← 09_H**

The 8-bit data 09_H given in the instruction is moved to the D-register.

Before execution	After execution
D C2	D 09

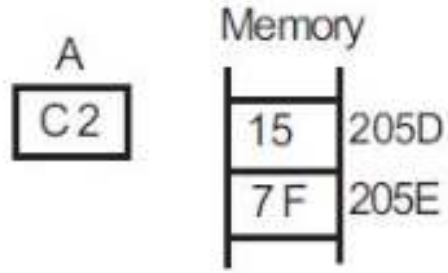
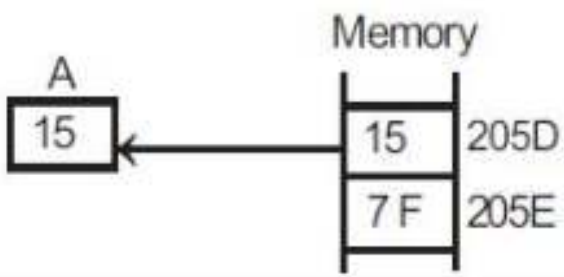
- **MVI M, d8** **(M) ← d8 or ((HL)) ← d8**
- The 8-bit data (d8) given in the instruction is moved to the memory location addressed by the HL pair.
- No flags are affected.
- Two byte instruction
- Three machine cycles :
  - Opcode fetch - 4T
  - Memory read - 3T
  - Memory write - 3T
- Register indirect addressing
- Or Immediate addressing
- Total number of instructions = 1



- **LDA addr16 (A) ← (M) or (A) ← (addr16)**
- The content of the memory location whose address is given in the instruction, is moved to accumulator.
- No flags are affected.
- Three byte instruction
- Four machine cycles :
  - Opcode fetch - 4T
  - Memory read - 3T
  - Memory read - 3T
  - Memory read - 3T

Example : **LDA 205DH** **(A) ← ($205D_H$)**

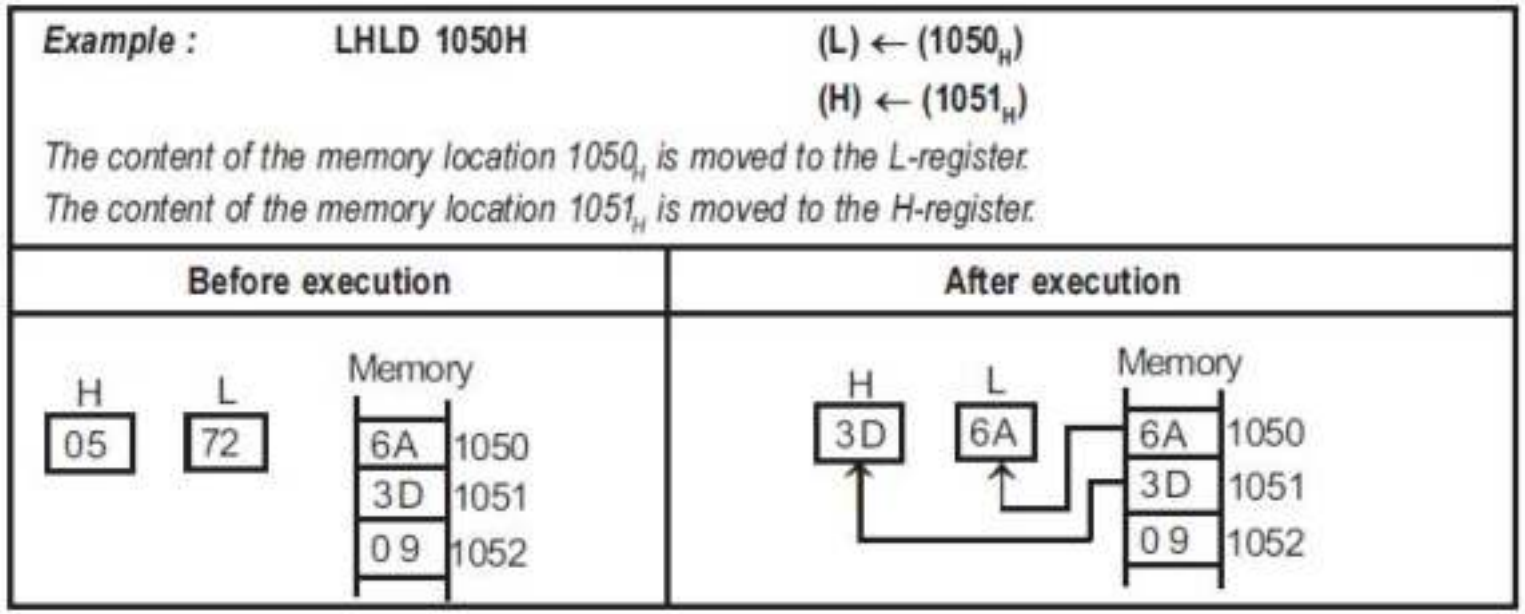
The content of the memory location $205D_H$ is moved to the A-register.

Before execution	After execution
	

- Direct addressing
- Total number of instructions = 1

Contd...

- **LHLD addr16** **(L) ← (M) or (L) ← (addr16)**
  (H) ← (M) or (H) ← (addr16 + 01)
- The content of the memory location whose address is given in the instruction, is moved to the L-register. The content of the next memory location is moved to the H-register.
- No flags are affected.
- Three byte instruction
- Five machine cycles:
  - Opcode fetch - 4T
  - Memory read - 3T
  - Memory read - 3T
  - Memory read - 3T
  - Memory read - 3T
- Direct addressing
- Total number of instructions = 1



- **LXI rp, d16** **(rp) ← d16**
- The 16-bit data given in the instruction is moved to the register pair (rp). The register pair can be BC, DE, HL or SP.

Example : LXI H, 1050H

$(L) \leftarrow 50_H$

$(H) \leftarrow 10_H$

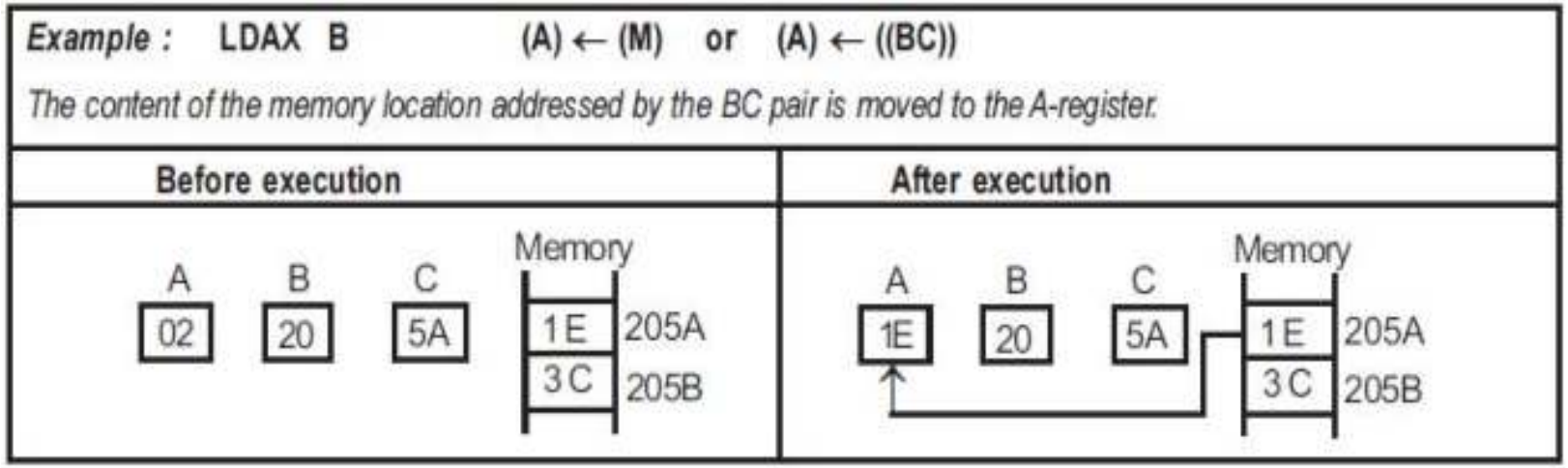
The 16-bit data 1050_H given in the instruction is moved to the HL register pair.

Before execution	After execution
H: xx, L: yy (some arbitrary value)	H: 10, L: 50

- Three byte instruction
- Three machine cycles :
  - Opcode fetch - 4T
  - Memory read - 3T
  - Memory read - 3T
- Immediate addressing
- Total number of instructions = 4

LXI B, d16 LXI D, d16 LXI H, d16 LXI SP, d16

- **LDAX rp** **(A) ← (M) or (A) ← ((rp))**
- The content of the memory addressed by the register pair (rp) is moved to the accumulator. (The content of the register pair is the memory address). The register pair can be either BC or DE.
- One byte instruction
- Two machine cycles:
  - Opcode fetch - 4T
  - Memory read - 3T

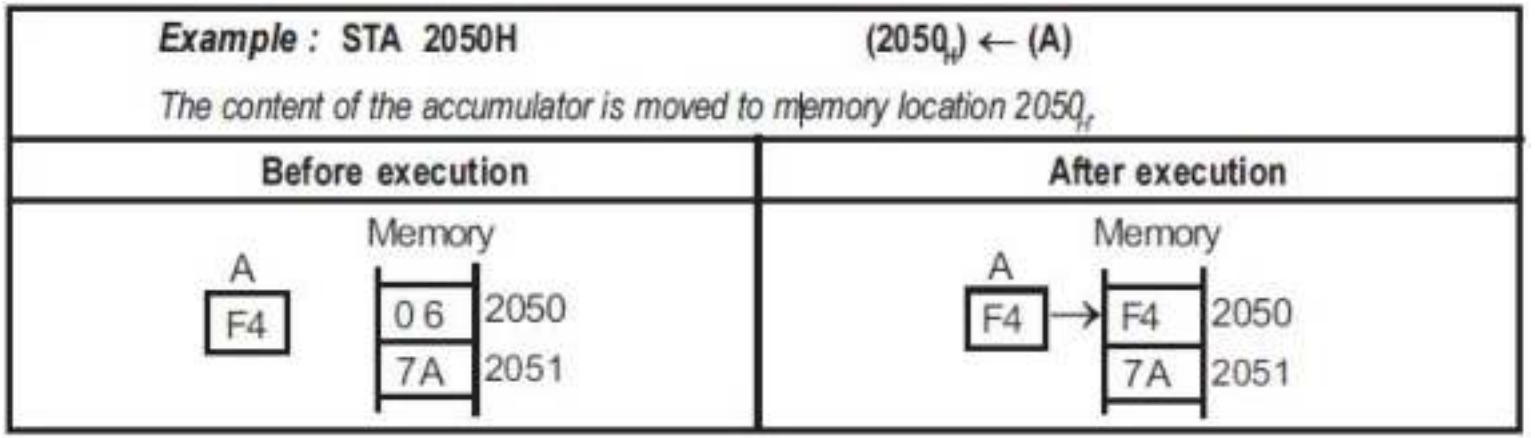


- Register indirect addressing
- Total number of instructions = 2

LDAX B LDAX D

Contd....

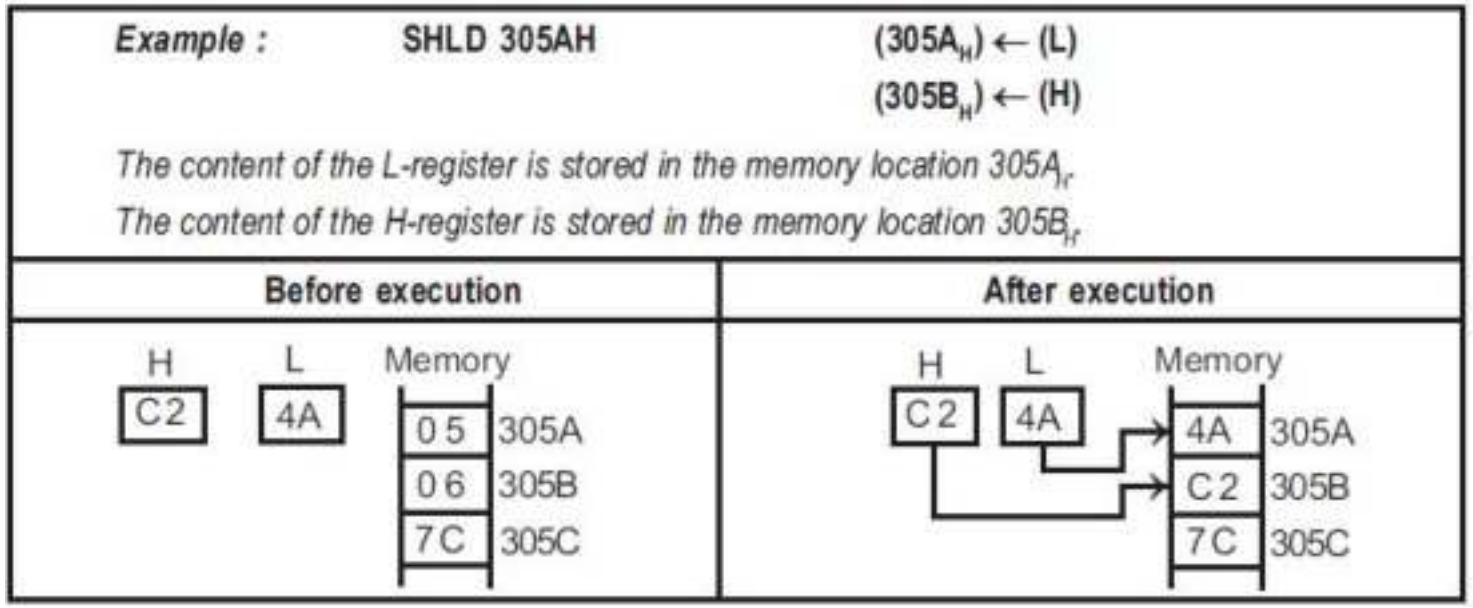
- **STA addr16** **(M) ← (A) or (addr16) ← (A)**
- The content of the accumulator is moved to the memory . The address of the memory location is given in the instruction.
- No flags are affected.
- Three byte instruction
- Four machine cycles:
  - Opcode fetch - 4T
  - Memory read - 3T
  - Memory read - 3T
  - Memory write- 3T
- Direct addressing Mode
- Total number of instructions = 1



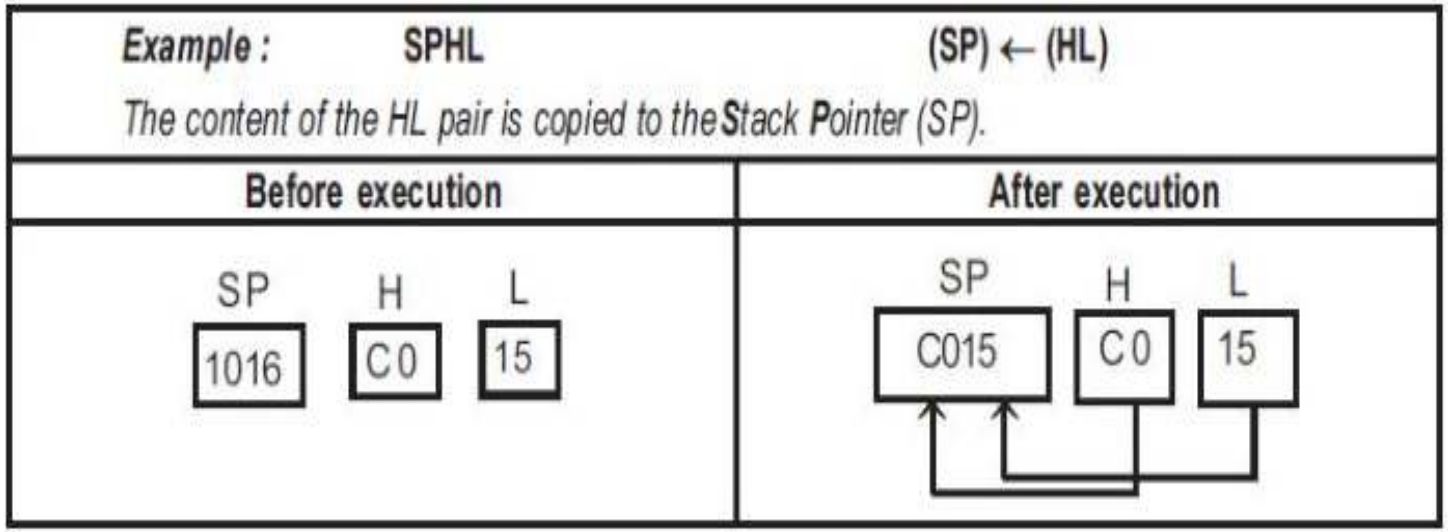
Contd....

- **SHLD addr16** **(M) ← (L) or (addr16) ← (L)**
  (M) ← (H) or (addr16+1) ← (H)

- The content of the L-register is stored in the memory location, whose address is given in the instruction. The content of the H-register is stored in the next memory location.
- No flags are affected.
- Three byte instruction
- Five machine cycles:
  - Opcode fetch - 4T
  - Memory read - 3T
  - Memory read - 3T
  - Memory write - 3T
  - Memory write - 3T
- Direct addressing
- Total number of instructions = 1



- **SPHL** **(SP) ← (HL)**
- The content of the HL pair is moved to the Stack Pointer (SP).
- No flags are affected.
- One byte instruction
- One machine cycle:
  - Opcode fetch - 6T
- Implied addressing
- Total number of instructions = 1



Thank You

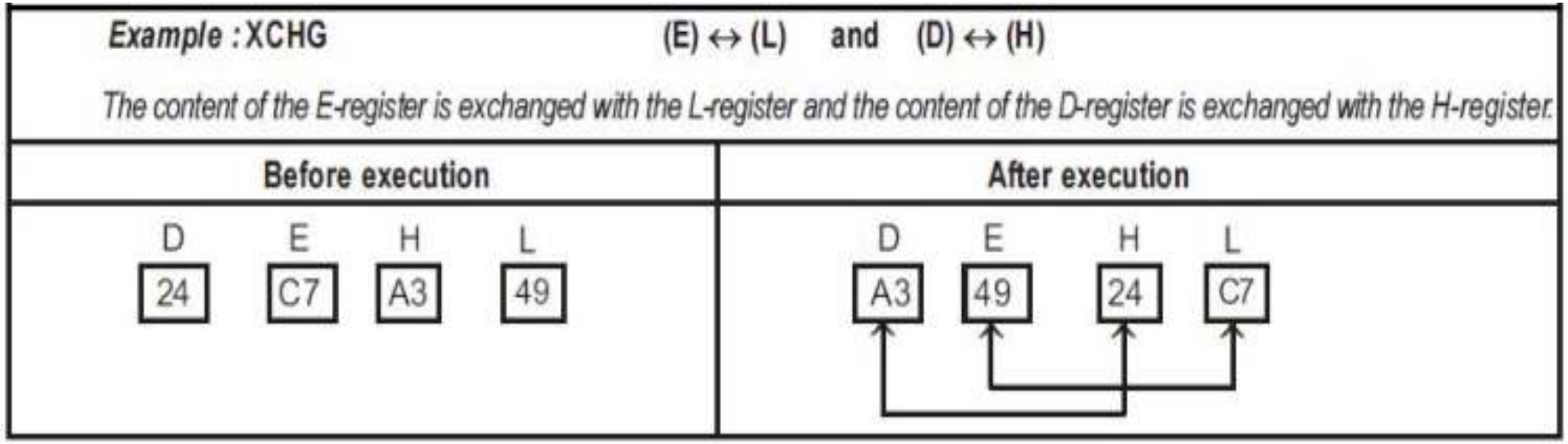
MPES
Module 1_6

XCHG

(E) ↔ (L)

(D) ↔ (H)

- The content of the HL pair is exchanged with the DE pair.
- No flags are affected.
- One byte instruction
- One machine cycle:
  - Opcode fetch - 4T
- Implied addressing
- Total number of instructions = 1



Contd…

PUSH rp $(SP) \leftarrow (SP) - 1$; $((SP)) \leftarrow (rp)_H$

$(SP) \leftarrow (SP) - 1$; $((SP)) \leftarrow (rp)_L$

- The content of the register pair (rp) is pushed to the stack. After execution of this instruction, the content of the Stack Pointer (SP) will be 02 less than the earlier value.
- The register pairs can be BC, DE , HL and PSW.
- No flags are affected.
- PSW (Program Status Word) : Accumulator and Flag register together called PSW. Accumulator is high order register and Flag register is low order register.
- The instruction is executed as follows:

  (i) The content of the SP is decremented by one.

  (ii) The content of the high order register is moved to memory addressed by SP.

  (iii) The content of the SP is decremented by one.

  (iv) The content of the low order register is moved to memory addressed by SP

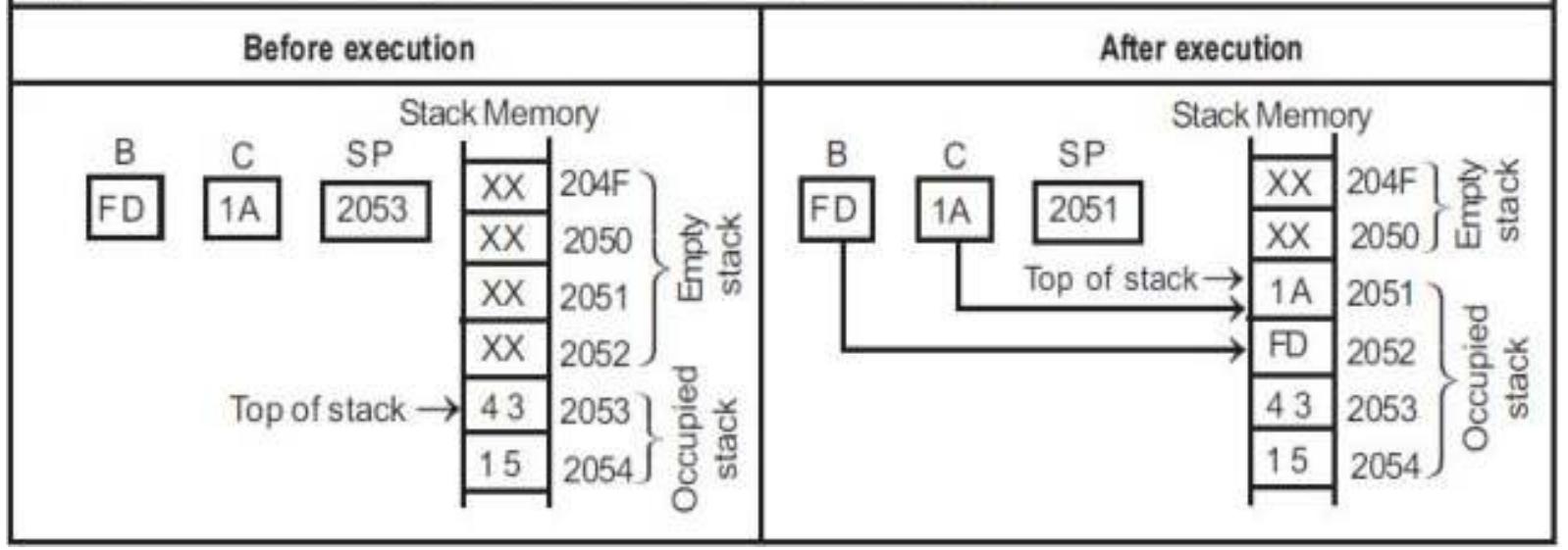
Contd....

- One byte instruction
- Three machine cycles:
  - Opcode fetch - 6T
  - Memory write - 3T
  - Memory write - 3T
- Register indirect addressing

Example : PUSH B

(SP) ← (SP) – 01
((SP)) ← (B)
(SP) ← (SP) – 01
((SP)) ← (C)

(i) The content of the SP is decremented by one.
(ii) The content of the B-register is moved to the memory addressed by the Stack Pointer (SP).
(iii) Again the content of SP is decremented by one.
(iv) The content of the C-register is moved to the memory addressed by SP.



- Total number of instructions = 4

PUSH PSW PUSH B PUSH D PUSH H

POP rp (rp)L ← ((SP)) ; (SP) ← (SP) + 1

(rp)H ← ((SP)) ; (SP) ← (SP) + 1

- The content of top of stack memory is moved to the register pair. After execution of this instruction the content of the Stack Pointer (SP) will be 02 greater than the earlier value.
- The register pairs can be BC, DE , HL and PSW.
- No flags are affected.
- PSW (Program Status Word) : Accumulator and Flag register are together called PSW. The accumulator is a high order register and the flag register is a low order register.
- The pop instruction is executed as follows:

  (i) The content of the memory addressed by the SP is moved to the low order register.

  (ii) The content of the SP is incremented by one.

  (iii) The content of the memory addressed by the SP is moved to the high order register.

  (iv) The content of the SP is incremented by one.

Contd....

-
-
  -
  -
  -

Example : POP D

(E) ← ((SP))
(SP) ← (SP) + 01
(D) ← ((SP))
(SP) ← (SP) + 01

4T

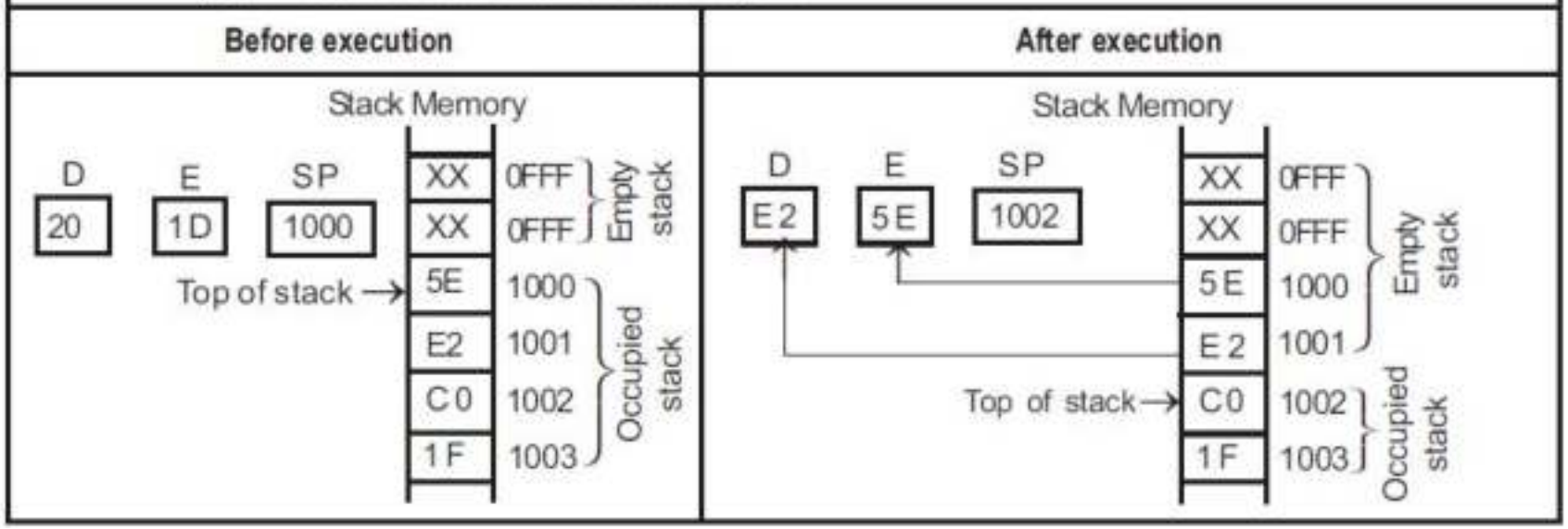
(i) The content of the memory addressed by the SP is moved to the E-register.

3T

(ii) The content of the SP is incremented by one.
(iii) The content of the memory addressed by the SP is moved to the D-register.

3T

(iv) The content of the SP is incremented by one.



-
-

Contd....

IN addr8 **(A)← (addr8)**

- The content of the port is moved to the A-register.
- The 8-bit port address will be given in the instruction.
- No flags are affected.
- Two byte instruction
- Three machine cycles:
  - Opcode fetch - 4T
  - Memory read - 3T
  - IO read - 3T

- Direct addressing
- Total number of instructions = 1

Data transfer instructions Contd….

OUT addr8 (addr8) ← (A)

- The content of the A-register is moved to the port.
- The 8-bit port address will be given in the instruction.
- No flags are affected.
- Two byte instruction
- Three machine cycles:
  - Opcode fetch - 4T
  - Memory read - 3T
  - IO write - 3T

- Direct addressing
- Total number of instructions = 1

[In an 8085 processor-based system when the IO devices are mapped by IO mapping then the processor can communicate with these IO devices only by using IN and OUT instructions. The processor uses an 8-bit address to select IO-mapped IO devices.]

Stack in 8085:

- The stack is a portion of RAM memory defined by the user for temporary storage and retrieval of data while executing a program.
- The microprocessor will have a dedicated internal register called Stack Pointer (SP) to hold the address of the stack.
- Also, the processor will have a facility to automatically decrement/increment the content of SP after every write/read operation into stack.
- The user can initialize or create a stack by loading a RAM address in the Stack Pointer (SP).
- Once an address is loaded in SP, the RAM memory locations below the address pointed by SP are reserved for stack. Typically 25 to 100 RAM memory locations are sufficient for stack.
- The user should take care that the reserved RAM memory locations for stack are not used for any other purpose.
- The user has to create/implement a stack whenever the program consists of PUSH, POP, RST n, CALL and RET instructions. Also, the stack is needed whenever the system uses interrupt facility.
- In a program, when the number of the available registers are not sufficient for storing intermediate result and data, then some of intermediate result and data can be stored in a stack using PUSH instruction and retrieved whenever required using POP instruction.
- The CALL instruction and the interrupts store the return address (content of program counter) in the stack before executing the subroutine. Usually the subroutines are terminated with RET instruction. When RET instruction is executed, the top of stack is poped to program counter and the program control returns to the main program after the execution of subroutine.

Stack in 8085 contd…

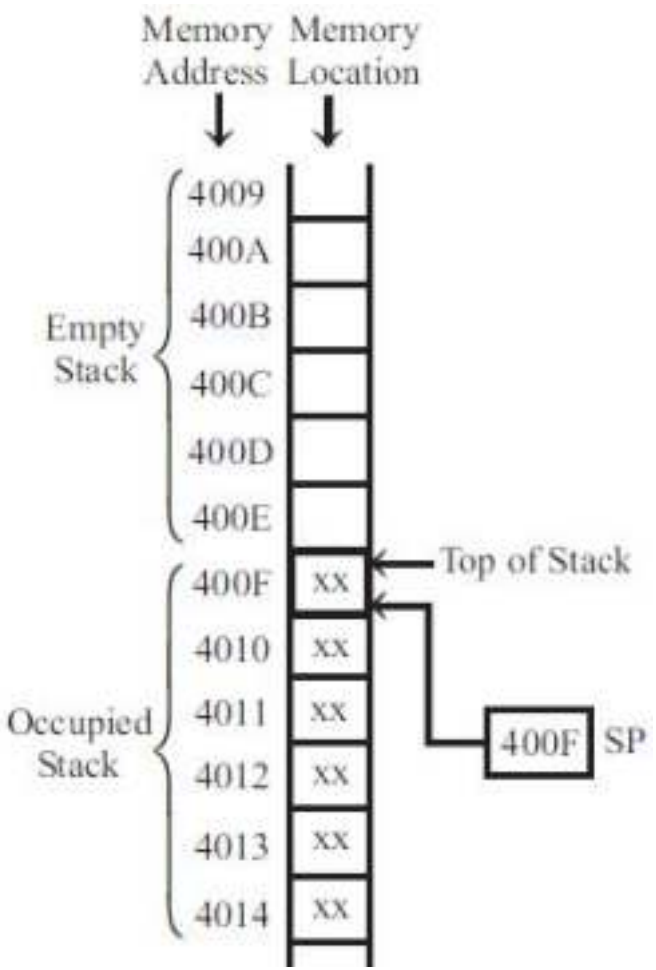


Fig. Example for Stack in 8085

- In an 8085 processor, for every write operation into stack, the SP is automatically decremented by two and for every read operation from stack, the SP is automatically incremented by two.
- Hence, data can be stored only in lower addresses from the address pointed by SP. Therefore, we can say that the SP holds the address of the top of stack.
- The storage and retrieval in stack are in reverse order, because the SP is decremented for every write operation into stack and SP is incremented for every read operation from stack. Therefore, the stack in 8085 is called **Last-In-First-Out (LIFO) stack,** i.e., the last stored information can be read first.

Thank You

MPES
Module 1_7

Arithmetic Instructions in 8085:

ADD reg (A) ← (A) + (reg)

- The content of the register is added to the content of the accumulator (A-register). After addition the result is stored in the accumulator.
- All flags are affected.
- The register can be any one of the general purpose register A, B, C, D, E, H or L.
- One byte instruction
- One machine cycle:
  - Opcode fetch - 4T
- Register addressing
-
- Total number of instructions = 7

ADD A ADD B ADD C ADD D ADD E ADD H ADD L

Example : **ADD E** **(A) ← (A) + (E)**

The content of the E-register is added to the content of theA-register.

The result will be in the A-register. All flags are affected.

Before execution	Addition	After execution
A: C2 E: B8 CF = 0 PF = 0 AF = 0 ZF = 0 SF = 0	$C2_H$ = 1100 0010 $B8_H$ = 1011 1000 1 0111 1010 Sum = 0111 1010 = $7A_H$ Carry = 1 (Addition is performed in ALU)	A: 7A E: B8 CF = 1 PF = 0 AF = 0 ZF = 0 SF = 0

ADI d8 (A) ← (A) + d8

- The 8-bit data given in the instruction is added to the content of the A-register (Accumulator). After addition, the result is stored in the accumulator.
- All flags are affected.
- Two byte instruction
- Two machine cycles:
  - Opcode fetch - 4T
  - Memory read - 3T
- Immediate addressing
- Total number of instructions = 1

ADD M (A) ← (A) + (M) or (A) ← (A) + ((HL))

- The content of memory addressed by HL pair is added to the content of the A-register. After addition, the result is stored in the A-register.
- All flags are affected.
- One byte instruction
- Two machine cycles:
  - Opcode fetch - 4T
  - Memory read - 3T
- Register indirect addressing
- Total number of instructions = 1

Example : **ADD M (A) ← (A) + (M) or (A) ← (A) + ((HL))**

Let the content of A be 44_H
Let the content of memory location $C00A_H$ be 73_H
The content of the memory location $C00A_H$ is added to the content of theA-register. The result is put back in the A-register.

Before execution	Addition	After execution
A: 44 HL: C00A Memory: 73 C00A, 14 C00B, 27 C00C CF = 0 PF = 0 AF = 0 ZF = 0 SF = 0	44_H = 0100 0100 73_H = 0111 0011 1011 0111 Sum = B7 Carry = 0 (Addition is performed in ALU)	A: B7 HL: C00A Memory: 73 C00A, 14 C00B, 27 C00C CF = 0 PF = 1 AF = 0 ZF = 0 SF = 1

ACI d8 (A) ← (A) + d8 + CF

- The 8-bit data given in the instruction and the carry flag (the value of carry flag before executing this instruction) are added to the content of the A-register (Accumulator). After addition, the result is stored in the accumulator.
- All flags are affected.
- Two byte instruction
- Two machine cycles :
  - Opcode fetch - 4T
  - Memory read - 3T
- Immediate addressing
- Total number of instructions = 1

ADC reg (A) ← (A) + (reg) + CF

- The content of the register and the carry flag are added to the content of the A-register. After addition, the result is stored in the A-register.
- All flags are affected.
- The register can be any one of the general purpose register A, B, C, D, E, H or L.
- One byte instruction
- One machine cycle :
  - Opcode fetch - 4T
- Register addressing

Example : **ADC H** **(A) ← (A) + (H) + CF**

The content of the H-register and the value of the carry flag (before executing this instruction) are added to the content of the A-register. After addition, the result will be in the A-register.

Before execution	Addition	After execution
A H 43 7A CF = 1 PF = 0 AF = 0 ZF = 0 SF = 1	43_H = 0100 0011 $7A_H$ = 0111 1010 CF = 1 1011 1110 Sum = BE_H Carry = 0 (Addition is performed in the ALU)	A H BE 7A CF = 0 PF = 1 AF = 0 ZF = 0 SF = 1

- Total number of instructions = 7

ADC A ADC B ADC C ADC D ADC E ADC H ADC L

ADC M (A) ← (A) + (M) + CF or (A) ← (A) + ((HL)) + CF

- The content of the memory addressed by the HL pair and the value of the carry flag (before executing this instruction) are added to the content of A-register. After addition, the result is stored in the A-register.
- All flags are affected.
- One byte instruction
- Two machine cycles:
  - Opcode fetch - 4T
  - Memory read - 3 T
- Register indirect addressing
- Total number of instructions = 1

INR reg (reg) ← (reg) + 01

- The content of the register is incremented by one. Except carry flag, all other flags are affected.
- The register can be any one of the general purpose register A, B, C, D, E, H or L.
- One byte instruction
- One machine cycle:
  - Opcode fetch - 4T
- Register addressing

Example : INR B (B) ← (B) + 01

The content of the B-register is incremented by one. The increment opertation is performed by adding 01_H to the content of B-register.

Before execution	Increment Operation	After execution
B: 4A CF = 0 PF = 0 AF = 0 ZF = 0 SF = 0	$4A_H$ = 0100 1010 $+01_H$ = 0000 0001 0100 1011 4 B	B: 4B CF = 0 PF = 1 AF = 0 ZF = 0 SF = 0

- Total number of instructions = 7

INR A INR B INR C INR D INR E INR H INR L

INR M (M) ← (M) + 01 or ((HL)) ← ((HL)) + 01

- The content of the memory addressed by the HL pair is incremented by one.
- Except carry, all other flags are affected.
- One byte instruction
- Three machine cycles :
  - Opcode fetch - 4T
  - Memory read - 3T
  - Memory write - 3T

Example : **INR M** **(M) ← (M) + 01**

Let the content of the HL pair be $C00A_H$. Let the content of memory location $C00A_H$ be $C5_H$. The content of the memory location $C00A_H$ is incremented by one. The increment operation is performed by adding 01_H to the content of the memory.

Before execution	Increment Operation	After execution
HL: C00A CF = 0 PF = 0 AF = 0 ZF = 0 SF = 0 Memory: C00A = C5, C00B = A2, C00C = 07	$C5_H$ = 1100 0101 $+ 01_H$ = 0000 0001 1100 0110 C 6	HL: C00A CF = 0 PF = 1 AF = 0 ZF = 0 SF = 1 Memory: C00A = C6, C00B = A2, C00C = 07

- Register indirect addressing
- Total number of instructions = 1

INX rp (rp) ← (rp) + 01

- The content of the register pair is incremented by one.
- The register pair can be BC, DE, HL or SP.
- No flags are affected.
- One byte instruction
- One machine cycle :
  - Opcode fetch - 6T
- Register addressing
- Total number of instructions = 4

INX B INX D INX H INX SP

Example : **INX H** **(HL) ← (HL) + 01**

The content of the HL pair is incremented by one.

Before execution	After execution
H L 00FF	H L 0100

DAD rp (HL) ← (HL) + (rp)

- (DAD - Double Addition)
- The content of the register pair is added to the content of the HL pair. After addition, the result is stored in the HL pair.
- Only the carry flag is affected.
- The register pair can be BC, DE, HL or SP.
- One byte instruction
- Three machine cycles:
  - Opcode fetch - 4T
  - Bus idle - 3T
  - Bus idle - 3T
- Register addressing
- Total number of instructions = 4

DAD B DAD D DAD H DAD SP

Thank You

MPES
Module 1_8

Arithmetic Instructions Contd….

SUB reg (A) ← (A) – (reg)

- The content of the register is subtracted from the content of the accumulator (A-register). After subtraction the result is stored in the A-register.
- All flags are affected.
- The register can be any one of the general purpose register A, B, C, D, E,H or L.

Example: **SUB C** **(A) ← (A) – (C)**

The content of the C-register is subtracted from A-register. The result will be in the A-register.

Case i

Before execution

A: C4 C: 89

CF = 0
PF = 0
AF = 0
ZF = 0
SF = 1

After execution

A: 3B C: 89

CF = 0
PF = 0
AF = 0
ZF = 0
SF = 1

Subtraction

$C4_H$ = 1100 0100 = 196 d

89_H = 1000 1001 = 137 d

1's complement of 89_H = 0111 0110

2's complement of 89_H = 0111 0110 + 1

= 0111 0111 = 77_H

$C4_H$ = 1100 0100

$+77_H$ = 0111 0111

1 0011 1011 = 59 d

Complement Carry ↓ 3 B

0

Result = $3B_H$

CF = 0

Contd....

- **One byte instruction**
- **One machine cycle:**
  Opcode fetch - 4T
- **Register addressing**
- **Total number of instructions = 7**

SUB A SUB B SUB C SUB D SUB E
SUB H SUB L

Case ii

Before execution	Subtraction
A: 89 C: C4 CF = 0 PF = 0 AF = 0 ZF = 1 SF = 1	89_H = 1000 1001 = 137 d $C4_H$ = 1100 0100 = 196 d 1's complement of $C4_H$ = 0011 1011 2's complement of $C4_H$ = 0011 1011 + 1 = 0011 1100 = $3C_H$

Case ii continued ...

After execution	Subtraction
A: C5 C: C4 CF = 1 PF = 1 AF = 1 ZF = 0 SF = 1	89_H = 1000 1001 +$3C_H$ = 0011 1100 0 1100 0101 = 197 d Complement Carry: 1 C 5 Result = $C5_H$ CF = 1 *Note : 2's complement of $C5_H$ = $3B_H$*

Note : The 8085 microprocessor performs 2's complement subtraction. But after subtraction, it will complement the carry alone. In 2's complement subtraction, if CF =1, then the result is positive and if CF =0, then the result is negative. Since, the 8085 processor complements the carry after subtraction, here if CF = 0, then the result is positive and if CF = 1, then the result is negative. If the result is negative, then it will be in 2's complement form.

Contd….

SUI d8 (A) ← (A) – d8

- The 8-bit data given in the instruction is subtracted from the A-register (accumulator). After subtraction, the result is stored in the A-register.
- All flags are affected.
- Two byte instruction
- Two machine cycles :
  - Opcode fetch - 4T
  - Memory read - 3T
- Immediate addressing
- Total number of instructions = 1

SUB M (A) ← (A) – (M) or (A) ← (A) – ((HL))

- The content of the memory addressed by the HL pair is subtracted from the A-register. After subtraction, the result is stored in the A-register.
- All flags are affected.
- One byte instruction
- Two machine cycles :
  - Opcode fetch - 4T
  - Memory read - 3T
- Register indirect addressing
- Total number of instructions = 1

Contd….

SBB reg (A) ← (A) – (reg) – CF

- The content of the register and the value of carry (before executing this instruction) are subtracted from the accumulator (A-register). After subtraction, the result is stored in the accumulator.
- All flags are affected.
- The register can be any one of the general purpose register A, B, C, D, E, H or L.
- One byte instruction
- One machine cycle :
  - Opcode fetch - 4T
- Register addressing
- Total number of instructions = 7

SBB A SBB B SBB C SBB D SBB E SBB H SBB L

Arithmetic Instructions in 8085 Contd…

SBI d8 (A) ← (A) – d8 – CF

- The 8-bit data given in the instruction and the value of carry (before executing this instruction) are subtracted from accumulator. After subtraction, the result is stored in the accumulator.
- All flags are affected.
- Two byte instruction
- Two machine cycles :
  - Opcode fetch - 4T
  - Memory read - 3T
- Immediate addressing
- Total number of instructions = 1

SBB M (A) ← (A) – (M) – CF or (A) ← (A) – ((HL)) – CF

- The content of the memory addressed by HL and the value of carry (before executing this instruction) are subtracted from accumulator (A-register). After subtraction, the result is stored in the A-register.
- All flags are affected .
- One byte instruction
- Two machine cycles:
  - Opcode fetch - 4T
  - Memory read - 3T
- Register indirect addressing
- Total number of instructions = 1

Contd….

DCR reg (reg) ← (reg) – 01

- The content of the register is decremented by one.
- Except carry, all other flags are affected.
- The register can be A, B, C, D, E, H or L.
- One byte instruction
- One machine cycle :
  - Opcode fetch - 4T
- Register addressing
- Total number of instructions = 7

DCR A DCR B DCR C DCR D DCR E DCR H DCR L

Example : DCR D ***(D) ← (D) – 01***

The content of the D-register is decremented by one. The decrement operation is performed by subtracting 01_H from the content of the D-register.

Before execution		Decrement operation
D [60]	CF = 0 PF = 0 AF = 0 ZF = 0 SF = 0	01_H = 0000 0001 1's complement of 01_H = 1111 1110 2's complement of 01_H = 1111 1110 + 1 = 1111 1111 = FF_H
After execution		
D [5F]	CF = 0 PF = 1 AF = 0 ZF = 0 SF = 0	60_H = 0110 0000 + FF_H = 1111 1111 1 0101 1111 5 F Carry is discarded

Contd…

DCR M (M) ← (M) – 01 or ((HL)) ← ((HL)) – 01

- The content of memory addressed by the HL pair is decremented by one.
- Except carry, all other flags are affected.
- One byte instruction
- Three machine cycles :
  - Opcode fetch - 4T
  - Memory read - 3T
  - Memory write - 3T
- Register indirect addressing
- Total number of instructions = 1

Example: **DCR M** **(M) ← (M) – 01**

Let the content of the HL pair be 2010_H. Let the content of memory location 2010_H be FA_H. The content of memory location 2010_H is decremented by one.

Before execution

HL: 2010

Memory: 2010 = FA; 2011 = 02

CF = 0
PF = 0
AF = 0
ZF = 0
SF = 0

After execution

HL: 2010

Memory: 2010 = F9; 2011 = 02

CF = 0
PF = 1
AF = 1
ZF = 0
SF = 1

Decrement operation

$01_H = 0000\ 0001$
1's complement of $01_H = 1111\ 1110$
2's complement of $01_H = 1111\ 1110 + 1$
$= 1111\ 1111 = FF_H$

$FA_H = 1111\ 1010$
$+FF_H = 1111\ 1111$
$1\ 1111\ 1001$
$F\ \ 9$

Carry is discarded

Contd....

DCX rp (rp) ← (rp) – 01

- The content of the register pair is decremented by one.
- The register pair can be BC, DE, HL or SP.
- No flags are affected.
- One byte instruction
- One machine cycle :
  - Opcode fetch - 6T
- Register addressing
- Total number of instructions = 4

DCX B DCX D DCX H DCX SP

Example : **DCX SP** **(SP) ← (SP) – 01**

The content of the stack pointer is decremented by one.

Before execution	After execution
SP 1000	SP 0FFF

Thank You

MPES
Module 1_9

BCD Numbers:

- Binary coded decimal
- Binary representation of decimal numbers 0 to 9 are called BCD.
- Unpacked and Packed BCD's are there.
- In Unpacked BCD, lower 4 bits (lower nibble) of one byte represents BCD number and rest of the bits are zero.
- In Packed BCD, a single byte has two BCD numbers in it. Lower 4 bits and upper 4 bits. Eg. 0101 1001 is packed BCD for 59 decimal.

Binary Coded Decimal BCD Numbers

Each decimal digit to be represented in BCD is converted into its 4 digit binary equivalent.

As the decimal digits relating to 1010, 1011, 1100, 1101, 1110, 1111 do not exist these are invalid.

Decimal	Binary	BCD
0	0000 0000	0000 0000
1	0000 0001	0000 0001
2	0000 0010	0000 0010
3	0000 0011	0000 0011
4	0000 0100	0000 0100
5	0000 0101	0000 0101
6	0000 0110	0000 0110
7	0000 0111	0000 0111
8	0000 1000	0000 1000
9	0000 1001	0000 1001
10	0000 1010	0001 0000
11	0000 1011	0001 0001
12	0000 1100	0001 0010
13	0000 1101	0001 0011
14	0000 1110	0001 0100
15	0000 1111	0001 0101
16	0001 0000	0001 0110
17	0001 0001	0001 0111
18	0001 0010	0001 1000
19	0001 0011	0001 1001
20	0001 0100	0010 0000

$10_D = 1010_B = A_H$ is not a valid BCD number as BCD is from 0 to 9 only.

To convert it to valid BCD, add 06_H to that.

1010 +

0110

1 0000 = 10

DAA

- (DAA - Decimal Adjust Accumulator)
- After BCD addition, the DAA instruction is executed to get the result in BCD. When DAA instruction is executed, the content of the accumulator is altered or adjusted as explained below :

  i) If the sum of the lower nibbles exceeds 09H or auxiliary carry is set, then a correction 06H (0110) is added to sum of lower nibbles.

  ii) If the sum of the upper nibbles exceeds 09H or carry is set, then a correction 06H (0110) is added to sum of upper nibble.
- After executing this instruction all flags are modified to indicate the status of the result.
- One byte instruction
- One machine cycle:
  - Opcode fetch - 4T
- Implied addressing
- Total number of instructions = 1

Eg. For BCD addition:

85 + 25 = 110 in actual BCD format.

But in binary, 1000 0101 +

0010 0101

1010 1010 is AA_H, which is not a valid BCD number.

To make the result in BCD, add 06 H to both upper and lower nibbles as they exceeds 09 decimal value.

Then,

1010 1010 +

0110 0110

1 0001 0000 = 110 which is the correct answer.

DAA instruction is used after the addition instruction.

Eg:

```
ORG 4000H
MOV A, 85H
MOV B, 25H
ADD B
DAA
HLT
```

Thank You

MPES
Module 1_10

Logical Instructions in 8085

ANA reg **(A) ← (A) & (reg)**

(& is the symbol used for logical AND operation)

- The content of the register is logically ANDed bit by bit with the content of the accumulator. In bit by bit AND operation, the bit D0 of register is ANDed with the bit D0 of A-register, the bit D1 of register is ANDed with bit D1 of A-register, and so on.
- The register can be any one of the general purpose register A, B, C, D, E, H or L. After execution of the instruction, carry flag is always reset and auxiliary carry flag is always set. Other flags are altered (according to the results).
- After AND operation, result is stored in accumulator.
- One byte instruction
- One machine cycle:
  - Opcode fetch - 4T
- Register addressing
- Total number of instructions = 7

ANA A ANA B ANA C ANA D

ANA E ANA H ANA L

Example : **ANA E** **(A) ← (A) & (E)**

The content of E-register is logically ANDed bit by bit with the content of accumulator.

Before execution	AND operation	After execution
A: 15 E: E2 CF = 0 PF = 0 AF = 0 ZF = 0 SF = 0	15_H = 0001 0101 $E2_H$ = 1110 0010 0000 0000 0 0	A: 00 E: E2 CF = 0 PF = 1 AF = 1 ZF = 1 SF = 0

ANI d8 (A) ← (A) & d8

- The 8-bit data given in the instruction is logically ANDed bit by bit with the content of the accumulator.
- The result is stored in the accumulator.
- After execution of this instruction, CF = 0 and AF = 1. Other flags are affected.
- Two byte instruction
- Two machine cycles :
  - Opcode fetch - 4T
  - Memory read - 3T
- Immediate addressing
- Total number of instructions = 1

ANA M (A) ← (A) & (M) or (A) ← (A) & ((HL))

- The content of the memory addressed by the HL pair is logically ANDed bit by bit with the content of the accumulator.
- The result is stored in the accumulator.
- After execution, CF = 0 and AF = 1. Other flags are affected .
- One byte instruction
- Two machine cycles:
  - Opcode fetch - 4T
  - Memory read - 3T
- Register indirect addressing
- Total number of instructions = 1

Example : **ANA M** **(A) ← (A) & (M)**

Let the content of HL be $105A_H$. Let the content of the memory location $105A_H$ be $4C_H$. The content of the memory location $105A_H$ is logically ANDed bit by bit with the content of the accumulator. The result is stored in the accumulator.

Before execution

A	HL	Memory	
27	105A	14	1059
		4C	105A

CF = 0
PF = 0
AF = 0
ZF = 0
SF = 0

AND operation

27_H = 0010 0111
$4C_H$ = 0100 1100
0000 0100
0 4

After execution

A	HL	Memory	
04	105A	14	1059
		4C	105A

CF = 0
PF = 0
AF = 1
ZF = 0
SF = 0

ORA reg (A) ← (A) | (reg)

(| is the symbol used for logical OR operation)

- The content of the register is logically ORed bit by bit with the content of the accumulator. In bit by bit OR operation, the bit D0 of the register is ORed with bit D0 of the A-register, the bit D1 of the register is ORed with bit D1 of the A register, and so on.
- The register can be any one of the general purpose register A, B, C, D, E, H or L.
- After execution of the instruction, both the carry and auxiliary flags are always reset (AF = 0, CF = 0). Other flags are modified (according to the result).
- After OR operation, the result is stored in the accumulator.
- One byte instruction
- One machine cycle:
  - Opcode fetch - 4T
- Register addressing
- Total number of instructions = 7

ORA A ORA B ORA C ORA D

ORA E ORA H ORA L

Example : **ORA B** **(A) ← (A) | (B)**

The content of the B-register is logically ORed bit by bit with the content of the accumulator.

Before execution	OR operation	After execution
A: 04 B: 7A CF = 0 PF = 0 AF = 0 ZF = 0 SF = 0	04_H = 0000 0100 $7A_H$ = 0111 1010 0111 1110 7 E	A: 7E B: 7A CF = 0 PF = 1 AF = 0 ZF = 0 SF = 0

ORA M (A) ← (A) | (M) or (A) ← (A) | ((HL))

- The content of the memory addressed by the HL pair is logically ORed bit by bit with the content of the accumulator.
- The result is stored in the accumulator.
- After execution, CF = AF = 0. Other flags are affected .
- One byte instruction
- Two machine cycles:
  - Opcode fetch - 4T
  - Memory read - 3T

Example : **ORA M** **(A) ← (A) | (M)**

Let the content of the HL pair be 2050_H. Let the content of memory location 2050_H be $1B_H$. The content of the memory location 2050_H is logically ORed bit by bit with the content of the accumulator. The result is stored in the accumulator.

Before execution

A	HL	Memory	
45	2050	1B	2050
		07	2051

CF = 0
PF = 0
AF = 0
ZF = 0
SF = 0

OR operation

45_H = 0100 0101

$1B_H$ = 0001 1011

0101 1111

5 F

After execution

A	HL	Memory	
5F	2050	1B	2050
		07	2051

CF = 0
PF = 1
AF = 0
ZF = 0
SF = 0

- Register indirect addressing
- Total number of instructions = 1

ORI d8 (A) ← (A) | d8

- The 8-bit data given in the instruction is logically ORed bit by bit with the content of the accumulator.
- The result is stored in the accumulator.
- After execution of this instruction, CF = AF = 0. Other flags are affected.
- Two byte instruction
- Two machine cycles :
  - Opcode fetch - 4T
  - Memory read - 3T
- Immediate addressing
- Total number of instructions = 1

XRA reg (A) ← (A) ^ (reg)

(^ is the symbol used for logical EXCLUSIVE-OR operation).

- The content of the register is logically EXCLUSIVE-ORed bit by bit with the content of the accumulator . In bit by bit EXCLUSIVE-OR operation, the bit D0 of register is EXCLUSIVE-ORed with bit D0 of A-register, the bit D1 of register is EXCLUSIVORed with bit D1 of A-register, and so on.
- The register can be any one of the general purpose register A, B, C, D, E, H or L.
- After execution AF = CF = 0. Other flags are modified (according to the result).
- The result is stored in the accumulator.
- One byte instruction
- One machine cycle:
  - Opcode fetch - 4T
- Register addressing
- Total number of instructions = 7

Example : **XRA A** **(A) ← (A) ^ (A)**

The content of the A-register is EXCLUSIVE-ORed bit by bit with the content of the A-register itself.

Before execution	EXCLUSIVE-OR operation	After execution
A 74 CF = 1, PF = 0, AF = 1, ZF = 0, SF = 1	74_H = 0111 0100 74_H = 0111 0100 0000 0000	A 00 CF = 0, PF = 1, AF = 0, ZF = 1, SF = 0

XRA A XRA B XRA C XRA D XRA E XRA H XRA L

XRI d8 (A) ← (A) ^ d8 or (A) ← (A) ^ d8

- The 8-bit data given in the instruction is logically EXCLUSIVE-ORed bit by bit with the content of the accumulator.
- The result is stored in the accumulator.
- After execution of this instruction, CF = AF = 0. Other flags are affected.
- Two byte instruction
- Two machine cycles :
  - Opcode fetch - 4T
  - Memory read - 3T
- Immediate addressing
- Total number of instructions = 1

XRA M (A) ← (A) ^ (M) or (A) ← (A) ^ ((HL))

- The content of the memory addressed by the HL pair is logically EXCLUSIVE-ORed bit by bit with the content of accumulator.
- The result is stored in accumulator.
- After execution, CF = AF = 0. Other flags are affected.
- One byte instruction
- Two machine cycles :
  - Opcode fetch - 4T
  - Memory read - 3T
- Register indirect addressing
- Total number of instructions = 1

Example : **XRA M** **(A) ← (A) ^ (M)**

Let the content of the HL pair be $805A_H$. Let the content of memory location $805A_H$ be $C4_H$. The content of the memory location $805A_H$ is logically EXCLUSIVE-ORed bit by bit with the content of the accumulator. The result will be in the accumulator.

Before execution	EXCLUSIVE-OR operation	After execution
A: B7 HL: 805A CF = 1 PF = 1 AF = 1 ZF = 0 SF = 1 Memory: 8059: 1C; 805A: C4; 805B: 20; 805C: 51	$B7_H$ = 1011 0111 $C4_H$ = 1100 0100 0111 0011 7 3	A: 73 HL: 805A CF = 0 PF = 0 AF = 0 ZF = 0 SF = 0 Memory: 8059: 1C; 805A: C4; 805B: 20; 805C: 51

Thank you

MPES
Module 1_11

Logical instructions in 8085 contd….

CMP reg **(A) – (reg) ⇒ Modify flags**

- The content of the register is compared with the accumulator. The comparison is performed by subtracting the content of register from the A-register. The subtraction is performed in the ALU, and the result is used to modify flags and then the result is discarded (i.e., it is not stored in any register). After execution of this instruction, the content of accumulator and the register are not altered.
- All flags are affected by this instruction.
- The register can be any one of the general purpose register A, B, C, D, E, H or L.
- The status of carry and zero flag after comparison are given below :
  - i) If (A) < (reg) then the carry flag is set (i.e., CF = 1)
  - ii) If (A) > (reg) then the carry flag is reset or cleared (i.e., CF = 0)
  - iii) If (A) = (reg) then the zero flag is set (i.e., ZF = 1)
- One byte instruction
- One machine cycle:
  - Opcode fetch - 4T
- Register addressing
- Total number of instructions = 7

  CMP A CMP B CMP C CMP D CMP E CMP H CMP L

Contd....

Example : **CMP B** **(A) – (B) ⇒ Modify flags.**

The content of the B-register is compared with the accumulator. The comparison is performed by subtracting the content of the B-register from the content of the accumulator. The subtraction is performed in the ALU and the result is used to modify the flags and then discarded. The content of the accumulator and the B-register are not altered.

Before execution	**Comparison**	**After execution**
A B 15 C2 CF = 0 PF = 0 AF = 0 ZF = 0 SF = 0	$C2_H$ = 1100 0010 1's complement of $C2_H$ = 0011 1101 2's complement of $C2_H$ = 0011 1101+1 = 0011 1110 = $3E_H$ 15_H = 0001 0101 $+3E_H$ = 0011 1110 0 0101 0011 Complement Carry ↓ 5 3 1	A B 15 C2 CF = 1 PF = 1 AF = 1 ZF = 0 SF = 0

CPI d8 (A) – d8 ⇒ Modify flags.

- The 8-bit data given in the instruction is compared with the accumulator. The comparison is performed by subtracting the 8-bit data from the A-register. The subtraction is performed in ALU and the result is used to modify flags and then discarded. After execution of the instruction, the content of the accumulator is not altered.
- All flags are affected.
- The status of carry and zero flag after comparision are given below :
  - i) If (A) < d8 then the carry flag is set (i.e., CF = 1)
  - ii) If (A) > d8 then the carry flag is reset or cleared (i.e., CF = 0)
  - iii) If (A) = d8 then the zero flag is set (i.e., ZF = 1).
- Two byte instruction
- Two machine cycles :
  - Opcode fetch - 4T
  - Memory read - 3T
- Immediate addressing
- Total number of instructions = 1

CMP M (A) – (M) ⇒ Modify flags or (A) – ((HL)) ⇒ Modify flags.

- The content of the memory addressed by HL pair is compared with the accumulator. The comparison is performed by subtracting the content of memory from the A-register. The subtraction is performed in the ALU and the result is used to modify flags and then discarded. After execution of the instruction, the content of the accumulator and the memory are not altered.
- All flags are affected by this instruction.
- The status of carry and zero flag after comparison are given below:
  - i) If (A) < (M) then the carry flag is set (i.e., CF = 1).
  - ii) If (A) > (M) then the carry flag is reset or cleared (i.e., CF = 0).
  - iii) If (A) = (M) then the zero flag is set (i.e., ZF = 1).
- One byte instruction
- Two machine cycles:
  - Opcode fetch - 4T
  - Memory read - 3T
- Register indirect addressing
- Total number of instructions = 1

Contd....

Example : **CMP M**

Let the content of the HL pair be $C050_H$. Let the content of the memory location $C050_H$ be $7A_H$. The content of the memory location $C050_H$ is compared with the content of the accumulator. Only flags are altered. The content of the accumulator and the memory remains the same.

Before execution	**Comparison**	**After execution**
A HL 25 C050 Memory 7A C050 10 C051 CF = 0 PF = 0 AF = 0 ZF = 0 SF = 0	25_H = 0010 0101 $7A_H$ = 0111 1010 1'complement of $7A_H$ = 1000 0101 2'complement of $7A_H$ = 1000 0101 +1 = 1000 0110 = 86_H 25_H = 0010 0101 $+86_H$ = 1000 0110 0 1010 1011 Complement Carry → 1 A B	A HL 25 C050 Memory 7A C050 10 C051 CF = 1 PF = 0 AF = 0 ZF = 0 SF = 1

CMA $(A) \leftarrow (\bar{A})$

- (CMA - Complement Accumulator)
- The content of the accumulator is complemented.
- No flags are affected.
- One byte Instruction
- One machine cycle:
  - Opcode fetch - 4T
- Implied addressing

STC $(CF) \leftarrow 1$

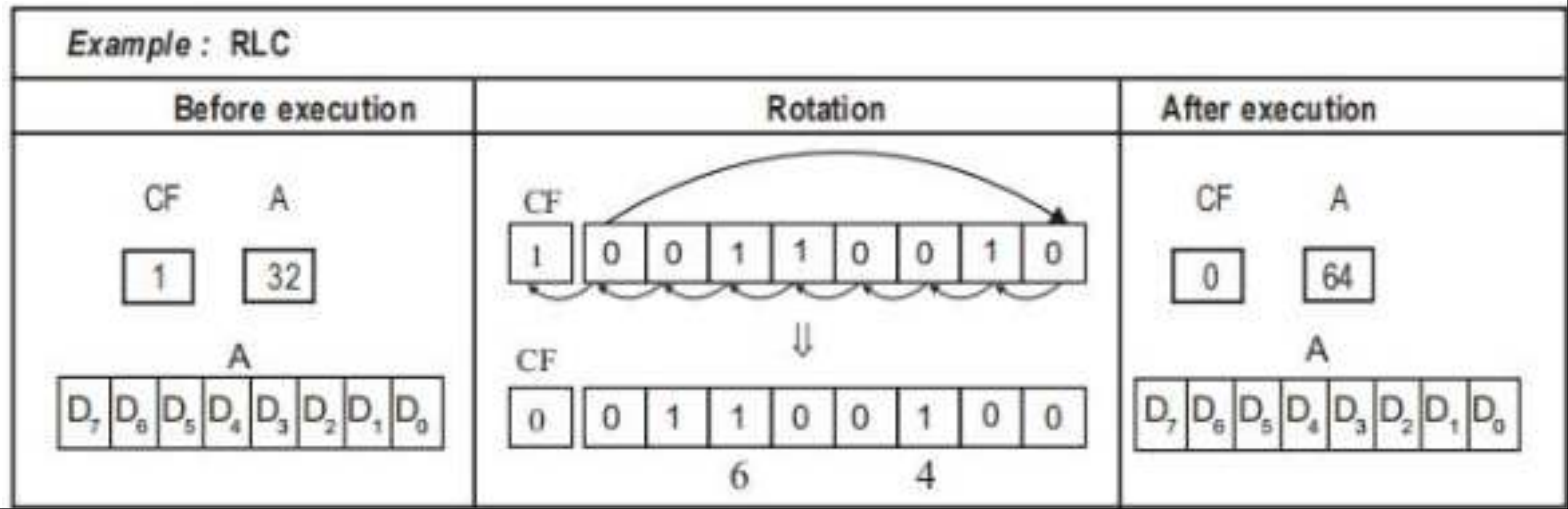
- (STC - Set Carry)
- The carry flag is set to 1.
- Only carry flag is affected by this instruction.
- One byte instruction
- One machine cycle :
  - Opcode fetch - 4T
- Implied addressing

CMC $(CF) \leftarrow (\overline{CF})$

- (CMC - Complement Carry)
- The carry flag is complemented. Only the carry flag is affected by this instruction.
- One byte instruction
- One machine cycle:
  - Opcode fetch - 4T
- Implied addressing

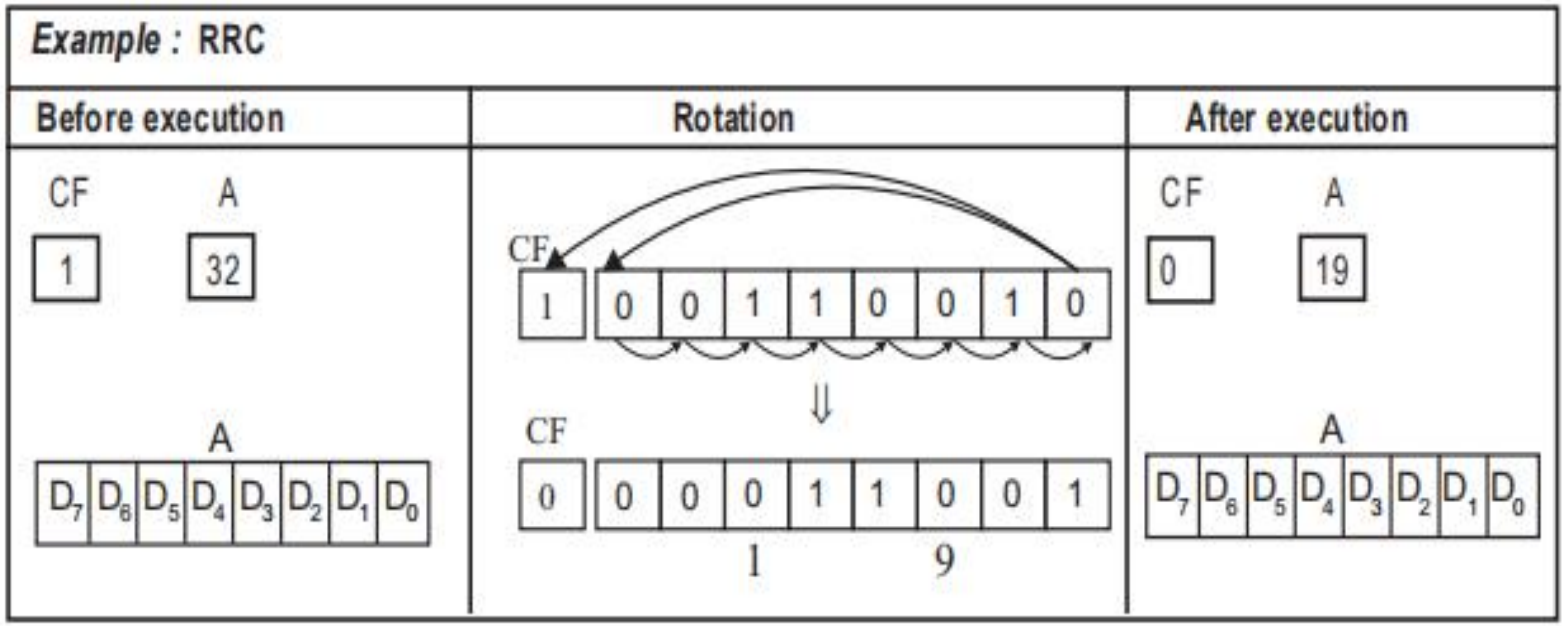
RLC $Dn + 1 \leftarrow Dn$; $D0 \leftarrow D7$ and $(CF) \leftarrow D7$

- (RLC - Rotate Accumulator Left to carry)
- The content of the A-register is rotated left by one bit and the left most bit of A-register is rotated to the carry. [The left most bit is most significant bit.]
- Only the carry flag is affected.
- One byte instruction
- One machine cycle:
  - Opcode fetch - 4T
- Implied addressing



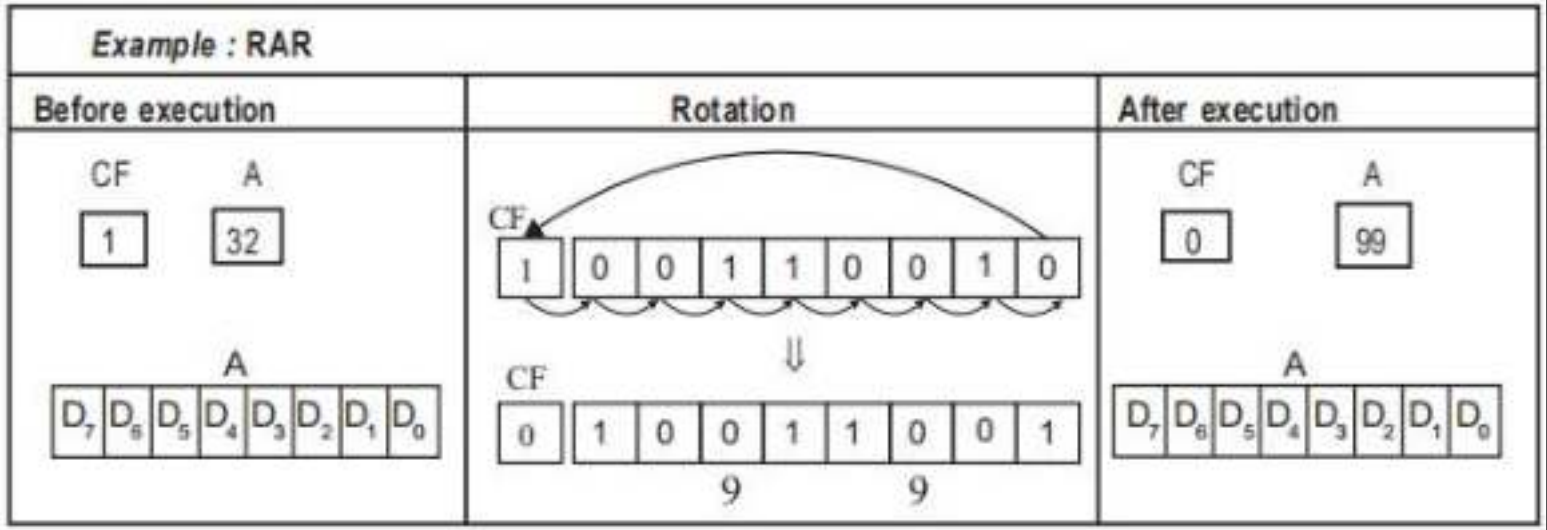
RRC Dn ← Dn+1 ; D7 ← D0 and (CF) ← D0

- (RRC - Rotate Accumulator Right to Carry)
- The content of A-register is rotated right by one bit and the right most bit of A-register is rotated to carry. [The right most bit is least significant bit.]
- Only carry flag is affected.
- One byte instruction
- One machine cycle:
  - Opcode fetch - 4T
- Implied addressing



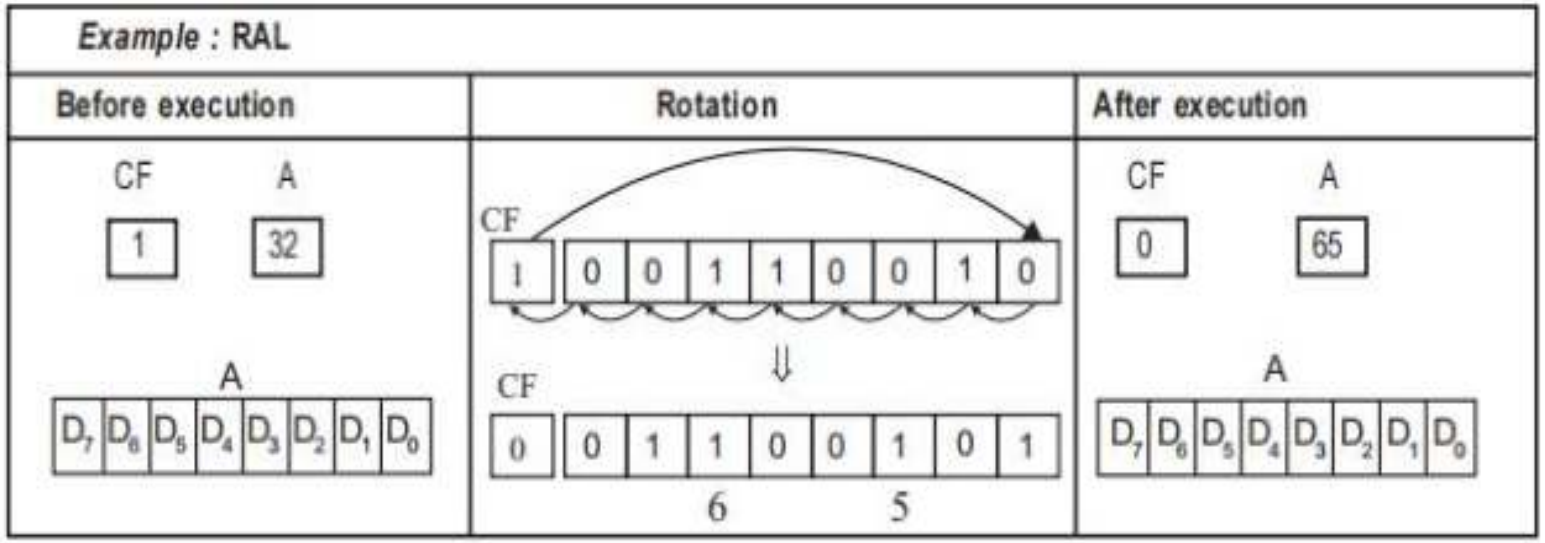
RAR Dn ← Dn+1 ; D7 ← (CF) and (CF) ← D0

- (RAR - Rotate Accumulator Right through carry)
- The content of the A-register along with the carry is rotated right by one bit. Here the carry is moved to the most significant bit position (D7) and the least significant bit (D0) is moved to the carry.
- Only the carry flag is affected.
- One byte instruction
- One machine cycle:
  - Opcode fetch - 4T
- Implied addressing



RAL Dn +1 ← Dn ; D0 ← (CF) and (CF) ← D7

- (RAL - Rotate Accumulator Left through carry)
- The content of the A-register along with the carry is rotated left by one bit. Here the carry is moved to the least significant bit position (D0) and the most significant bit (D7) is moved to the carry.
- Only the carry flag is affected.
- One byte instruction
- One machine cycle:
  - Opcode fetch - 4T
- Implied addressing



Thank You

MPES
Module 1_12

Branching Instructions in 8085:

- The control transfer instructions include Unconditional Branch, Jump and Jump-to-subroutine. The Branch instruction uses relative addressing while the Jump instruction uses direct or indirect addressing.
- Sub routine is a program other than the main program which executes a specific task , which can be used in main programs wherever that particular task should be performed by the help of Branching instructions or interrupts.

JMP addr16 (PC) ← addr16

- It is unconditional jump instruction. When this instruction is executed, the address given in the instruction is moved to the program counter. Now, the processor starts executing the instructions stored from this address.
- Three byte instruction
- Three machine cycles:
  - Opcode fetch - 4 T
  - Memory read - 3 T
  - Memory read - 3 T
- Immediate addressing

J <condition> addr16

- If <condition> is TRUE then, (PC) ← addr16
- It is conditional jump instruction. The conditional jump instruction will check a flag condition. If the flag condition is true, then the address given in the instruction is moved to the program counter. Thus the program control is branched to the jump address. If the flag condition is false, then the next instruction is executed.
- There are eight conditional jump instructions.
  - **JZ addr16 ;** **Jump on Zero - Jump if zero flag = 1.**
  - **JNZ addr16 ;** **Jump on Not Zero - Jump if zero flag = 0.**
  - **JC addr16 ;** **Jump on Carry - Jump if carry flag = 1.**
  - **JNC addr16 ;** **Jump on No Carry - Jump if carry flag = 0.**
  - **JM addr16 ;** **Jump on Minus - Jump if sign flag = 1.**
  - **JP addr16 ;** **Jump on Positive - Jump if sign flag = 0.**
  - **JPE addr16 ;** **Jump on Parity Even - Jump if parity flag = 1.**
  - **JPO addr16 ;** **Jump on Parity Odd - Jump if parity flag = 0.**
- Three byte instruction
- Two or three machine cycles:

Condition False	Condition True
Opcode fetch - 4T	Opcode fetch - 4T
Memory read - 3T	Memory read - 3T
	Memory read - 3T
7T	10T

CALL addr16

(SP) ← (SP) – 1 ; ((SP)) ← (PC)H
(SP) ← (SP) – 1 ; ((SP)) ← (PC)L
(PC) ← addr16

- It is unconditional CALL used to call a subroutine program. When this instruction is executed, the address of the next instruction in the program counter is pushed to the stack. The 16-bit address (which is the address of the subroutine program) given in the instruction is loaded in the program counter. Now, the processor will start executing the instructions stored in this call address .
- Three byte instruction
- Five machine cycles:
  - Opcode fetch - 6T
  - Memory read - 3T
  - Memory read - 3T
  - Memory write - 3T
  - Memory write - 3T
- Immediate addressing

C<condition> addr16

- If <condition> is TRUE then,

  (SP) ← (SP) – 1 ; ((SP)) ← (PC)H

  (SP) ← (SP) – 1 ; ((SP)) ← (PC)L

  (PC) ← addr16

- It is conditional subroutine call instruction. The conditional CALL instruction will check for a flag condition. If the flag condition is true, then the address of the next instruction is pushed to the stack and the call address (address given in the instruction) is loaded in the program counter. Now, the processor will start executing the instructions stored in this address. If the flag condition is false, then the next instruction is executed.
- There are eight conditional CALL instructions. These are:
  - **CZ addr16 ; Call on Zero - Call if zero flag = 1.**
  - **CNZ addr16 ; Call on Not Zero - Call if zero flag = 0.**
  - **CC addr16 ; Call on Carry - Call if carry flag = 1.**
  - **CNC addr16 ; Call on No Carry - Call if carry flag = 0.**
  - **CM addr16 ; Call on Minus - Call if sign flag = 1.**
  - **CP addr16 ; Call on Positive - Call if sign flag = 0.**
  - **CPE addr16 ; Call on Parity Even - Call if parity flag = 1.**
  - **CPO addr16 ; Call on Parity Odd - Call if parity flag = 0.**
- Immediate addressing
- Three byte instruction

Two or five machine cycles:	*Condition False*		*Condition True*	
	Opcode fetch	*6T*	*Opcode fetch*	*6T*
	Memory read	*3T*	*Memory read*	*3T*
		9T	*Memory read*	*3T*
			Memory write	*3T*
			Memory write	*3T*
				18T

RET **(PC)L ← ((SP)) ; (SP) ← (SP) + 1**

(PC)H ← ((SP)) ; (SP) ← (SP) + 1

- (RET - Return to the main program)
- It is an unconditional return instruction. This instruction is placed at the end of the subroutine program, in order to return to the main program. When this instruction is executed, the top of the stack is poped to (loaded in) the program counter .

Note : While calling the subroutine using CALL instruction, the return address of the main program is pushed to the stack. The return instruction, (RET) pops that to the program counter. Thus the processor resumes the execution of main program.

- One byte instruction
- Three machine cycles:
  - Opcode fetch - 4 T
  - Memory read - 3 T
  - Memory read - 3 T

Register indirect addressing

R<condition>

- If <condition> is TRUE then,

  (PC)L ← ((SP)) ; (SP) ← (SP) + 1

  (PC)H ← ((SP)) ; (SP) ← (SP) + 1
- It is conditional return instruction.
- In a conditional return instruction a flag condition is tested. If the flag condition is true, then the program control return to main program by poping the top of the stack to the program counter. If the flag condition is false, then the next instruction is executed.
- There are eight conditional return instructions:
  - **RZ ; Return on Zero - Return if zero flag = 1.**
  - **RNZ ; Return on Not Zero - Return if zero flag = 0.**
  - **RC ; Return on Carry - Return if carry flag = 1.**
  - **RNC ; Return on No Carry - Return if carry flag = 0.**
  - **RM ; Return on Minus - Return if sign flag = 1.**
  - **RP ; Return on Positive - Return if sign flag = 0.**
  - **RPE ; Return on Parity Even - Return if parity flag = 1.**
  - **RPO ; Return on Parity Odd - Return if parity flag = 0.**
- One byte instruction
- Register indirect addressing

One or three machine cycles:	Condition False	Condition True	
	Opcode fetch	- 6T	Opcode fetch - 6T
			Memory read - 3T
			Memory read - 3T

RST n

- It is a restart instruction. The restart instructions are also called software interrupts. Each restart instruction has a vector address. The vector address is fixed by the manufacturer (INTEL).
- When a restart instruction is executed, the content of the program counter is pushed to the stack and the vector address is loaded in the program counter. The vector address is internally generated (computed) by the processor. The vector address for RST n is obtained by multiplying n by 8. Thus the program control is branched to a subroutine program stored in this vector address.
- One byte instruction
- Register indirect addressing
- Three machine cycles:
  - Opcode fetch - 6 T
  - Memory write - 3 T
  - Memory write - 3 T
- There are eight restart instructions.

Restart instruction	Vector address	Computation of vector address
RST 0	0000_H	$0 \times 8 = 0_{10} = 0_H$
RST 1	0008_H	$1 \times 8 = 8_{10} = 8_H$
RST 2	0010_H	$2 \times 8 = 16_{10} = 10_H$
RST 3	0018_H	$3 \times 8 = 24_{10} = 18_H$
RST 4	0020_H	$4 \times 8 = 32_{10} = 20_H$
RST 5	0028_H	$5 \times 8 = 40_{10} = 28_H$
RST 6	0030_H	$6 \times 8 = 48_{10} = 30_H$
RST 7	0038_H	$7 \times 8 = 56_{10} = 38_H$

RST 0 RST 1 RST 2 RST 3 RST 4 RST 5 RST 6 RST 7

PCHL (PC) ← (HL)

- The content of the HL register pair is moved to the program counter. Since this instruction alters the content of the program counter, the program control is transferred to a new address. This instruction is used by the system designer to implement the system subroutine to execute a program.
- One byte instruction
- One machine cycle:
  - Opcode fetch - 6T
- Implied addressing

Thank You

MPES
Module 1_13

Machine control Instructions in 8085:

DI

- (DI - Disable Interrupts)
- When this instruction is executed, all the interrupts except TRAP are disabled. [When the interrupts are disabled the processor will not accept or recognize the interrupt request made by the external devices through the interrupt pins.]
- When the processor is doing an emergency work, it can execute DI instruction to prevent the interrupts from interrupting the processor.
- One byte instruction
- One machine cycle:
  - Opcode fetch - 4T

EI

- (EI - Enable Interrupts)
- This instruction is used (or executed) to allow the interrupts after disabling. (The interrupts except TRAP are disabled after processor reset or after execution of DI instruction. When we want to allow the interrupts, we have to execute EI instructions.)
- One byte instruction
- One machine cycle:
  - Opcode fetch - 4T

SIM

- (SIM - Set Interrupt Mask)
- The SIM instruction is used to mask the hardware interrupts RST 7.5, RST 6.5 and RST 5.5. It is also used to send data through the SOD line. (SOD: Serial Output Data pin of the 8085 processor.) The execution of SIM instruction uses the content of the accumulator to perform the following functions:
  - i) Program the interrupt mask for the hardware interrupts RST 5.5, RST 6.5 and RST 7.5.
  - ii) Reset the edge-triggered RST 7.5 input latch.
  - iii) Load the SOD output latch.
- One byte Instruction
- One machine cycle:
  - Opcode fetch - 4T

[If the mask set enable bit is set to "1" then the interrupt mask bits for RST 7.5, RST 6.5 and RST 5.5 (D0 , D1 and D2) are recognized and if it is "0" then these bits are not recognized by the processor. The interrupt mask bits D0 , D1 and D2 can be independently set to "1" to mask the particular interrupt and reset to "0" to unmask the particular interrupt. If the bit D4 isset to "1", then an internal flip-flop is reset to "0" in order to disable the RST 7.5 interrupt. If the serial output enable is "1", the serial output data is sent to the SOD pin.]

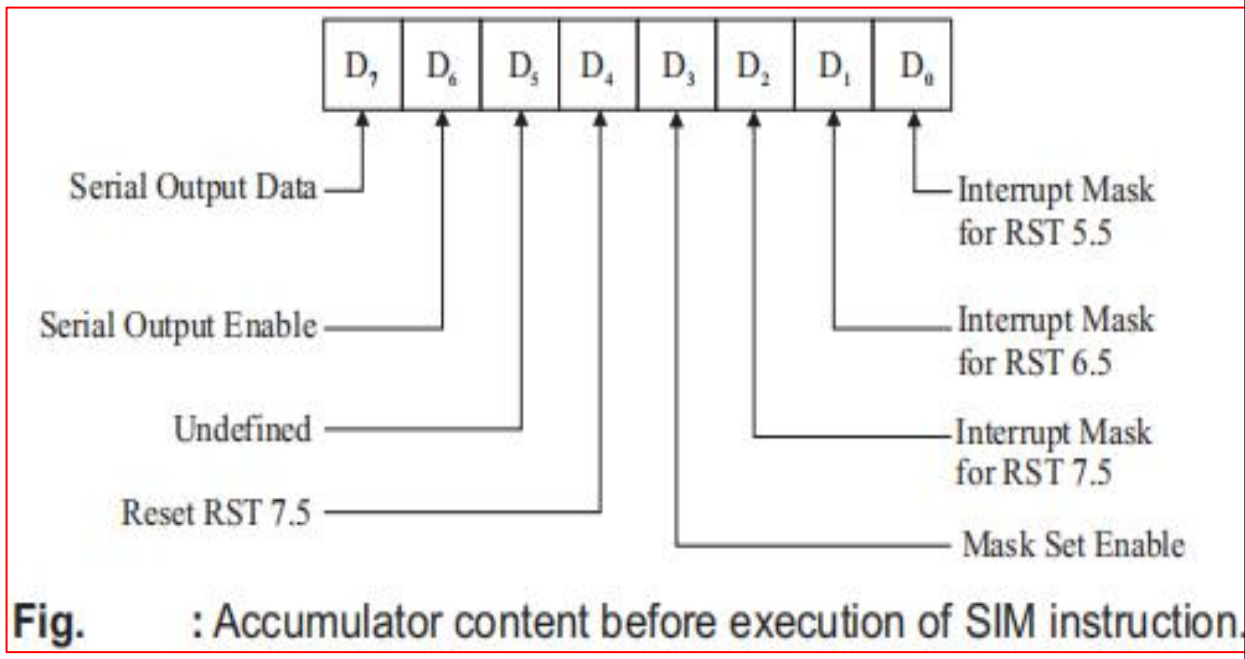


Fig. : Accumulator content before execution of SIM instruction.

Example program:

```
EI ;            Enable all interrupts of 8085
MVI A,0BH ;     Move 0BH to A-register
SIM ;           Mask 6.5 and 5.5, Enable 7.5
```

RIM

- (RIM - Read Interrupt Mask)
- The RIM instruction is used to check whether an interrupt is masked or not. It is also used to read data from the SID line. (SID: Serial Input Data pin of 8085 processor).
- When a RIM instruction is executed, the accumulator is loaded with 8-bit data. The 8-bit data in the accumulator (content of accumulator) can be interpreted as shown in Fig.
- Bits D0 , D1 and D2 provide the mask status of the RST 5.5, RST 6.5 and RST 7.5 interrupts respectively. If the mask bit corresponding to a particular RST is "1", then the interrupt is masked and if the mask bit is "0" then the interrupt is unmasked.
- If the interrupt enable bit (D3) is "0", the 8085's maskable interrupts are disabled. The interrupts are enabled if this bit is "1".

[A "1" in a particular interrupt pending bit indicates that an interrupt is being requested on the identified RST line. When this bit is "0", no interrupt is waiting to be serviced. The serial input data (bit D7) indicate the value of the signal at the SID pin.]

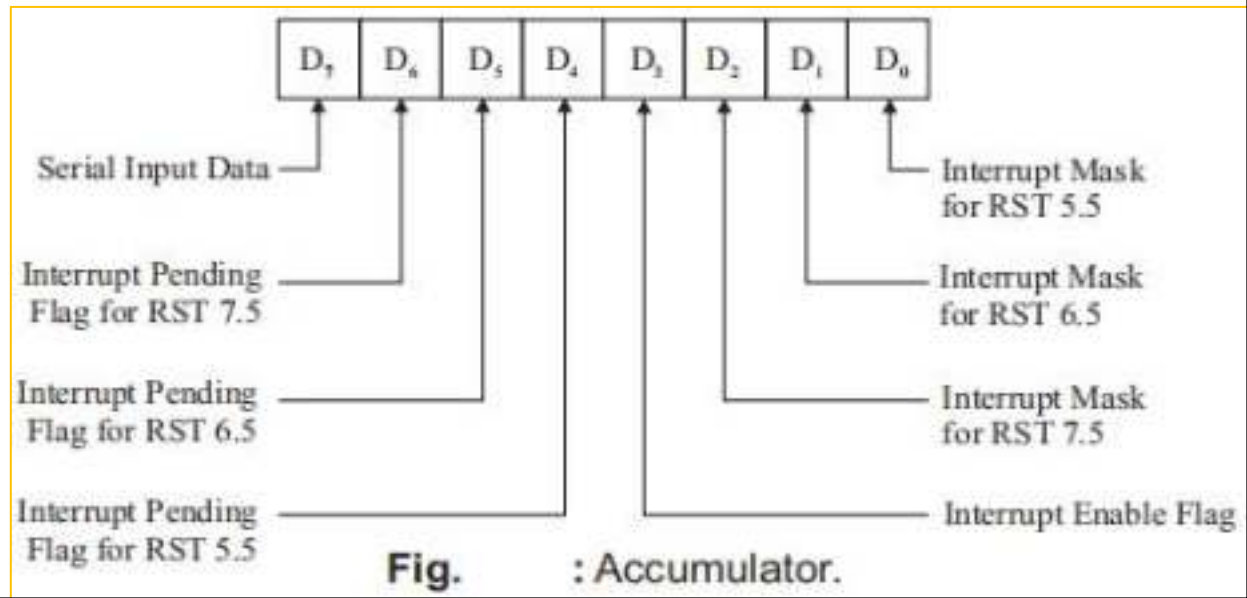


Fig. : Accumulator.

- One byte instruction
- One machine cycle:
  - Opcode fetch - 4T

HLT

- (HLT - Halt program Execution)
- This instruction is placed at the end of the program. When this instruction is executed, the processor suspends program execution and bus will be in idle state.
- One byte instruction
- Two machine cycle:
  - Opcode fetch - 3T
  - Bus idle - 2T

NOP

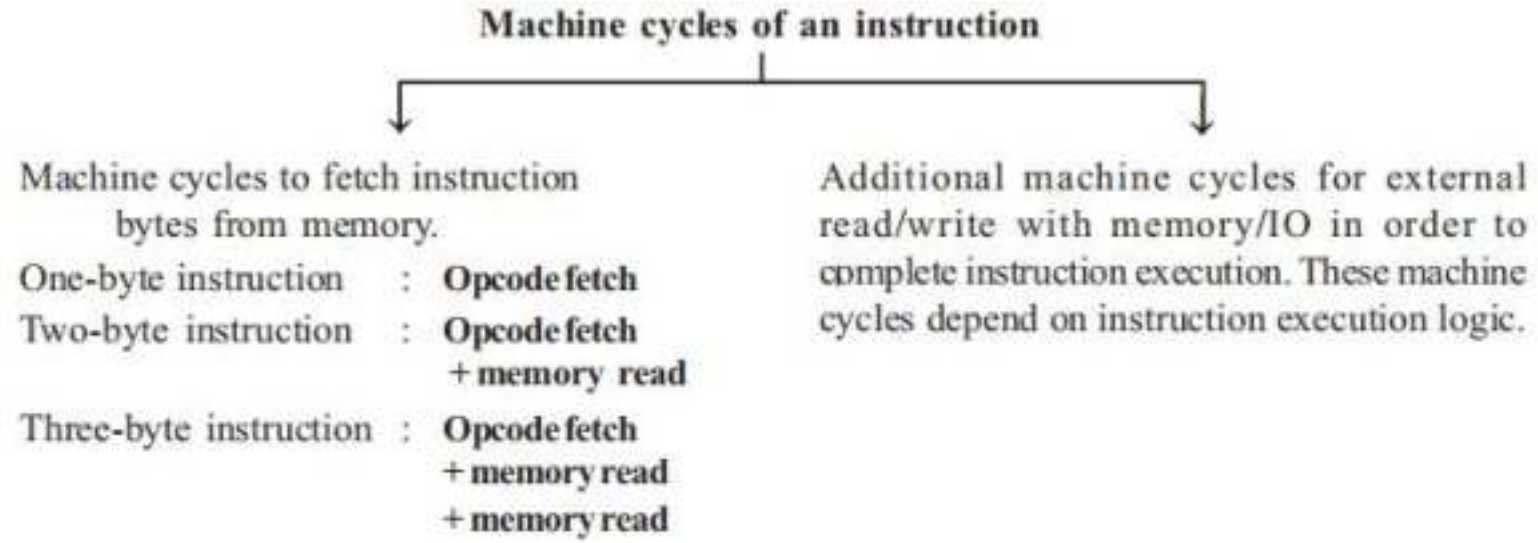
- (NOP - No operation)
- The NOP is a dummy instruction, it neither achieves any result nor affects any CPU registers. This is an useful instruction for producing software delay and reserve memory spaces for future software modifications.
- One byte instruction
- One machine cycle :
  - Opcode fetch - 4T

Thank You

MPES
Module 1_14

Timing Diagram of 8085 Instructions:

- The execution of an instruction is the execution of the machine cycles of that instruction in a predefined order.
- Therefore, from the knowledge of the timing diagrams of machine cycles, the timing diagram of an instruction can be obtained.



Example: STA 1250H etc….

The sequence of operations that a processor has to carry out while executing an instruction is called **Instruction cycle**.

Each instruction cycle of a processor in turn consists of a number of machine cycles.

The time required to access the memory or input/output devices is called **Machine cycle**.

To execute an instruction, the processor will run one or more machine cycles in a particular order.

The seven Machine Cycle in 8085 Microprocessor are :

- **Opcode Fetch Cycle**
- **Memory Read**
- **Memory Write**
- **I/O Read**
- **I/O Write**
- **Interrupt Acknowledge**
- **Bus Idle**

T-State:

The T-state is the time period of the internal clock signal of the processor.

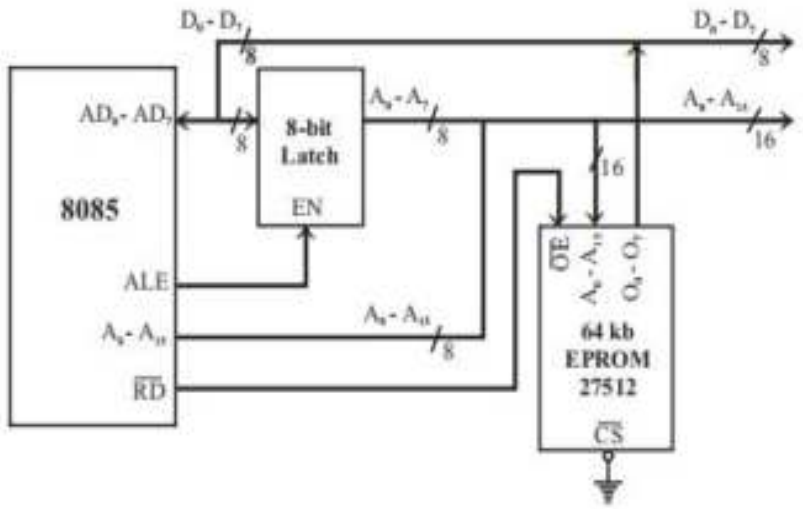


Fig. : Example of implementing 64 kb EPROM in the 8085 system.

TABLE - BUS STATUS SIGNALS

IO/$\overline{M}$	S_1	S_0	Operation performed by 8085
0	0	1	Memory write
0	1	0	Memory read
1	0	1	IO write
1	1	0	IO read
0	1	1	Opcode fetch
1	1	1	Interrupt acknowledge

Timing Diagram of Opcode Fetch Machine Cycle in 8085:

- Each instruction of the processor has one-byte opcode. The opcodes are stored in memory.
- The opcode fetch machine cycle is executed by the processor to fetch the opcode from memory.
- Hence, every instruction starts with opcode fetch machine cycle.
- The time taken by the processor to execute the opcode fetch cycle is either 4T or 6T. In this time, the first 3T states are used for fetching the opcode from memory and the remaining T states are used for internal operations by the processor.

Contd…

- At the falling edge of first T-state (T1), the microprocessor outputs the low byte address on AD0 -AD7 lines and high byte address on A8 to A15 lines. ALE is asserted high to enable the external address latch. The other control signals are asserted as follows.
- IO/M=0, S0 = 1, S1 = 1. (IO/M is asserted low to indicate memory access.)
- At the middle of T1 , the ALE is asserted low and this enables the external address latch to take low byte of the address and keep on its output lines.
- In the second T-state (T2), the memory is requested for read by asserting read line low. When read is asserted low, the memory is enabled for placing the opcode on the data bus. The time allowed for memory to output the opcode is the time during which read remains low.
- In the third T-state (T3), the read signal is asserted high. On the rising edge of read signal, the opcode is latched into microprocessor. Other control signals remain in the same state until the next machine cycle.
- The fourth T-state (T4) is used by the processor for internal operations to decode the instruction and encode into various machine cycles, and also for completing the task specified by 1-byte instruction. During this state (T4) the address and data bus will be in high impedance state.

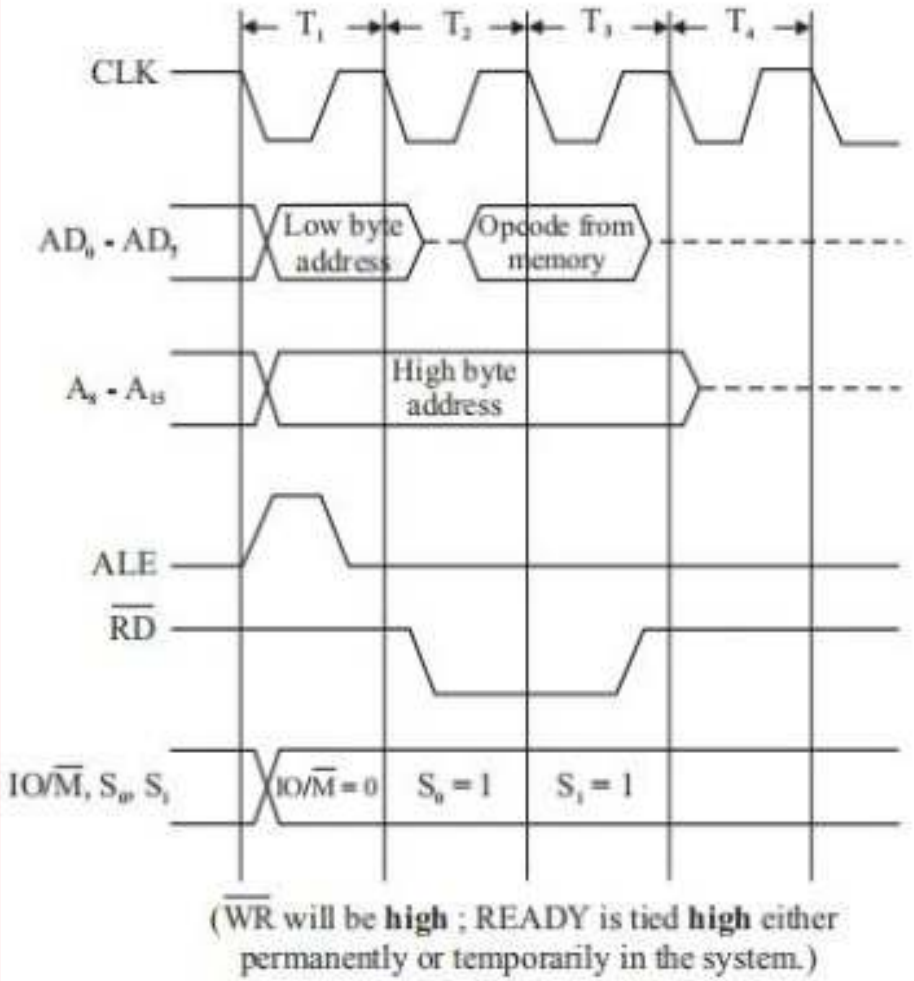
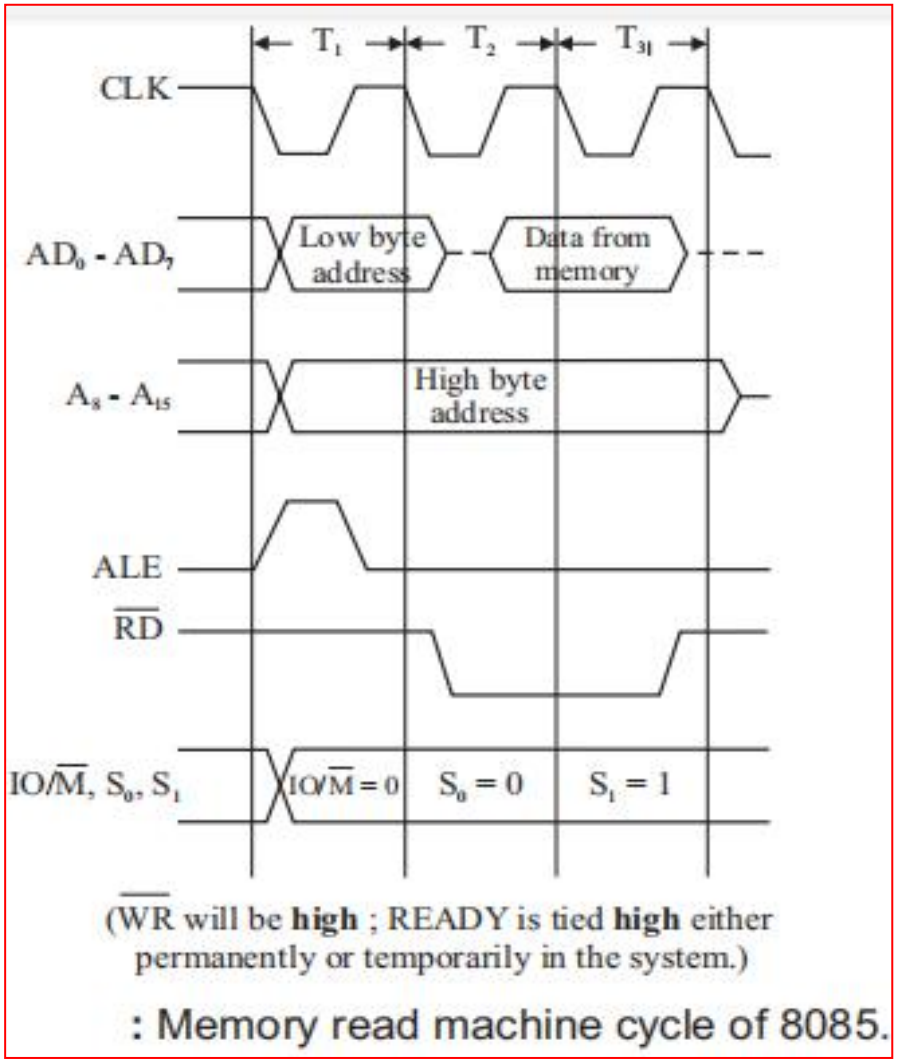


Fig. : Opcode fetch machine cycle of 8085.

Timing Diagram for Memory read cycle in 8085:

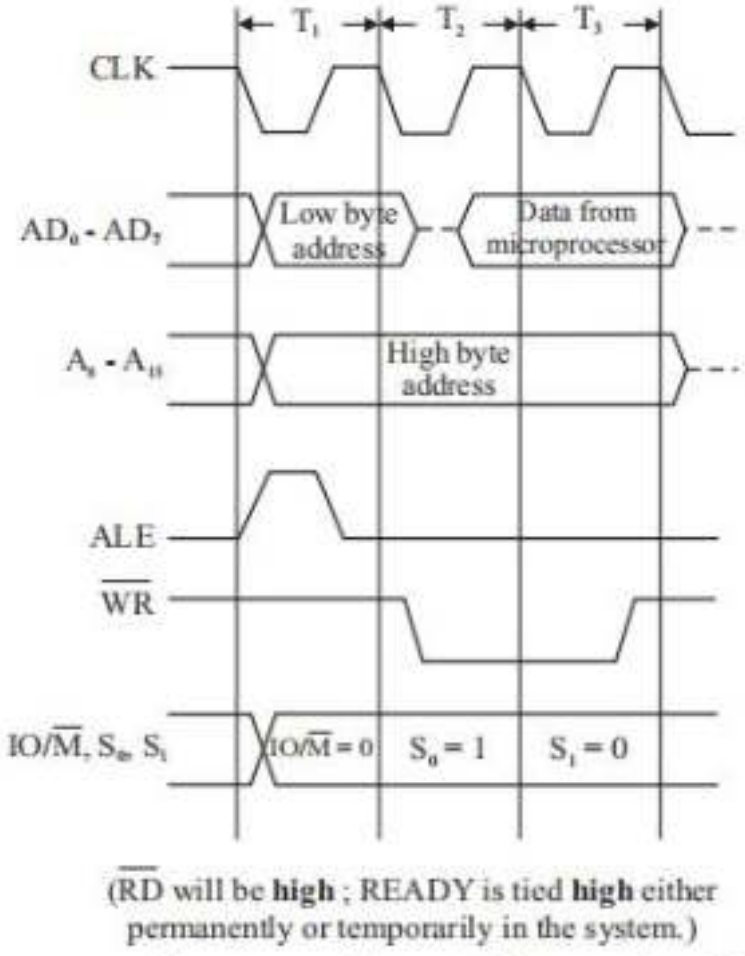
- The memory read machine cycle is executed by the processor to read a data byte from memory.
- The processor takes 3T states to execute this cycle.
- At the falling edge of T1 , the microprocessor outputs the low byte address on AD0 - AD7 lines and high byte address on A8 to A15 lines.
- ALE is asserted high to enable the external address latch.
- The other control signals are asserted as follows. IO/M=0, S0 = 0, S1 = 1. (IO/M is asserted low to indicate memory access.)
- At the middle of T1 , the ALE is asserted low and this enables the external address latch to take low byte of address and keep on its output lines.
- In the second T-state (T2), the memory is requested for read by asserting read line low. When read is asserted low, the memory is enabled for placing the data on the data bus. The time allowed for memory to output the data is the time during which read remains low.
- At the end of T3 , the read signal is asserted high. On the rising edge of read signal, the data is latched into microprocessor. Other control signals remain in the same state until the next machine cycle.



: Memory read machine cycle of 8085.

Timing Diagram for Memory write Cycle in 8085

- The memory write machine cycle is executed by the processor to write a data byte in a memory location.
- The processor takes 3T states to execute this machine cycle.
- At the falling edge of T1 , the microprocessor outputs the low byte address on AD0 - AD7 lines and high byte address on A8 to A15 lines.
- ALE is asserted high to enable the external address latch. The other control signals are asserted as follows. IO/M=0, S0 = 1, S1 = 0. (IO/M is asserted low to indicate memory access.)
- At the middle of T1 , the ALE is asserted low and this enables the external address latch for latching the low byte address into its output lines.
- In the falling edge of T2 , the processor output data on AD0 to AD7 lines and then request memory for write operation by asserting the write control signal WR to low.
- At the end of T3 , the processor asserts WR high. This enables the memory to latch the data into it. The memory should prepare itself to accept the data within the time duration in which write control signal remains low. Other control signals remain in the same state until the next machine cycle.



($\overline{RD}$ will be **high** ; READY is tied **high** either permanently or temporarily in the system.)

Memory write machine cycle of 8085.

Thank You

MPES
Module 1_15

Thank You

Assembly Language programs in 8085:

Addition of numbers in an array of three numbers (Assume 16 bit Result)

Address	Hex Code	Label	Mnemonics	Comments
6000	0E		MVI C, 03H;	Load register C with 03H
6001	01			
6002	6		MVI B, 00H;	Load register B with 00H
6003	0			
6004	21		LXI H, 6100 H;	Initialise memory pointer to 6100
6005	0			
6006	61			
6007	AF		XRA A;	Clear Accumulator
6008	86	RPT	ADD M;	Add contents of accumulator with contents in memory
6009	D2		JNC NEXT;	Jump to NEXT if carry is not set
600A	0D			
600B	60			
600C	3		INR B;	Increment B for getting the higher byte of result
600D	23	NEXT	INX H;	Increment M, Memory location pointed by HL pair
600E	0		DCR C;	Decrement C for the counter operation.
600F	C2		JNZ RPT;	Jump to RPT if register [C]= 0
6010	8			
6011	60			
6012	6F		MOV L, A;	Move contents of A to L
6013	60		MOV H, B;	Move contents of B to H

6100	80 H
6101	80 H
6102	02 H

Program to arrange the numbers in an array. [Ascending / Descending Order]

Address	Hex Code	Label	Mnemonics	Comments
6000	0E		MVI C, 04 H;	C is initialised as a counter 1 to count the iterations
6001	9			
6002	41	LOOP 1	MOV B, C;	B is initialised as counter 2 to count the no of comparisons in each iteration
6003	21		LXI H, 6100 H;	Memory Pointer initialised
6004	0			
6005	61			
6006	7E	LOOP 2	MOV A, M;	First number moved to accumulator.
6007	63		INX H;	Memory Pointer incremented
6008	56		MOV D, M;	Next number in array is moved to register D
6009	BA		CMP D ;	Numbers are compared.
600A	DA / D2		JC / JNC NEXT;	Go to the next comparison if [A] < [D] for ascending order
600B	11			
600C	60			
600D	77		MOV M, A;	Otherwise exchange the numbers.
600E	2B		DCX H ;	Memory pointer Decremented
600F	72		MOV M, D ;	Move [D] to memory
6010	63		INX H;	Memory pointer incremented
6011	5	NEXT	DCR B;	Counter 2 decremented and then repeat the process
6012	C2		JNZ LOOP 2;	Jump to LOOP 2 if [B] not equal to zero.
6013	6			
6014	60			
6015	0D		DCR C;	Decrement counter 1 and repeat the process
6016	C2		JNZ LOOP 1;	Jump to LOOP 1 if [C] not equal to zero.
6017	2			
6018	60			

Example for Ascending order sorting:

		First iteration (C = 4) (B = 4)			
Original Array	Address	B= 3	B= 2	B=1	B= 0
5	6100	5	5	5	5
7	6101	7	3	3	3
3	6102	3	7	7	7
8	6103	8	8	8	1
1	6104	1	7	1	8
		Second iteration (C = 3) (B=3)			
		B = 2	B = 1	B = 0	
	6100	3	2	3	
	6101	5	5	5	
	6102	7	7	1	
	6103	1	1	7	
	6104	8	8	8	
		Third iteration (C = 2) (B=2)			
		B = 1	B = 0		
	6100	3	3		
	6101	5	1		
	6102	1	5		
	6103	7	7		
	6104	8	8		
		Fourth iteration (C = 1) (B= 1)			
		B = 0			
	6100	1			
	6101	3			
	6102	5			
	6103	7			
	6104	8			
		C = 0			

MPES
Module 1_16

Program to Convert a Binary number to BCD number

Address	Hex Code	Label	Mnemonics	Comments
6000	06		MVI B, 00 H;	Clear Register B to store the hundreds
6001	0			
6002	48		MOV C, B;	Clear Register C to store tens
6003	3A		LDA 6200 H;	The number to be converted is loaded to A
6004	0			
6005	62			
6006	FE	HUN	CPI 64 H;	If the number > 100 (64 H), find the number of hundreds in the number
6007	64			
6008	DA		JC TEN;	Jump to TEN if carry is generated
6009	11			
600A	60			
600B	D6		SUI 64 H ;	Division by hundred
600C	64			
600D	4		INR B;	Increment register B
600E	C3		JMP HUN;	Jump to HUN
600F	6			
6010	60			
6011	FE	TEN	CPI 0A H;	If number > 10 (0A H), Find the number of tens in the number
6012	0A			
6013	DA		JC UNIT;	Jump to unit if carry
6014	1C			
6015	60			
6016	D6		SUI 0A H;	Division by ten
6017	0A			
6018	0C		INR C;	Increment the content of register C
6019	C3		JMP TEN;	Jump to TEN
601A	11			
601B	60			
601C	57	UNIT	MOV D, A;	unit is saved in register D
601D	60		MOV H, B;	Hundreds is moved to register H
601E	79		MOV A, C;	Tens value stored to A
601F	7		RLC;	
6020	7		RLC;	
6021	7		RLC;	
6022	7		RLC;	
6023	82		ADD D;	Add the units value to tens value in A
6024	6F		MOV L, A;	Move the content of accumulator to L
6025	76		HLT;	Program Halted , now the result , ie, BCD number in [HL]

	Binary	BCD Number	
Eg:	80 H	128	[100*1 + 10 *2 + 1* 8]

Program to Convert a BCD to Binary number

Address	Hex Code	Label	Mnemonics	Comments
6000	21		LXI H, 6200 H;	Memory pointer initialised
6001	0			
6002	62			
6003	7E		MOV A, M;	Content in memory location pointed is moved to A
6004	47		MOV B, A;	Move the number to register B
6005	E6		ANI 0F H;	To separate BCD1 and BCD 2, Mask higher nibble.
6006	0F			
6007	4F		MOV C, A;	BCD 1 is saved in register C
6008	78		MOV A, B;	Original BCD number is moved to A
6009	E6		ANI F0 H;	Mask Lower nibble of BCD number
600A	F0			
600B	0F		RRC;	Rotate four times to right to get BCD 2
600C	0F		RRC;	
600D	0F		RRC;	
600E	0F		RRC;	
600F	57		MOV D, A;	Save BCD 2 to register D
6010	AF		XRA A;	Clear Accumulator
6011	1E		MVI E, 0A;	BCD2 X 10 as it comes at tenth place.
6012	0A			
6013	83	LOOP	ADD E;	Add [E] to [A]
6014	15		DCR D;	Decrement D register
6015	C2		JNZ LOOP;	Jump to LOOP if [D] not equal to zero
6016	13			
6017	60			
6018	81		ADD C;	(BCD2 x 10) + BCD1
6019	76		HLT;	Program Halted , now the result , ie, Binary number in [A]

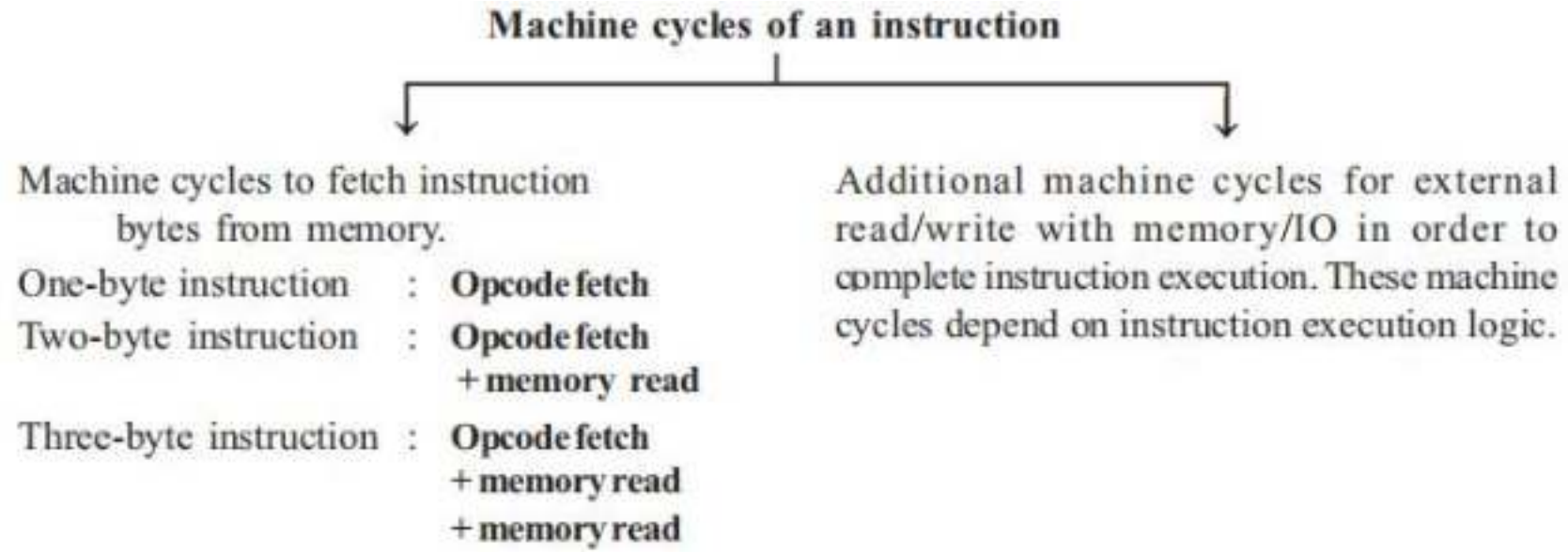
BCD	BCD2	BCD1		
28	2	8		
0010 1000	0010 0000	0000 1000		Separated BCD 1 and BCD2
	0000 0010			Rotated BCD2 for 4 times
	0001 0100			Added 0A H to cleared [A] for two times(BCD2 after rotation) here.
			00011100 (1C H)	Binary of 28 , ie result after adding BCD1 to [A]

Thank You

MPES
Module 1_17

Timing Diagram of 8085 Instructions (Contd…):

- The execution of an instruction is the execution of the machine cycles of that instruction in a predefined order.
- Therefore, from the knowledge of the timing diagrams of machine cycles, the timing diagram of an instruction can be obtained.

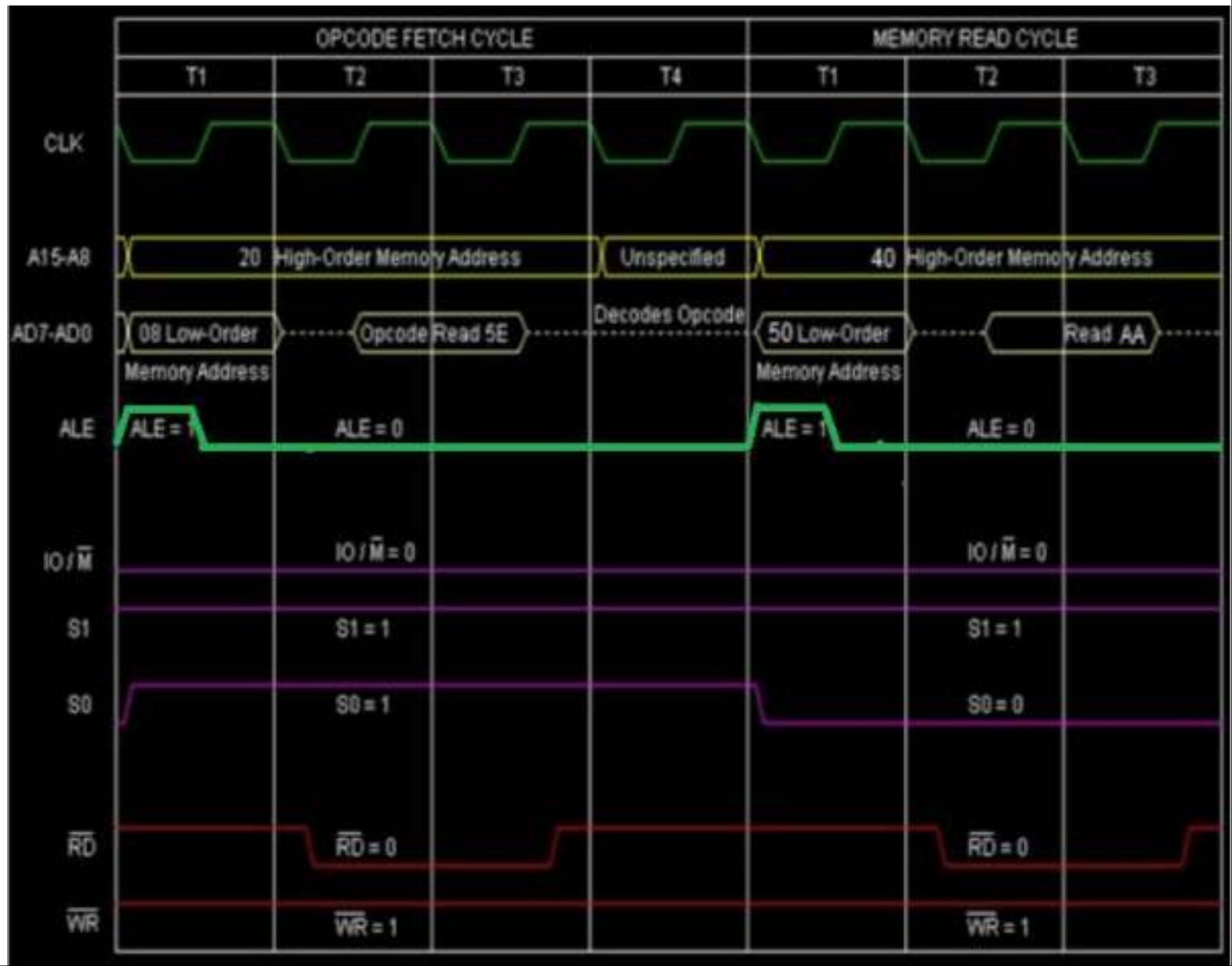


Example: STA 526AH , PUSH B etc….

Timing Diagram of MOV Rd, M

- Though its a one byte instruction , it requires two machine cycles for execution. (Eg. for a special case)
  - Opcode fetch
  - Memory Read
- **Here in the given timing diagram, we have considered MOV E, M;**

  [Suppose the instruction is stored in memory location 2008 H and E register content is DB H, H register content is 40H, and L register content is 50H.Let us say location 4050H has the data value AAH. When the 8085 executes this instruction, the contents of E register will change to AAH, as shown below.]



	Before	After
(E)	DBH	AAH
(HL)	4050H	4050H
(4050H)	AAH	AAH

Timing Diagram for STA 526A H;

- Let the content of the accumulator be C7H and it is desired to store the content of the accumulator to a memory location 526AH.
- The STA addr16 instruction is a three byte instruction. The first byte is the opcode of the instruction 32H. The second byte is low byte address 6AH and the third byte is high byte address 52H.
- Let the three bytes of the instructions be stored in memory locations 41FFH, 4200H and 4201H.

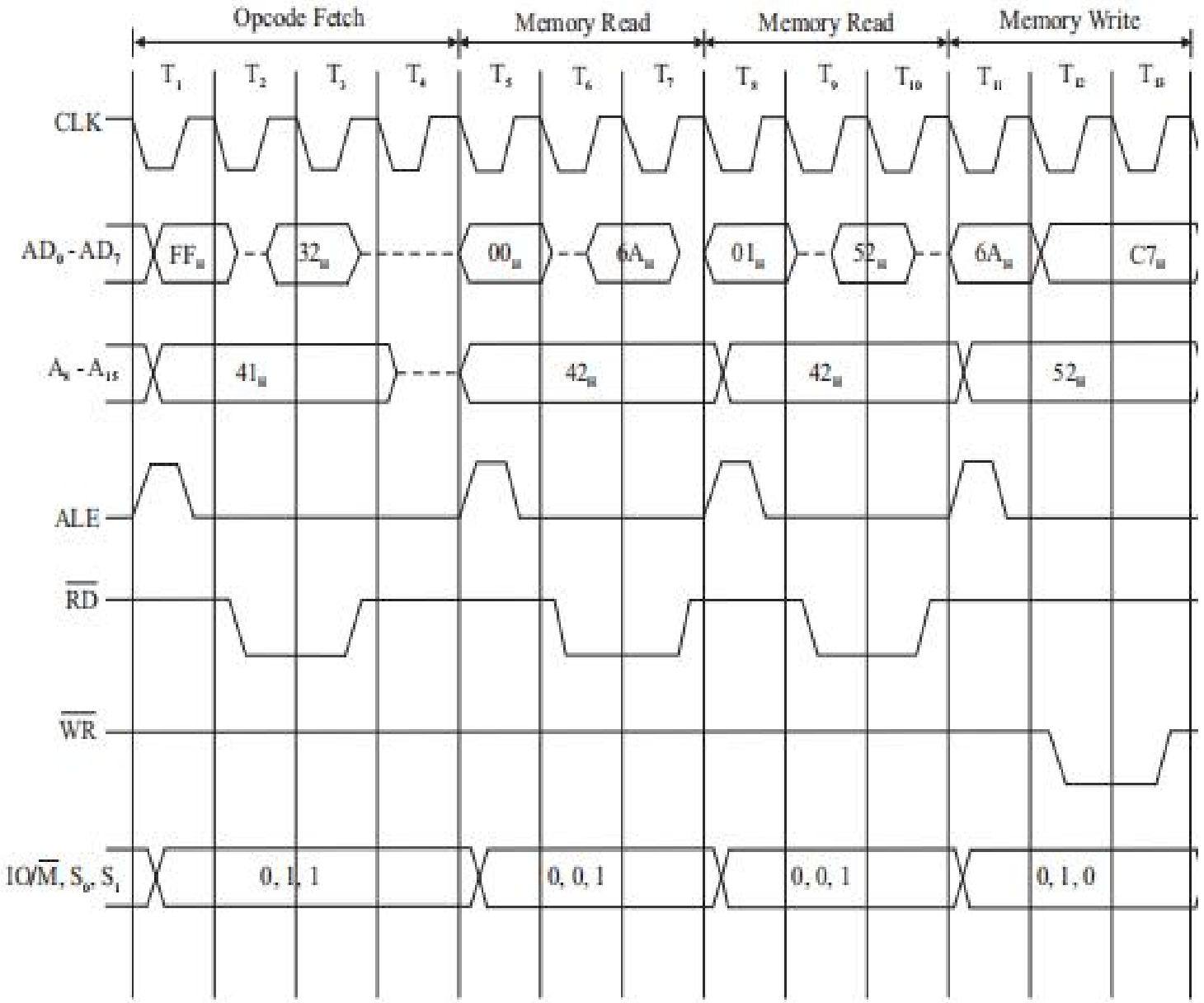


Fig. : Timing diagram of STA 526AH instruction.

Thank You

MPES
Module 1_18

Delay Subroutines in 8085:

- Delay routines are the subroutines used for maintaining the timings of various operations in a microprocessor.
- As an example, In control applications, certain equipment need to be ON/OFF after a specified time delay. In some applications, a certain operation has to be repeated after a specified time interval.
- A delay routine is generally written as a subroutine (It need not be a subroutine always. It can even be a part of the main program.) In a delay routine a count (number) is loaded in a register of microprocessor. Then it is decremented by one and the zero flag is checked to verify whether the content of register is zero or not. This process is continued until the content of the register is zero. When it is zero the time delay is over and the control is transferred to the main program to carry out the desired operation.
- The delay time is given by the total time taken to execute the delay routine. It can be computed by multiplying the total number of T states required to execute the subroutine and the time for one T-state of the processor. The total of number of T states can be computed from the knowledge of T states required for each instruction.

- The time for one T-state of the processor is given by the inverse of the internal clock frequency of the processor.
- For example, if the 8085 microprocessor has 5 MHz quartz crystal then,

$$\text{The internal clock frequency} = \frac{5}{2} = 2.5 \text{ MHz}$$

$$\text{Time for one T-state} = \frac{1}{2.5 \times 10^6} = 0.4 \text{ msec}$$

- For small time delays (< 0.5 millisecond) an 8-bit register can be used as counter, but for large time delays (< 0.5 second) 16-bit register should be used as counter. For very large time delays (>0.5 second), a delay routine can be repeatedly called in the main program. The disadvantage in delay routines is that the processor time is wasted.
- An alternate solution is to use a dedicated timer like 8253/8254 to produce time delays or to maintain timings of various operations.

EXAMPLE DELAY ROUTINE - 1

Write a delay routine to produce a time delay of 0.5 millisecond in 8085 processor-based system whose clock source is 6 MHz quartz crystal.

Solution

The delay required is 0.5 millisecond, hence an 8-bit register of 8085 can be used to store a count value. The count is decremented by one and the zero flag is verified. If zero flag is set then decrement operation is terminated. The delay routine is written as a subroutine as shown below:

Delay routine

```
        MVI  D,N   ; Load the count value, N in D-register.
LOOP:   DCR  D     ; Decrement the count.
        JNZ  LOOP  ; If count is not zero go to LOOP.
        RET        ; If count is zero return to main program.
```

Instruction	T-state required for execution of an instruction	Number of times the instruction is executed	Total T states
CALL addr16	18	1	$18 \times 1 = 18$
MVI D,N	7	1	$7 \times 1 = 7$
DCR D	4	N times	$4 \times N = 4N$
JNZ LOOP	10	(N–1) times	$10 \times (N-1) = 10N-10$
	or 7	1	$7 \times 1 = 7$
RET	10	1	$10 \times 1 = 10$
		Total T-state required for subroutine	$= 14N+32$

Calculation to find the count value, N

External Clock frequency = 6 MHz

$$\text{Internal Clock frequency} = \frac{\text{External Clock}}{2} = \frac{6}{2} = 3 \text{ MHz}$$

$$\text{Time period of one T-state} = \frac{1}{\text{Internal Clock frequency}} = \frac{1}{3 \times 10^6} = 0.3333\ \mu s$$

$$\left.\begin{matrix}\text{Number of T states} \\ \text{required for 0.5 ms}\end{matrix}\right\} = \frac{\text{Required time delay}}{\text{Time for one T-state}} = \frac{0.5 \times 10^{-3}}{0.3333 \times 10^{-6}} = 1500.15 = 1500_{10}$$

On equating the total T states required for the subroutine and the number of T states for the required time delay, the count value, N can be calculated.

$$\therefore 14N + 32 = 1500_{10}$$

$$N = \frac{1500 - 32}{14} = 104.857_{10} \approx 105_{10} = 69_{H}$$

$\therefore$ Count value, $N = 69_{H}$

If the above delay routine is called by a program and executed with count value of 69_{H} then the delay produced will be 0.5 millisecond.

EXAMPLE DELAY ROUTINE - 2

Write a delay routine to produce a time delay of 0.5 second in 8085 processor-based system whose internal clock frequency is 3 MHz.

Solution

The delay required is large, hence a 16-bit register can be used for storing the count value. The count is decremented one by one until it is zero. After each decrement operation we have to verify whether the content of register pair is zero or not. This can be performed by logically ORing the content of low order and high order register and then checking the zero flag. (Because the 16-bit increment/decrement instruction will not modify any flag.) The delay routine is written as a subroutine as shown below:

Delay Routine

```
      LXI  D,N   ; Load the count value, N in DE-register pair.
LOOP: DCX  D     ; Decrement the count.
      MOV  A,E   ; Logically OR the content of
      ORA  D     ; E-register with D-register.
      JNZ  LOOP  ; If count is not zero, go to LOOP.

      RET        ; If count is zero, return to main program.
```

Calculation to find the count value, N

Internal Clock frequency = 3 MHz

$$\text{Time period of one T-state} = \frac{1}{\text{Internal Clock frequency}} = \frac{1}{3 \times 10^6} = 0.3333\ \mu s$$

$$\left.\begin{array}{l}\text{Number of T states required}\\ \text{for 0.5 second}\end{array}\right\} = \frac{\text{Required time delay}}{\text{Time for one T-state}} = \frac{0.5\ \text{sec}}{0.3333 \times 10^{-6}}$$

$$= 1500150.015_{10} = 1500150_{10}$$

On equating the total T states required for the subroutine and the number of T states for the required time delay, the count value, N can be calculated.

$$\therefore 24N + 35 = 1500150_{10}$$

$$N = \frac{1500150 - 35}{24} = 62504.79_{10} \approx 62505_{10} = F429_H$$

$\therefore$ Count value, $N = F429_H$

If the above delay routine is called by a program and executed with count value of $F429_H$ then the delay produced will be 0.5 second.

Note : The registers used in the delay routine are A, D and E. Also the execution of delay routine will alter the flags. Hence if the contents of these register are to be preserved, then the main program has to save them in stack before calling the delay routine.

Thank You

MPES
Module 1_19

Interrupt Structure in 8085:

- The process of interrupting the normal program execution to carry out a specific task/work is referred to as interrupt.
- The interrupt is initiated by a signal generated by an external device (**Hardware Interrupts**) or by a signal generated internal to the processor (**Software Interrupts**).
- When a microprocessor receives an interrupt signal, it stops executing the current main program, saves the status (or content) of various registers (PC in case of 8085) in stack and then executes a subroutine in order to perform the specific task requested by the interrupt. The subroutine that is executed in response to an interrupt is also called **Interrupt Service Routine (ISR)**. At the end of ISR, the stored status of registers in stack are restored to respective registers and the processor resumes the normal main program execution from the point (instruction) where it was interrupted.
- When interrupt occured,

(SP) ← (SP) – 1 ; ((SP)) ← (PC)H

(SP) ← (SP) – 1 ; ((SP)) ← (PC)L

(PC) ← address of ISR

- When **RET** is executed in the ISR,

(PC)L ← ((SP)) ; (SP) ← (SP) + 1

(PC)H ← ((SP)) ; (SP) ← (SP) + 1

- The external interrupts are used to implement interrupt driven data transfer scheme. The interrupts generated by special instructions are called software interrupts and they are used to implement system services/calls (or monitor services/calls). The system /monitor services are procedures developed by the system designer for various operations and stored in memory. Theuser can call these services through software interrupts. The interrupts generated by exceptional conditions are used to implement error conditions in the system.

Interrupt Driven Data Transfer Scheme

- Interrupts are useful for efficient data transfer between the processor and the peripheral. When a peripheral is ready for data transfer, it interrupts the processor by sending an appropriate signal. Upon receiving an interrupt signal, the processor suspends the current program execution,saves the status in a stack and executes an ISR to perform the data transfer between the peripheraland the processor. At the end of ISR the processor status is restored from stack and the processor resumes its normal program execution. This type of data transfer scheme is called interrupt drivendata transfer scheme.

- The data transfer between the processor and peripheral devices can be implemented either by polling technique or by interrupt method. In polling technique, the processor has to periodically poll or check the status/readiness of the device and can perform data transfer only when the device is ready.
- In polling technique the processor time is wasted, because the processor has to suspend its work and check the status of the device in predefined intervals.

Example: To detect a keyboard press when a keyboard is interfaced to microprocessor.

CLASSIFICATION OF INTERRUPTS

In general interrupts can be classified in the following three ways:

- Hardware and software interrupts.
- Vectored and non-vectored interrupts.
- Maskable and non-maskable interrupts.

- Interrupts initiated by external hardware by sending an appropriate signal to the interrupt pin of the processor is called hardware interrupt. The 8085 processor has five interrupt pins TRAP, RST 7.5, RST 6.5, RST 5.5 and INTR and the interrupts initiated by applying appropriate signal to these pins are called hardware interrupts of 8085.
- Software interrupts are program instructions. When a software interrupt instruction is executed, the processor executes an Interrupt Service Routine(ISR) stored in the vector address of that software interrupt instruction. The software interrupts of 8085 are RST0, RST1, RST2, RST3, RST4, RST5, RST6 and RST7. The software interrupts of 8085 are vectored interrupts. Software interrupts cannot be masked or be disabled.
- When an interrupt signal is accepted by the processor, and the program control automatically branches to a specific address (called vector address) then the interrupt is called vectored interrupt. The automatic branching to a vector address is predefined by the manufacturer of the processor. (In these vector addresses the interrupt service subroutines(ISR) are stored.) In non-vectored interrupts the interrupting device should supply the address of the ISR to be executed in response to the interrupt. All the 8085 interrupts excepts **INTR** are vectored interrupts.
- The interrupts whose request can be either accepted or rejected by the processor are called maskable interrupts. The interrupts whose request has to be definitely accepted (i.e., it cannot be rejected) by the processor are called non-maskable interrupts. In 8085 the hardware interrupts RST 7.5, RST 6.5, and RST 5.5 can be masked/unmasked using SIM instruction. All the hardware interrupts except TRAP are disabled by executing DI instruction and they are enabled by executing EI instruction.

Thank You

MPES
Module 1_20

Interrupt Structure in 8085 Contd.....

Hardware Interrupts:

- Interrupts initiated by external hardware by sending an appropriate signal to the interrupt pin of the processor is called hardware interrupt. The 8085 processor has five interrupt pins TRAP, RST 7.5, RST 6.5, RST 5.5 and INTR.
- In 8085 the hardware interrupts RST 7.5, RST 6.5, and RST 5.5 can be masked/unmasked using SIM instruction.
- All the hardware interrupts except TRAP are disabled by executing DI instruction and they are enabled by executing EI instruction.
- All the 8085 interrupts except **INTR** are vectored interrupts, ie the program control automatically branches to a specific address (called vector address) predefined by the manufacturer.
- If the interrupt is non vectored, then the interrupting device has to supply the address of ISR when it receives INTA signal. Then the processor starts executing ISR in this address.

[TRAP is edge as well as level trigerred (to avoid false interrupt due to noise), ie, TRAP should go high and stay high until it is acknowledged by the processor by clearing the flip flop so that future interrupts can be accepted. RST 7.5 is rising edge triggered and INTR, RST 5.5, RST 6.5 are High level triggered.]

Interrupt	Vector address
RST 7.5	$003C_H$
RST 6.5	0034_H
RST 5.5	$002C_H$
TRAP	0024_H

Software Interrupts:

- Software interrupts are program instructions. When a software interrupt instruction is executed, the processor executes an Interrupt Service Routine(ISR) stored in the vector address of that software interrupt instruction.
- The software interrupts of 8085 are RST0, RST1, RST2, RST3, RST4, RST5, RST6 and RST7.
- The software interrupts of 8085 are vectored interrupts.
- Software interrupts cannot be masked or be disabled.

Interrupt	Vector address
RST 0	0000_H
RST 1	0008_H
RST 2	0010_H
RST 3	0018_H
RST 4	0020_H
RST 5	0028_H
RST 6	0030_H
RST 7	0038_H

Interrupt priority in 8085:

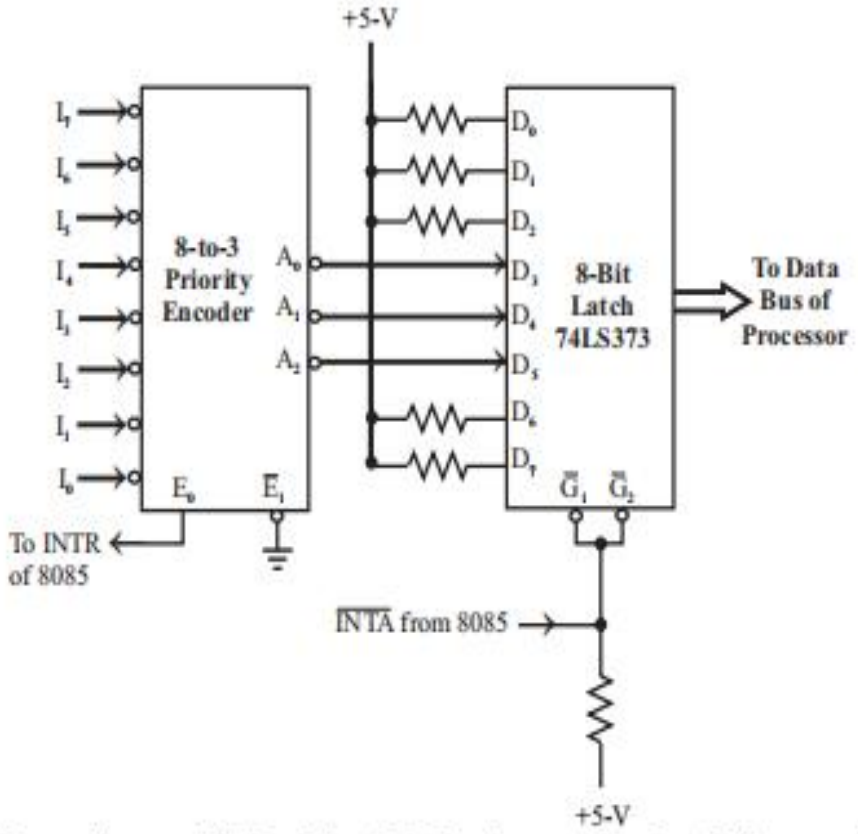
- When all the interrupts are enabled, the priority sequence of hardware interrupts from highest to lowest is TRAP, RST 7.5, RST 6.5, RST 5.5 and INTR.
- When the 8085 processor accepts an interrupt, it will disable all the hardware interrupts except TRAP. Hence in order to allow the higher priority interrupt while executing Interrupt Service Subroutine (ISR) for lower priority interrupt, enable the interrupt system in the beginning of ISR of lower priority interrupt, by executing EI instruction.

Enabling, Disabling and Masking of 8085 Interrupts:

- All the hardware Interrupts except TRAP in 8085 can be Enabled or Disabled with the help of EI and DI instructions respectively. Also the the interrupts except TRAP can be disabled by system (processor) reset or after recognition of another interrupt.
- The only signal which can override TRAP is HOLD signal. (i.e., If the processor receives HOLD and TRAP at the same time then HOLD is recognized first and only then is TRAP recognized.)
- [Software interrupts are non maskable and it is initiated only through program.]
- All the hardware Interrupts except TRAP in 8085 can be Masked or Unmasked with the help of **SIM** instruction.

INTR and its expansion:

- An external device can interrupt the processor by placing a **high** signal on INTR pin of 8085. If the processor accepts the interrupt, then it will send an acknowledge signal INTA to the interrupting device.
- On receiving the acknowledge signal, the interrupting device has to place either an RST n opcode (or CALL opcode followed by 16-bit address) on the data bus.
- On receiving the RST n opcode, the 8085 processor generates the vector address of RST n instruction.
- [The INTR interrupt can be expanded to accept 8-interrupt inputs using 8-to-3 priority encoder.]
- This opcode is read by the processor and then it generates t instruction internally.



: Expanding an INTR of the 8085 using an 8-to-3 priority encoder.

MPES
Module 3_1

Memory Interfacing to 8085:

Micro Processor Based System:

The microprocessor-based system to perform a specific task consists of microprocessor as CPU, semiconductor memories like EPROM and RAM, input device, output device and interfacing devices. The memories, input device, output device and interfacing devices are called peripherals.

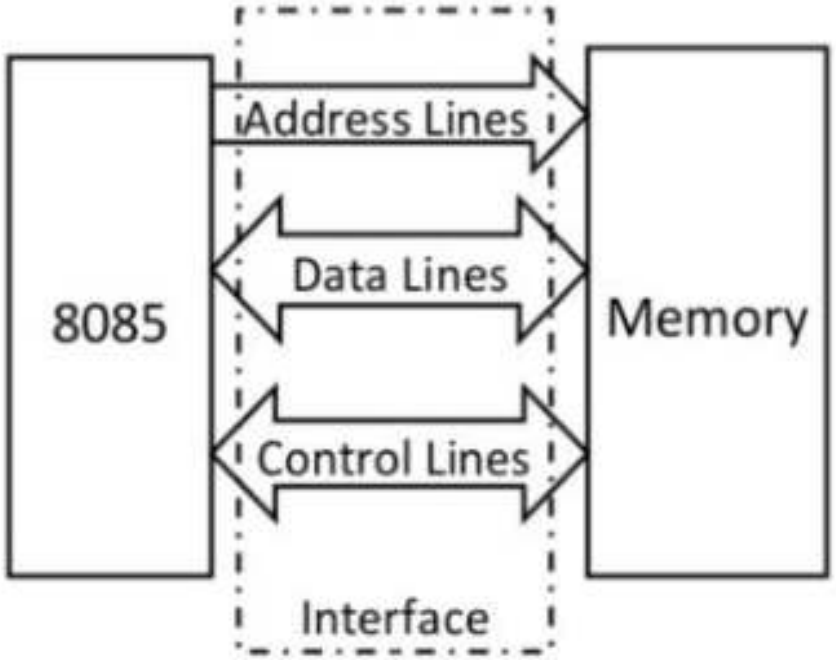
The EPROM memory is used to store permanent programs and data. The RAM memory is used to store temporary programs and data. The input device is used to enter the program, data and to operate the system. The output device is also used for examining the results.

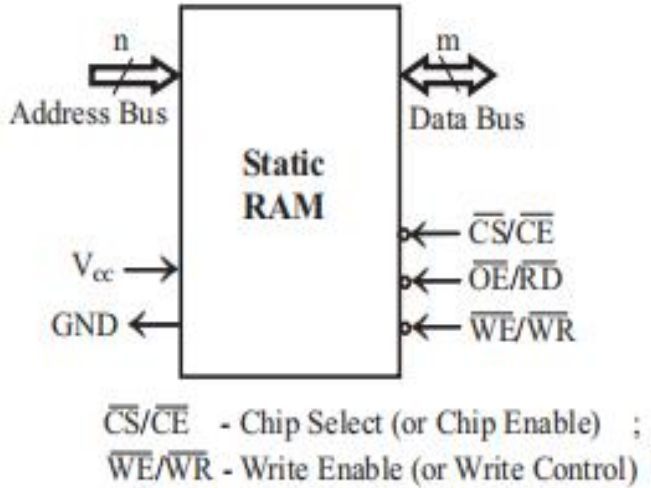
The microprocessor is the master, which controls all the activities of the system. To perform a specific job or task, the microprocessor has to execute a program stored in memory. The program consists of a set of instructions stored in consecutive memory location. In order to execute the program, the microprocessor issues address and control signals, to fetch the instruction and data from memory one by one. After fetching each instruction it decodes the instruction and carries out the task specified by the instruction.

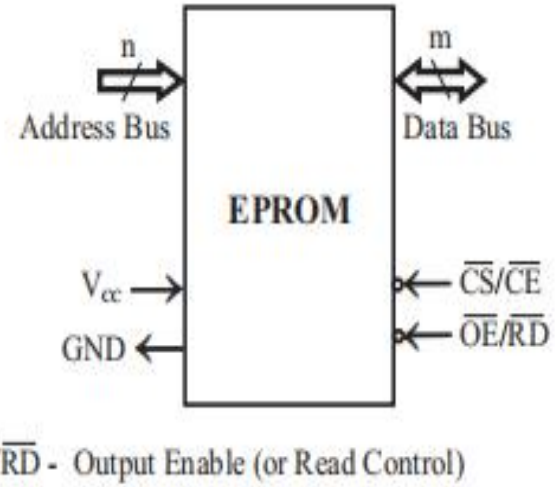
Contd...

- The basic data size of 8085 is 8-bit. Therefore, the memory word size of the memories interfaced with 8085 processor is also 8-bit or byte.
- The 8085 uses a 16-bit address to access memory and hence it can address upto 2^{16} = 65,53610 = 64 k memory locations.
- A memory unit is an integral part of any microcomputer system and its primary purpose is to store programs and data. In a broad sense, a microcomputer memory system can be logicallydivided into three groups. They are as follows:
    - Processor memory
    - Primary or main memory
    - Secondary memory
- Processor memory refers to registers inside the microprocessor. These registers are used to hold data and results temporarily when computation is in progress. Since the registers of the processor are fabricated using the same technology as that of a microprocessor, there is no speed disparity between these registers and a microprocessor. However, the cost involved in this approach forces a manufacturer to include only a few registers in the microprocessor.
- The READY is an input signal that can be used by slow peripherals to get extra time in order to communicate with 8085. The 8085 will work only when READY is tied to logic high. WheneverREADY is tied to logic low, the 8085 will enter a wait state.

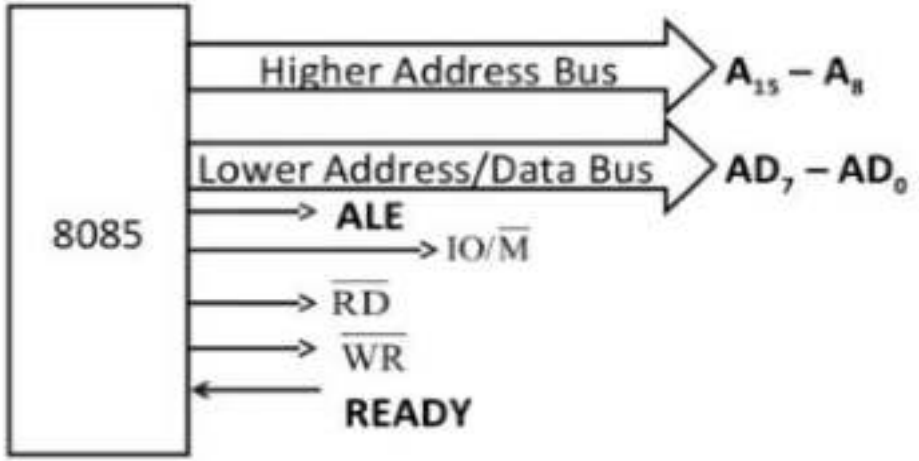
Block schematic representation for memory interfacing to 8085:





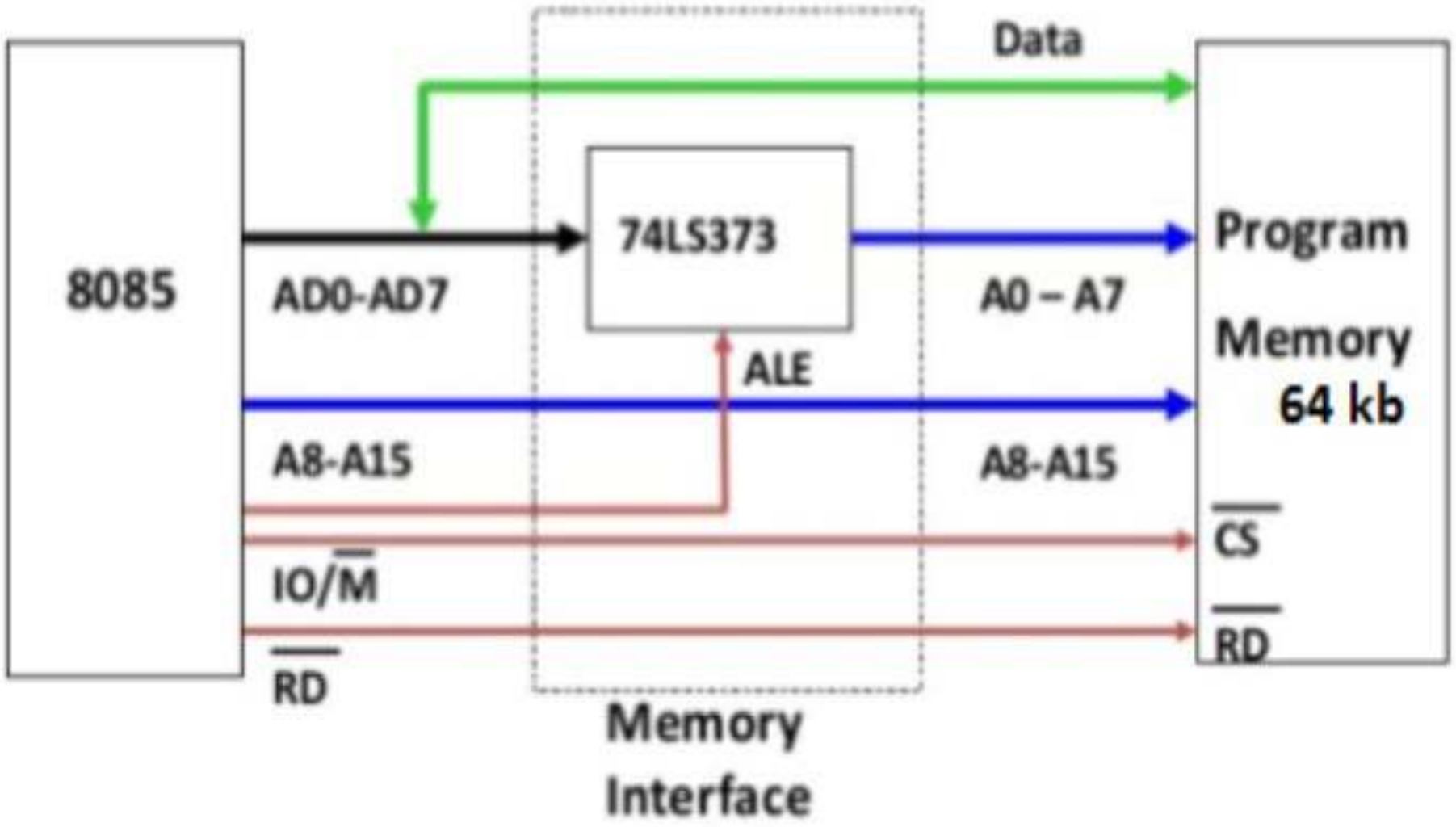


8085 Interfacing Pins



A typical semiconductor memory IC will have n address pins, m data pins (or output pins) and a minimum of two power supply pins (one for connecting required supply voltage (VCC) and the other for connecting ground). The control signals needed for static RAM are chip select (chip enable), read control (output enable) and write control (write enable). The control signals needed for read operation in EPROM are chip select (chip enable) and read control (output enable).

Example: Interfacing of a 64 kb program memory (EPROM) to 8085.



Thank You

MPES
Module 3_2

Generation of Chip select Signals for memory interfacing:

- Using logical gates
- Using Decoder IC's

    2 to 4 decoder [74LS139] and 3 to 8 decoder [74LS138] are generally used.

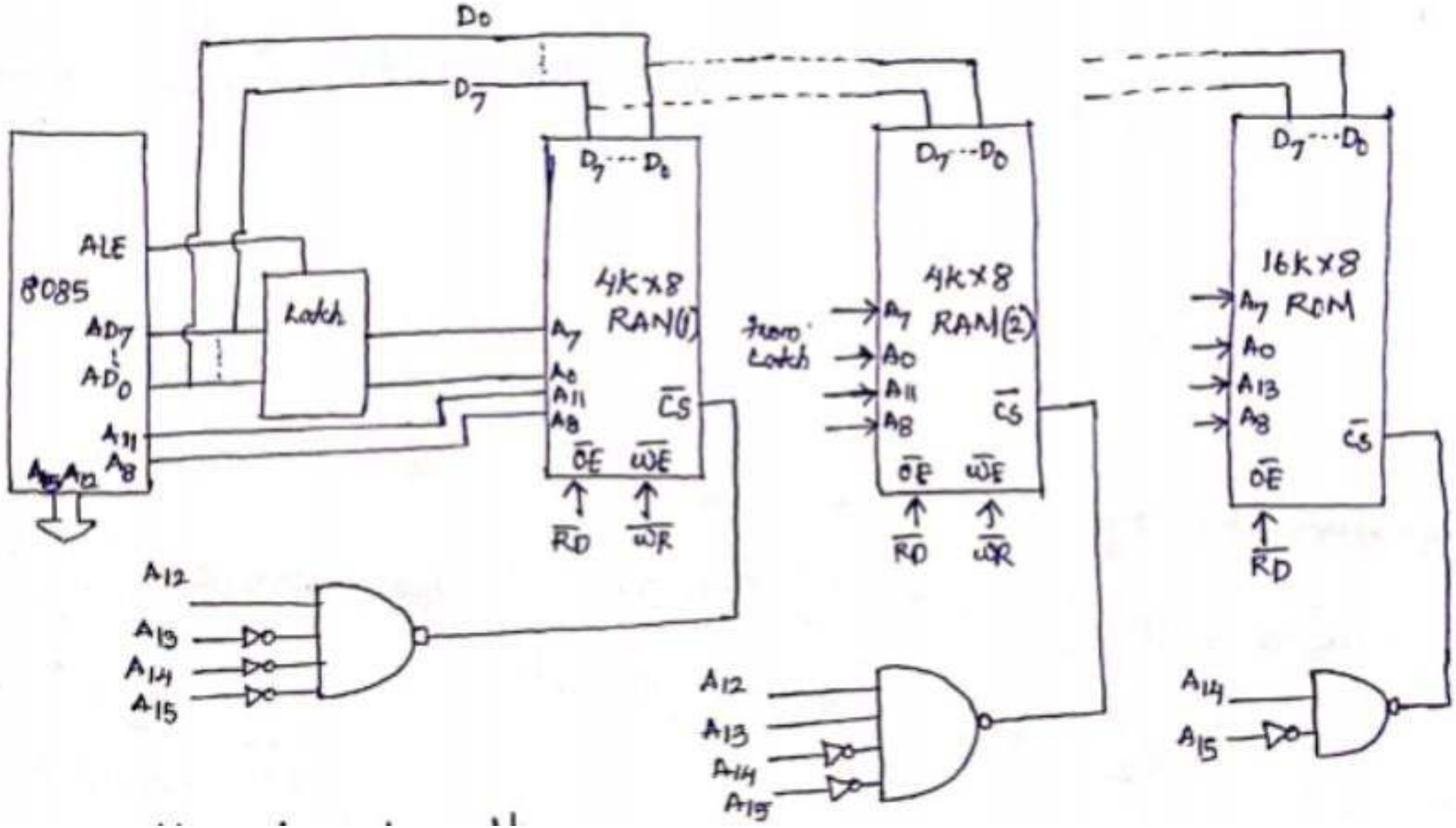
Example for using logical gates:

Q. Interface two separate 4kb of RAM and 16kb of ROM to 8085.

Solution:

For 4 k addressing we need 12 address lines, ie, $2^{12} = 4$ k

For 16 k addressing needs 14 address lines, ie, $2^{14} = 16$ k


D0
D7
8085
ALE
AD7
AD0
A11
A8
Latch
D7 ··· D0
4K×8
RAM(1)
A7
A0
A11
A8
CS
OE
WE
RD
WR
from Latch
4K×8
RAM(2)
16K×8
ROM
A12
A13
A14
A15

Starting address & Ending address

	A_{15}	A_{14}	A_{13}	A_{12}	A_{11}	A_{10}	A_{09}	A_{08}	A_{07}	A_{06}	A_{05}	A_{04}	A_{03}	A_{02}	A_{01}	A_{00}
RAM1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
	⋮															
	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1

i.e. First RAM responds for addresses 1000_H to $1FFF_H$.

RAM2	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0
	⋮															
	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1

i.e, Second RAM responds to addresses 3000_H to $3FFF_H$

ROM	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	⋮															
	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

i.e, ROM responds to addresses from 4000_H to $7FFF_H$

Using Decoder IC for chip selection:

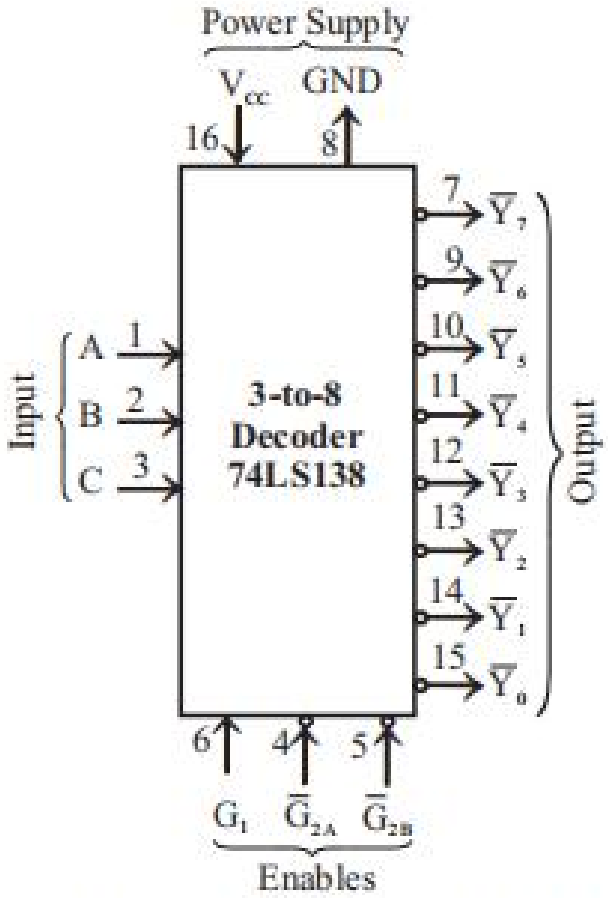


Fig. Signals of 74LS138.

TABLE : TRUTH TABLE OF 3-TO-8 DECODER

Enables			Input			Output							
G_1	$\overline{G}_{2A}$	$\overline{G}_{2B}$	C	B	A	$\overline{Y}_7$	$\overline{Y}_6$	$\overline{Y}_5$	$\overline{Y}_4$	$\overline{Y}_3$	$\overline{Y}_2$	$\overline{Y}_1$	$\overline{Y}_0$
1	0	0	0	0	0	1	1	1	1	1	1	1	0
1	0	0	0	0	1	1	1	1	1	1	1	0	1
1	0	0	0	1	0	1	1	1	1	1	0	1	1
1	0	0	0	1	1	1	1	1	1	0	1	1	1
1	0	0	1	0	0	1	1	1	0	1	1	1	1
1	0	0	1	0	1	1	1	0	1	1	1	1	1
1	0	0	1	1	0	1	0	1	1	1	1	1	1
1	0	0	1	1	1	0	1	1	1	1	1	1	1
0	1	1	X	X	X	H	H	H	H	H	H	H	H

Pin diagram and truth table for 3 to 8 decoder IC.

DESIGN EXAMPLE

Interface two numbers of 4 kb EPROM and one number of 8 kb RAM with 8085 processor. Explain the interface diagram and allocate binary addresses to memory ICs.

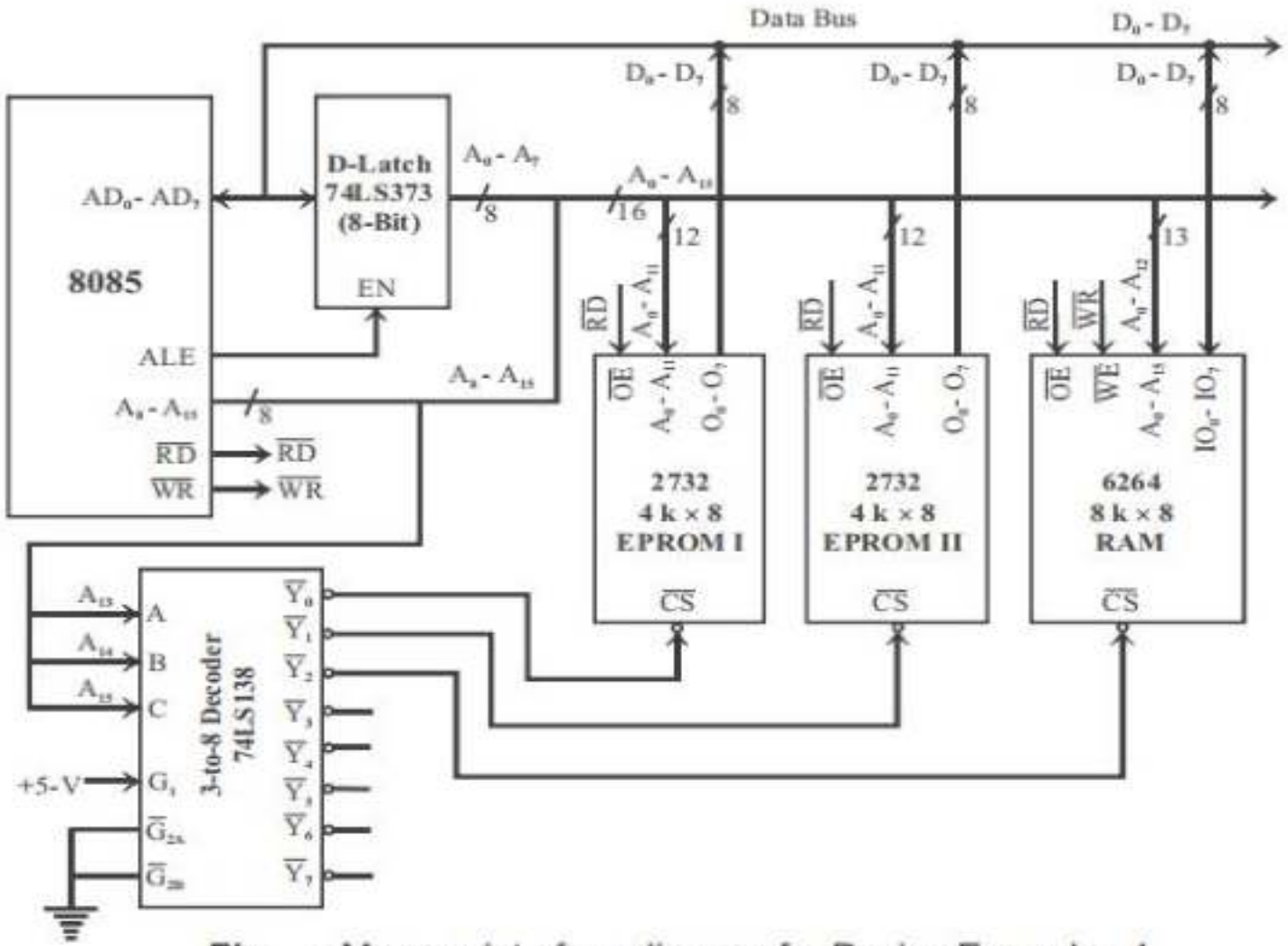


Fig. : Memory interface diagram for Design Example - 1.

The 4 kb EPROM IC requires 12 address lines ($2^{12} = 4$ k). The 8 kb RAM IC requires 13 address lines ($2^{13} = 8$ k). The address lines A_0 - A_{11} are connected to both EPROM and RAM address input pins. The address lines A_{13}, A_{14} and A_{15} are not used for memory address. Hence by decoding these address lines we can generate chip select signals.

The 3-to-8 decoder, 74LS138 is employed to produce the chip select signals for the system. The decoder has 8-output lines which can be used as 8-chip select signals. In this, three chip select signals are used for selecting memory ICs and the remaining five can be used for selecting other peripheral ICs in the system or for future expansion of the memory capacity. The interface diagram is shown in Fig. DE1. Addresss allotted to memory ICs are shown in Table-DE1.

The EPROM's are mapped in the beginning of memory space. The remaining addresses can be allotted to RAM's. The EPROM memory is mapped from 0000_H to $0FFF_H$ and 2000_H to $2FFF_H$. The RAM memory is mapped from 4000_H to $5FFF_H$.

Thank You

MPES
Module 3_3

Interfacing of IO and Peripheral devices to 8085:

- The IO devices connected to a microcomputer system provides an efficient means of communication between the microcomputer system and the outside world.
- These IO devices are commonly called peripherals and include keyboards, displays, printers and disks (hard disk and Compact Disc etc.)
- The IO devices are generally slow devices. So, they are connected to the system bus through ports. The ports are buffer IC which is used to temporarily hold the data transmitted from the microprocessor to IO device or to hold the data transmitted from IO device to the microprocessor.
- To data transfer from the input device to the processor the following operations are performed:
    - The input device will load the data to the port.
    - When the port receives the data, it sends message to the processor to read the data.
    - The processor will read the data from the port.
    - After the data has been read by the processor the input device will load the next data into the port.
- To data transfer from the processor to the output device the following operations are performed:
    - The processor will load the data to the port.
    - The port will send a message to the output device to read the data.
    - The output device will read the data from the port.
    - After the data has been read by the output device the processor can load the next data to the port.

The various INTEL IO port devices are 8212, 8155 /8156, **8255**, 8355 and 8755. **[Also since 8085 is having less number of IO ports, using 8255 PPI , we can increase the effective number of IO ports.]**

Intel 8255 PPI:

- The INTEL 8255 is a device used to implement parallel data transfer between processor and slow peripheral devices like ADC, DAC, keyboard, 7-segment display, LCD, etc.
- It has 3 numbers of 8-bit parallel IO ports (ports A, B and C).
- Port-A can be programmed in mode-0, mode-1 or mode-2 as input or output port.
- Port-B can be programmed in mode-1 and mode-2 as IO port.
- When ports A and B are in mode-0, port-C can be used as IO port. The individual pins of port-C can be set or reset.
- INTEL 8255 requires four internal addresses and has one logic low Chip Select (CS) pin. The address of internal devices of 8255 are listed in Table.

Table: **Internal Address of 8255**

Internal device	A_1	A_0
Port-A	0	0
Port-B	0	1
Port-C	1	0
Control Register	1	1

Control Words in 8255:

- The 8255 has **two** control words: **IO Mode Set control Word (MSW)** and **Bit Set/Reset (BSR) control word**.
- The MSW is used to specify IO functions and BSR word is used to set/reset individual pins of port-C. Both the control words are written in the same control register. The control register differentiates them by the value of bit B7
- The BSR control word does not affect the functions of ports A and B.
- Bit B7 of the control register specifies either the IO function or the bit set/reset function. If B7 = 1, then the bits B6 - B0determine IO functions in various modes. If bit B7 = 0, then the bits B6 -B0 determine the pin of port-C to be set or reset.

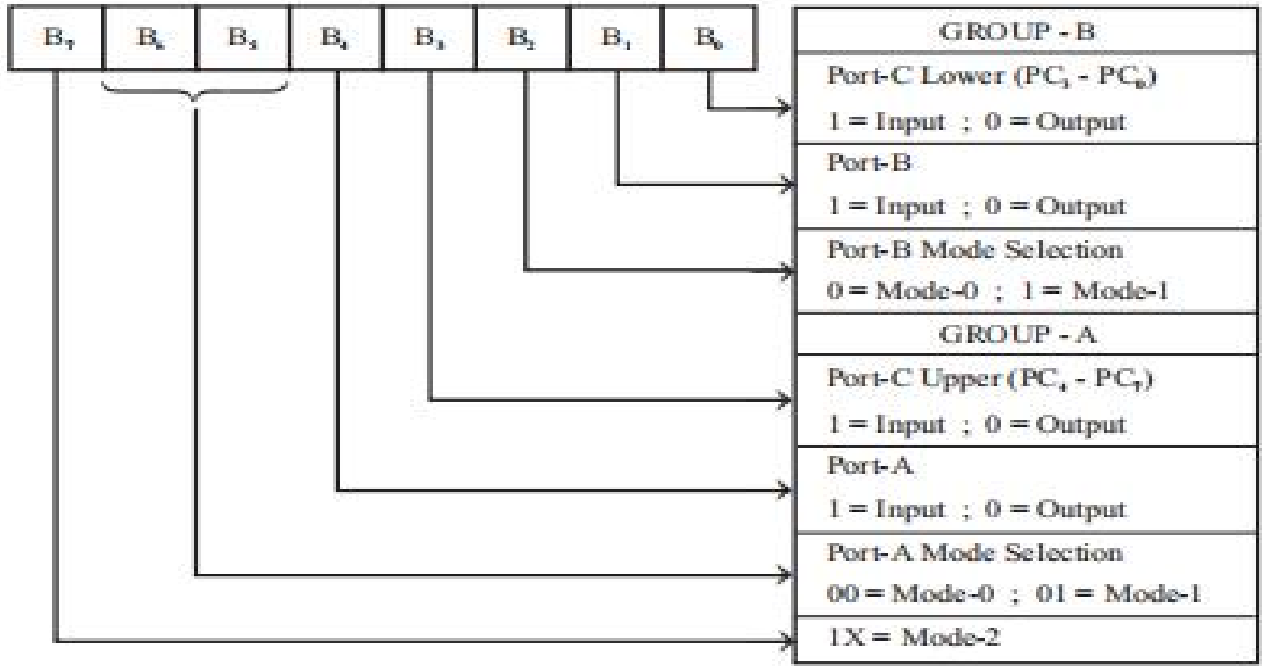


Fig. : Format of IO mode set control word of 8255.

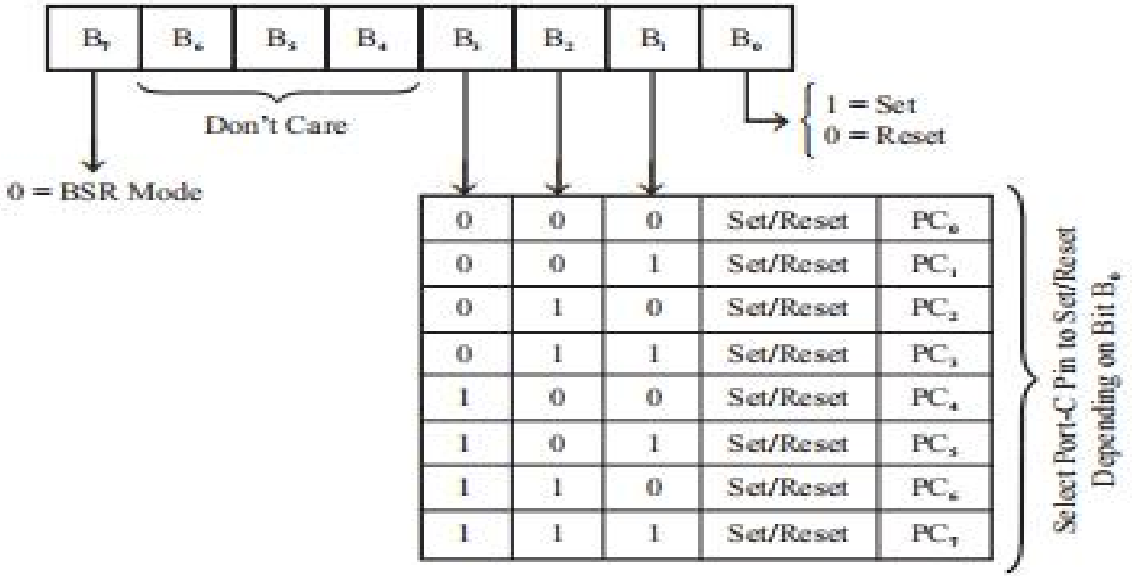


Fig. Format of Bit Set/Reset control word of 8255.

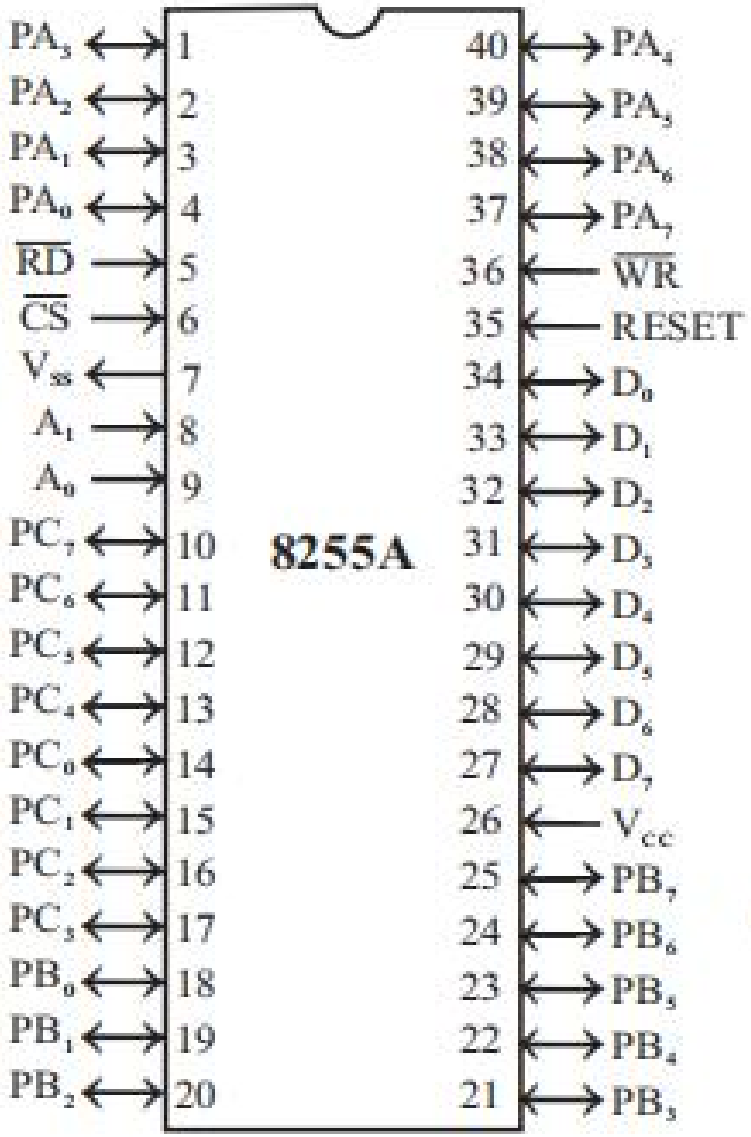
IO modes in 8255:

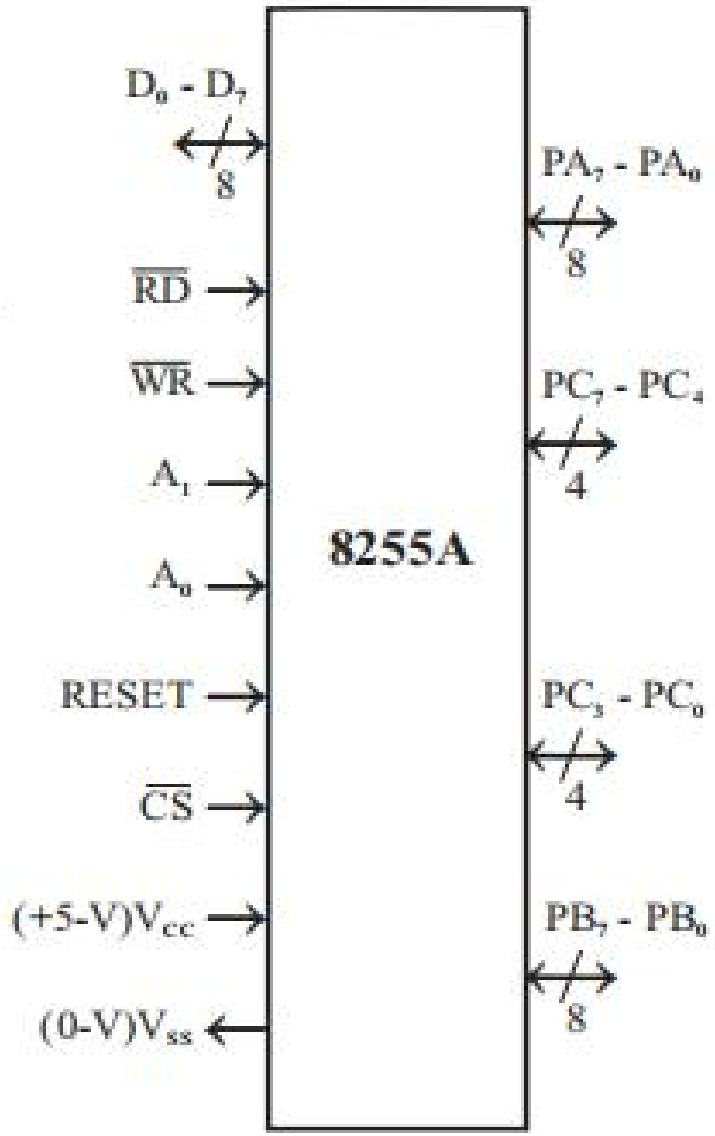
- The 8255 has three ports: Port-A, Port-B and Port-C. The ports A and B are 8-bit parallel ports. Port-A can be programmed to work in any one of the three operating modes as input or output port. The three operating modes are :
  - Mode-0 → Simple IO port.
  - Mode-1 → Handshake IO port.
  - Mode-2 → Bidirectional IO port.
- Port-B can be programmed to work either in mode-0 or mode-1 as input or output port.
- Port-C pins (8 pins) have different assignments depending on the mode of ports A and B. If ports A and B are programmed in mode-0, then port-C can perform any one of the following function :
  - 1. As 8-bit parallel port in mode-0 for input or output.
  - 2. As two numbers of 4-bit parallel port in mode-0 for input or output.
  - 3. The individual pins of port-C can be set or reset for various control applications.
- If port-A is programmed in mode-1/mode-2 and port-2 is programmed in mode-1 then some of the pins of port-C are used for handshake signals and the remaining pins can be used as input/ output lines or individually set/reset for control applications.

IO Modes of 8255

- Mode-0 : In this mode, all the three ports can be programmed either as input or output port. In mode-0, the outputs are latched and the inputs are not latched. The ports do not have handshake or interrupt capability. The ports in mode-0 can be used to interface DIP switches, Hexa-keypad, LEDs and 7-segment LEDs to the processor.
- Mode-1 : In this mode, only ports A and B can be programmed either as input or output port. In mode-1, handshake signals are exchanged between the processor and the peripherals prior to data transfer. The port-C pins are used for handshake signals. Input and output data are latched. Interrupt driven data transfer scheme is possible.
- Mode-2 : In this mode the port will be a bidirectional port (i.e., the processor can perform both read and write operations with an IO device connected to a port in mode-2).Only port-A can be programmed to work in mode-2. Five pins of port-C are used for handshake signals. This mode is used primarily in applications such as data transfer between two computers or floppy disk controller interface.

Pin Diagram of 8255:





Pin	Description
D_0 - D_7	Data Lines
RESET	Reset Input
$\overline{CS}$	Chip Select
$\overline{RD}$	Read Control
$\overline{WR}$	Write Control
A_0, A_1	Internal Address
PA_7 - PA_0	Port-A Pins
PB_7 - PB_0	Port-B Pins
PC_7 - PC_0	Port-C Pins
V_{CC}	+5-V
V_{SS}	0-V (GND)

Fig. Pin description of 8255.

Port C pin assignments:

TABLE - PORT-C PIN ASSIGNMENTS

Functions of Ports A and B	PC_7	PC_6	PC_5	PC_4	PC_3	PC_2	PC_1	PC_0
Ports A and B in mode-0 Input/Output	IO	IO	IO	IO	IO	IO	IO	IO
Ports A and B in mode-1 Input ports	IO	IO	IBF_A	$\overline{STB}_A$	$INTR_A$	$\overline{STB}_B$	IBF_B	$INTR_B$
Ports A and B in mode-1 Output ports	$\overline{OBF}_A$	$\overline{ACK}_A$	IO	IO	$INTR_A$	$\overline{ACK}_B$	$\overline{OBF}_B$	$INTR_B$
Port-A in mode-2 Port-B in mode-0	$\overline{OBF}_B$	$\overline{ACK}_A$	IBF_A	$\overline{STB}_A$	$INTR_A$	IO	IO	IO

IO	-	Input /Output line	$\overline{OBF}$	-	Output Buffer Full
$\overline{STB}$	-	Strobe	$\overline{ACK}$	-	Acknowledge
IBF	-	Input Buffer Full	The subscript A denotes port-A signal.		
INTR	-	Interrupt Request	The subscript B denotes port-B signal.		

Interfacing of 8255 to 8085:

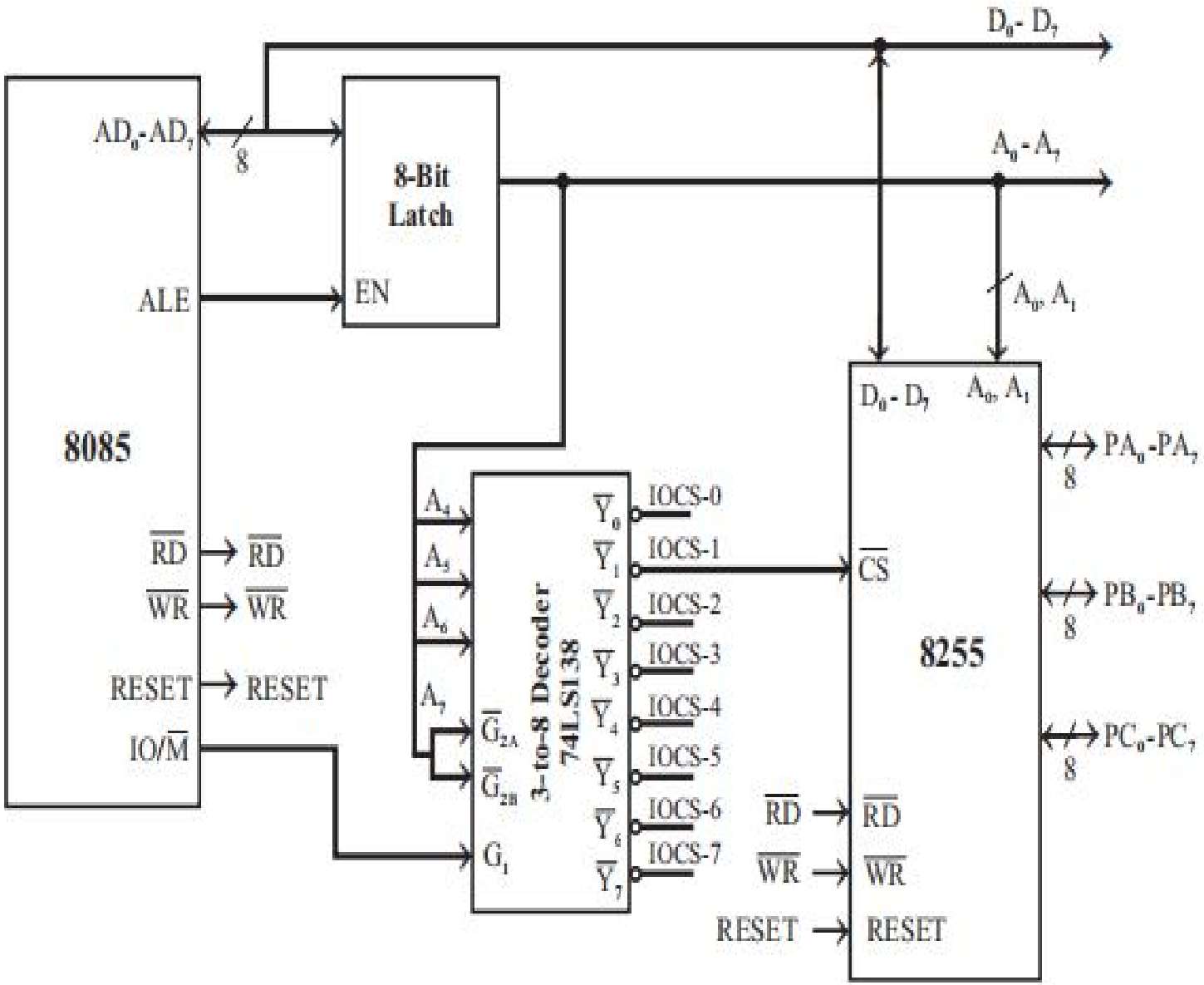


Fig.: Interfacing 8255 with 8085 processor.

- The address line A0 of 8085 is connected to A0 of 8255 and A1 of 8085 is connected to A1 of 8255 to provide the internal addresses. The IO addresses allotted to the internal devices of 8255 are listed in Table. The data lines D0 -D7 are connected to D0 -D7 of the processor to achieve paralleldata transfer. IO/ M is made high and connected to active high pin of decoder to ensure IO mapping else IO/ M is active low for Memory mapping.

TABLE - IO ADDRESSES OF 8255

Internal device	Binary address								Hexa address
	Decoder input and enable				Input to address pins of 8255				
	A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0	
Port-A	0	0	0	1	x	x	0	0	10
Port-B	0	0	0	1	x	x	0	1	11
Port-C	0	0	0	1	x	x	1	0	12
Control Register	0	0	0	1	x	x	1	1	13

Note : Don't care "x" is considered as zero.

Thank You

MPES
Module 3_4

Memory Mapping and IO Mapping of IO devices with 8085:

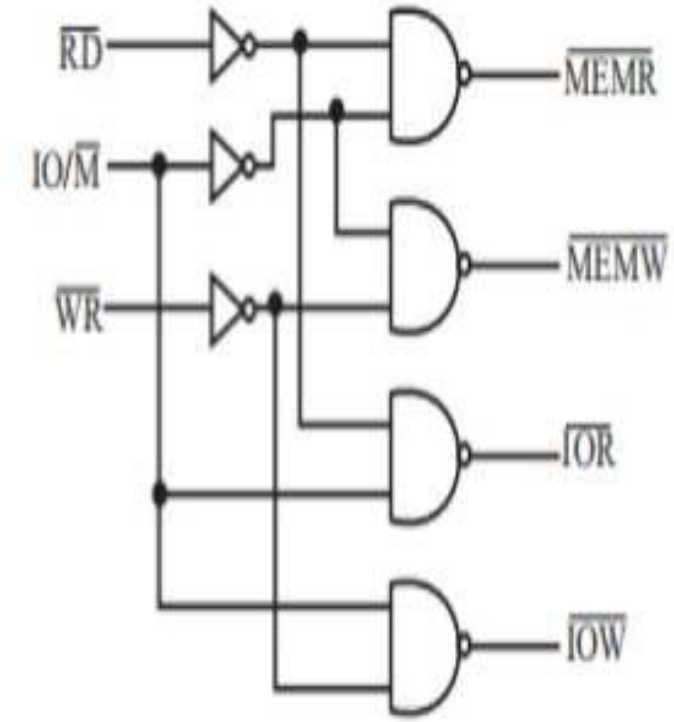
- The port and peripheral devices will have one logic low/high chip select pin. The processor can access the port/peripheral device by supplying internal address and chip select signals. Therefore, the port and peripheral device interfacing (IO interfacing) deals with allocation of various internal addresses and generation of chip select signals.
- There are two ways of interfacing IO devices in 8085-based system.
    - Memory-mapped IO device.
    - Standard IO-mapped IO device or Isolated IO mapping

In memory mapping of IO devices the ports are allotted a 16-bit address like that of the memory location. Some of the chip select signals generated to select memory ICs are used for selecting the IO port devices. Hence, the processor treats the IO ports as memory locations for reading and writing (i.e., the devices which are mapped by memory mapping are accessed by executing memory read cycle or memory write cycle).

In standard IO mapping or isolated IO mapping, a separate 8-bit address is allotted for the IO ports and the peripheral ICs. The processor differentiates the IO-mapped devices, from the memory-mapped devices in the following ways:

- For accessing the IO-mapped devices the processor executes IO read or write cycle.
- During IO read or write cycle, the 8-bit address is placed on both low order address lines and the high order address lines.
- IO/$\overline{M}$ is asserted high to indicate the IO operation (for read as well as write).

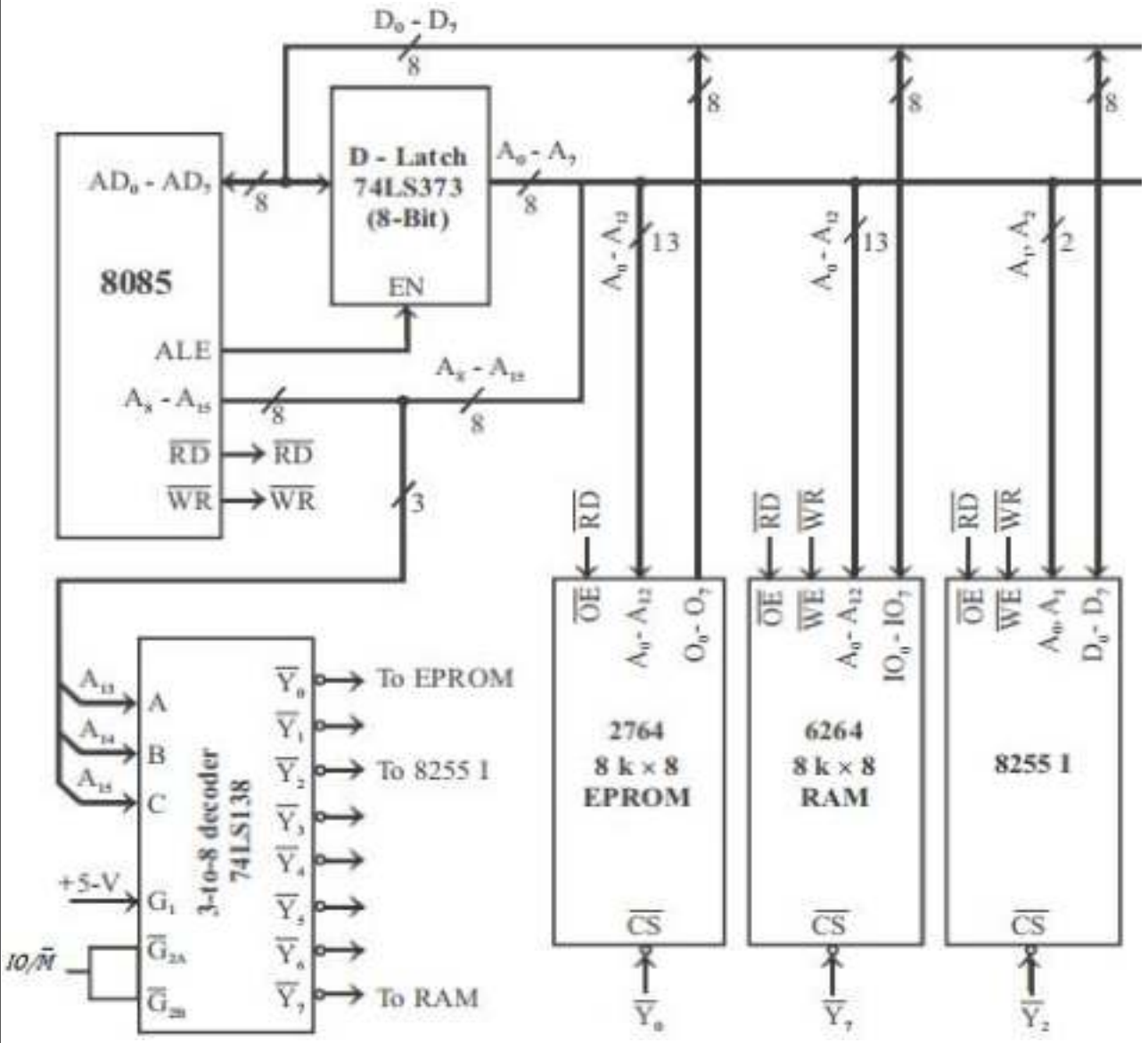
- A 8085 processor does not provide separate read (RD) and write (WR) signals for memory and IO devices. But it differentiates the memory and IO device accessed by IO/$\overline{M}$ signal. The three signals RD, WR and IO/$\overline{M}$ can be decoded as shown in Fig. to provide separate read and write control signals for IO devices and memory devices.
- When the devices are IO-mapped, then only IN and OUT instructions have to be used for data transfer between the device and the processor. For the IO-mapped devices a separate decoder should be used to generate the required chip select signals.



: Circuit to generate separate read and write signals for memory and IO devices in an 8085-based system.

Memory mapping of IO device	IO mapping of IO device
1. 16-bit addresses are provided for IO devices.	1. 8-bit addresses are provided for IO devices.
2. The devices are accessed by memory read or memory write cycles.	2. The devices are accessed by IO read or IO write cycle. During these cycles, the 8-bit address is available on both low order address lines and high order address lines.
3. The IO ports or peripherals can be treated like memory locations and so all instructions related to memory can be used for data transfer between the IO device and the processor.	3. Only IN and OUT instructions can be used for data transfer between the IO device and the processor.
4. In memory-mapped ports, the data can be moved from any register to the ports and vice versa.	4. In IO-mapped ports, the data transfer can take place only between the accumulator and the ports.
5. When memory mapping is used for IO devices, the full memory address space cannot be used for addressing memory. Hence memory mapping is useful only for small systems, where the memory requirement is less.	5. When IO mapping is used for IO devices, then the full memory address space can be used for addressing the memory. Hence it is suitable for systems which requires a large memory capacity.
6. In memory-mapped IO devices, a large number of IO ports can be interfaced.	6. In IO mapping, only 256 ports ($2^8 = 256$) can be interfaced.
7. For accessing memory-mapped devices, the processor executes the memory read or write cycle. During this cycle, IO/$\overline{M}$ is asserted **low** (IO/$\overline{M}$= 0).	7. For accessing the IO-mapped devices, the processor executes the IO read or write cycle. During this cycle, IO/$\overline{M}$ is asserted **high** (IO/$\overline{M}$=1).

Example for Memory mapping of IO devices:



Device	Binary address																Hexa address
	Decoder input			Input to address pins of memory/8255													
	A_{15}	A_{14}	A_{13}	A_{12}	A_{11}	A_{10}	A_9	A_8	A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0	
2764 EPROM	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0000
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0001
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0002
	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1FFF
6264 RAM	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	E000
	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	1	E001
	1	1	1	0	0	0	0	0	0	0	0	0	0	0	1	0	E002
	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	FFFF
8255 I																	
Port-A	0	1	0	X	X	X	X	X	X	X	X	X	X	0	0	X	4000
Port-B	0	1	0	X	X	X	X	X	X	X	X	X	X	0	1	X	4002
Port-C	0	1	0	X	X	X	X	X	X	X	X	X	X	1	0	X	4004
Control register	0	1	0	X	X	X	X	X	X	X	X	X	X	1	1	X	4006

Example for IO mapping for IO devices with 8085:

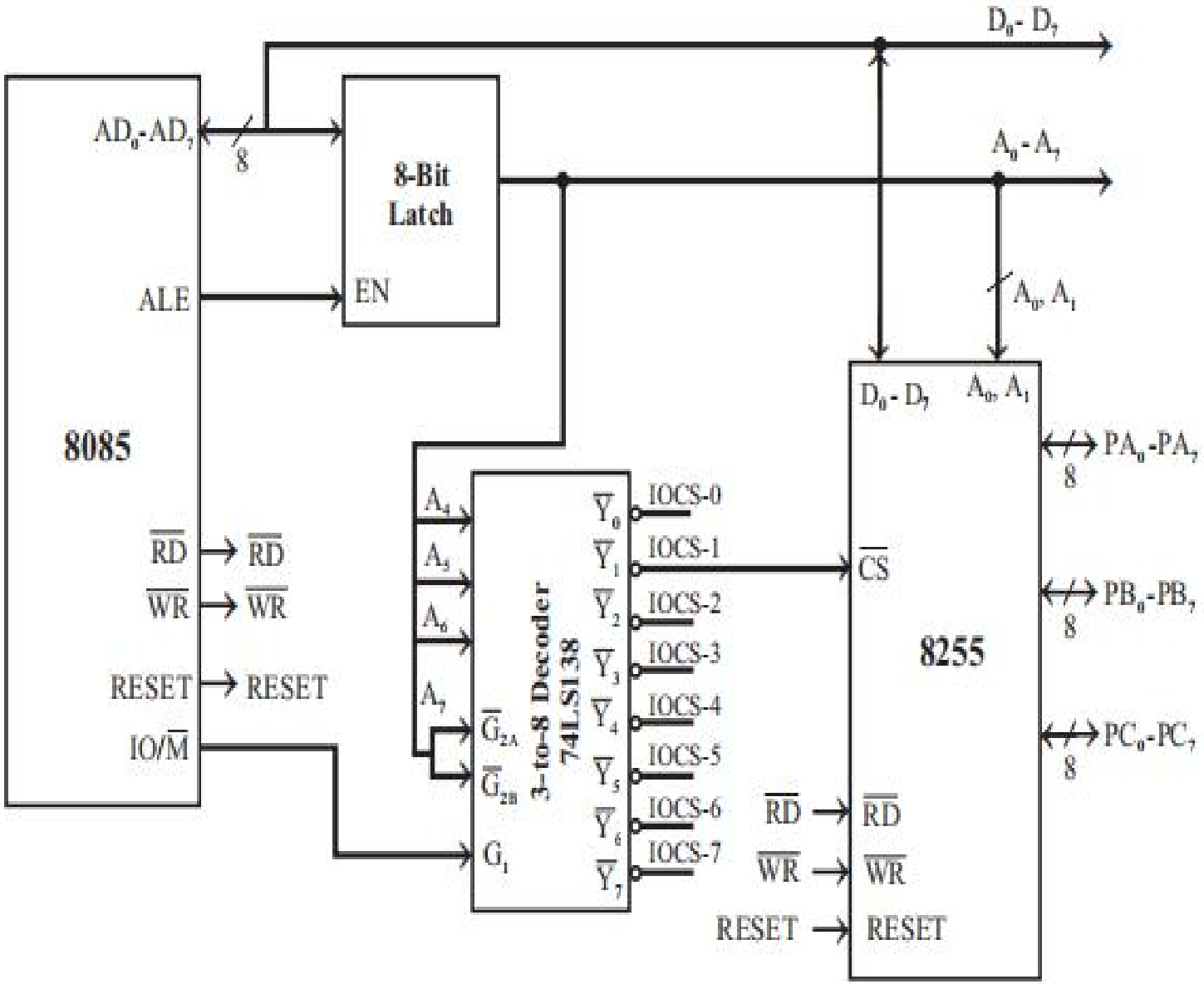


Fig.: Interfacing 8255 with 8085 processor.

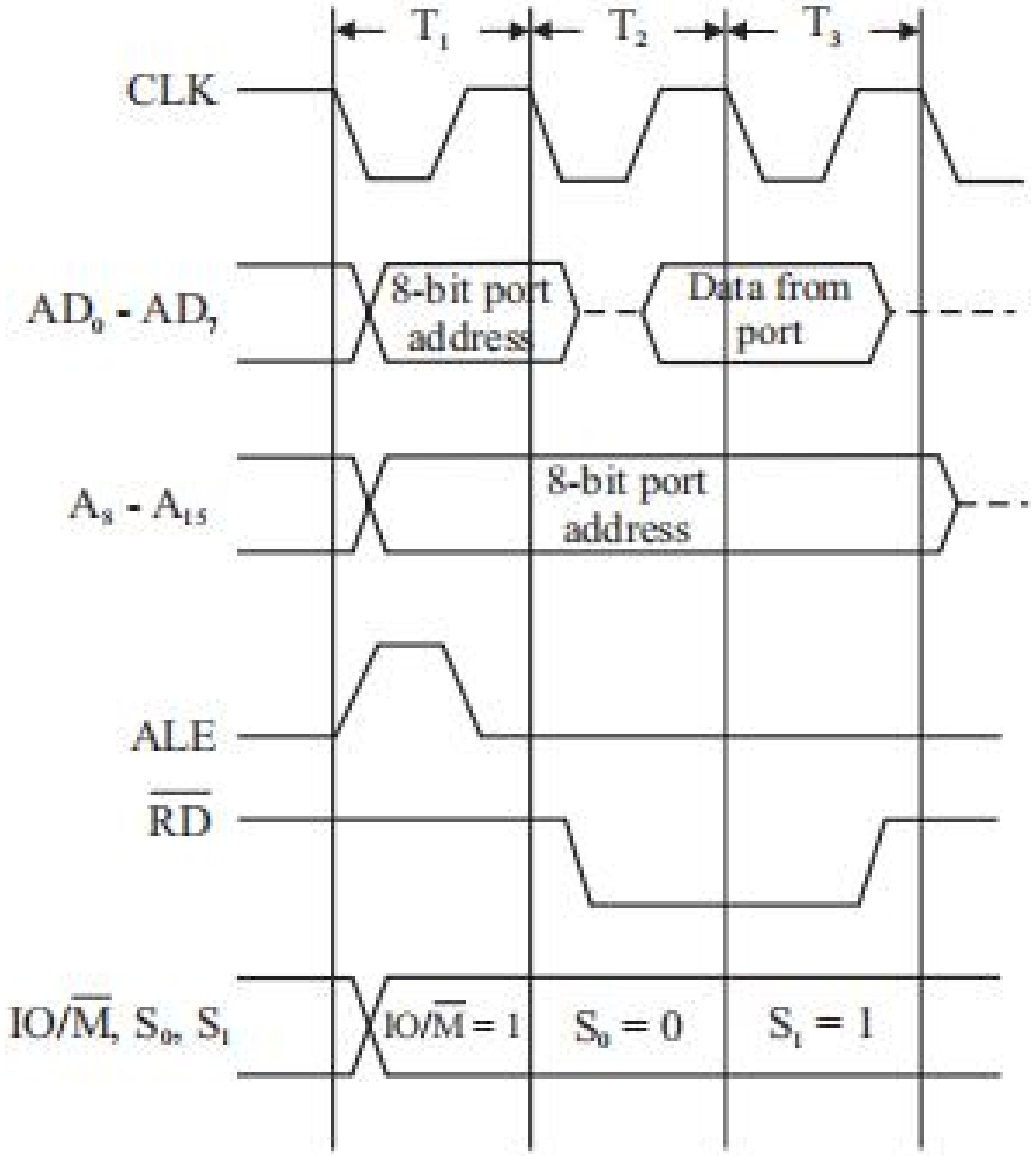
- The address line A0 of 8085 is connected to A0 of 8255 and A1 of 8085 is connected to A1 of 8255 to provide the internal addresses. The IO addresses allotted to the internal devices of 8255 are listed in Table. The data lines D0 -D7 are connected to D0 -D7 of the processor to achieve paralleldata transfer. IO/ $\overline{M}$ is made high and connected to active high pin of decoder to ensure IO mapping else IO/ $\overline{M}$ is active low for Memory mapping.

TABLE - IO ADDRESSES OF 8255

Internal device	Binary address								Hexa address
	Decoder input and enable				Input to address pins of 8255				
	A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0	
Port-A	0	0	0	1	x	x	0	0	10
Port-B	0	0	0	1	x	x	0	1	11
Port-C	0	0	0	1	x	x	1	0	12
Control Register	0	0	0	1	x	x	1	1	13

Note : Don't care "x" is considered as zero.

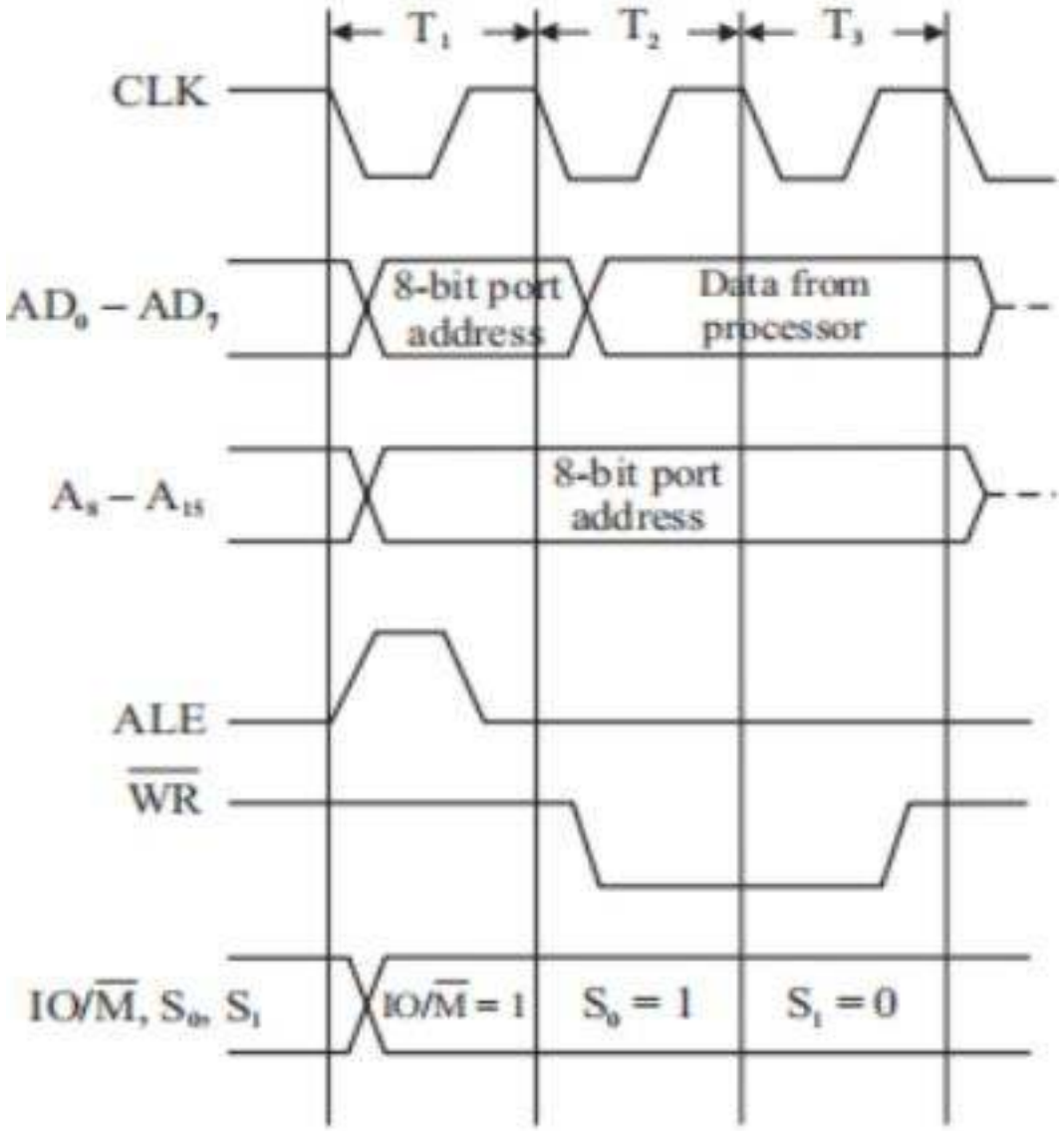
IO Read Machine Cycle:



(WR will be **high** ; READY is tied **high** either permanently or temporarily in the system.)

- At the falling edge of T1, the microprocessor output the 8- bit port address on both the low order address lines (AD0 -AD7) and high order address lines (A8 to A15). ALE is asserted high to enable the external address latch. The other control signals are asserted as follows. IO/M=1, S0 = 0 and S1 = 1. (IO/M is asserted high to indicate IO access.)
- At the middle of T1, the ALE is asserted low and this enables the external address latch to take the port address and keep on its output lines.
- In the second T-state (T2) the IO device is requested for read by asserting read line low. When read is asserted low, the IO port is enabled for placing the data on the data bus. The time allowed for IO port to output the data is the time during which read remains low.
- At the end of T3, the read signal is asserted high. On the rising edge of read signal the data is latched into microprocessor. Other control signals remains in the same state until the next machine cycle.
- IN instruction is used.

IO write Machine Cycle:



(RD will be **high** ; READY is tied **high** either permanently or temporarily in the system.)

- At the falling edge of T1, the microprocessor outputs the 8-bit port address on low order address line(AD0 - AD7) and high order address lines (A8 to A15).
- ALE is asserted high to enable the external address latch. The other control signals are asserted as follows : IO/M=1, S0 = 1 and S1 = 0. (IO/M is asserted high to indicate IO access.)2.
- At the middle of T1, the ALE is asserted low and this enables the external address latch for latching the port address into its output lines.
- In the falling edge of T2, the processor output data on AD0 - AD7 lines and then request IO port for write operation by asserting the write control signal WR to low.
- At the end of T3 , the processor asserts WR high. Thisenables the IO port to latch the data into it. The IO port should prepare itself to accept the data within the time duration in which write control signal remains low. Other control signals remains in the same state until the next machine cycle.
- OUT instruction is used.

Thank You

MPES
Module 3_5

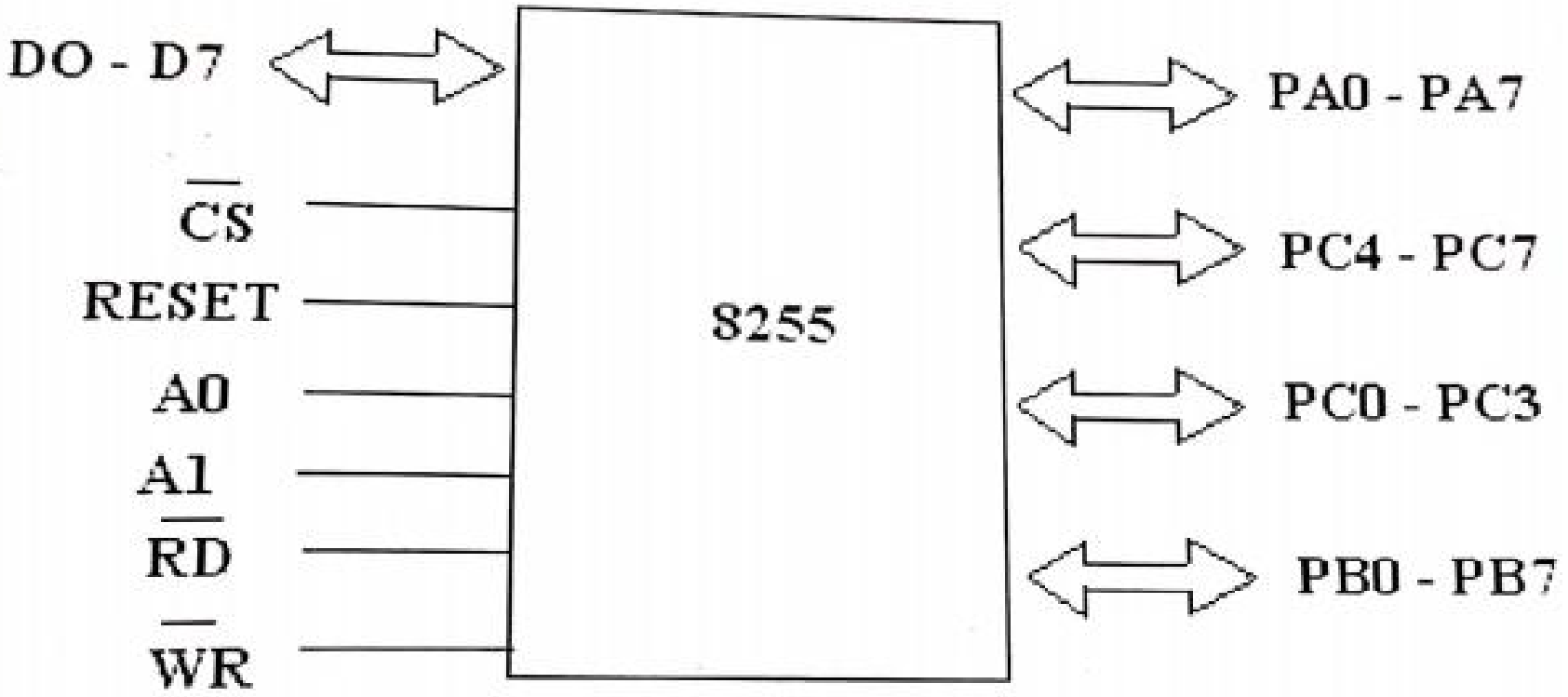
LED interfacing with 8085:

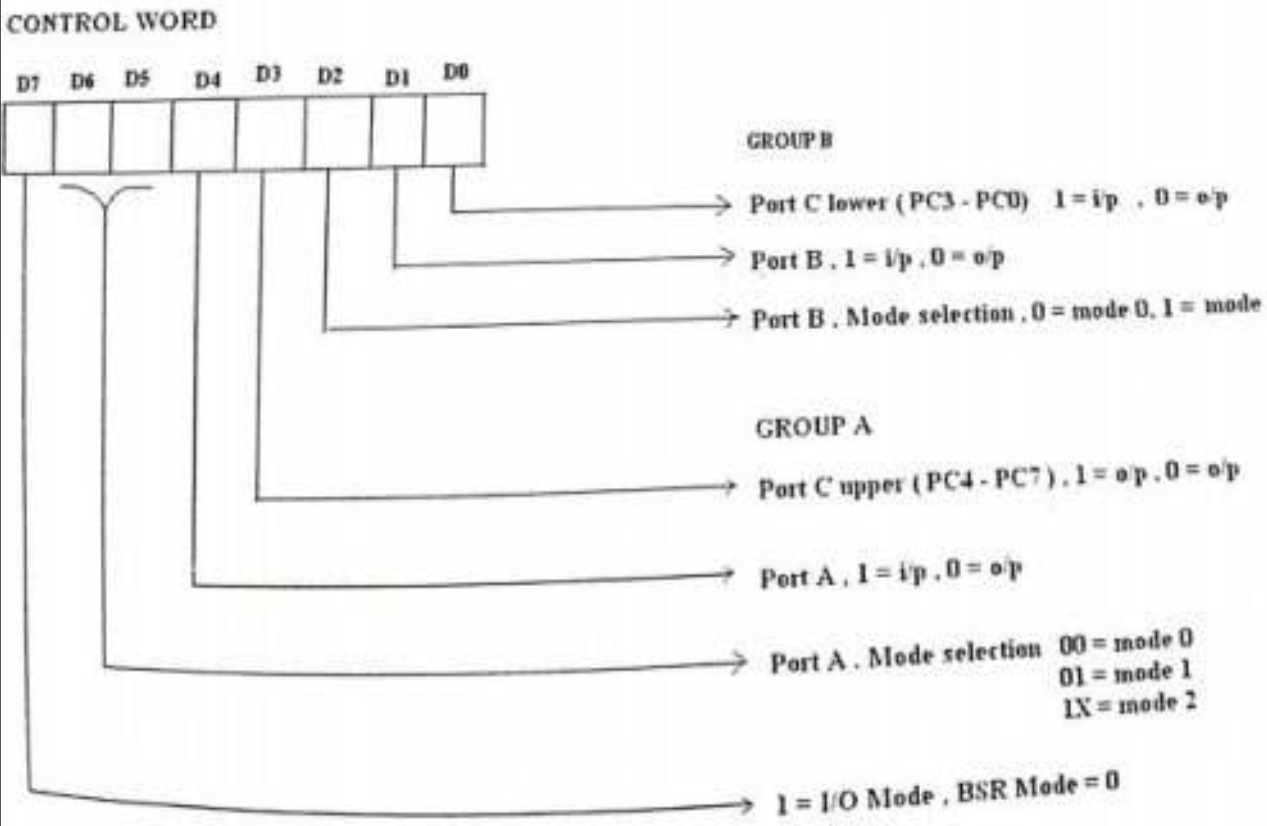
Program port A as input and port B as output. Read port A and display at port B.

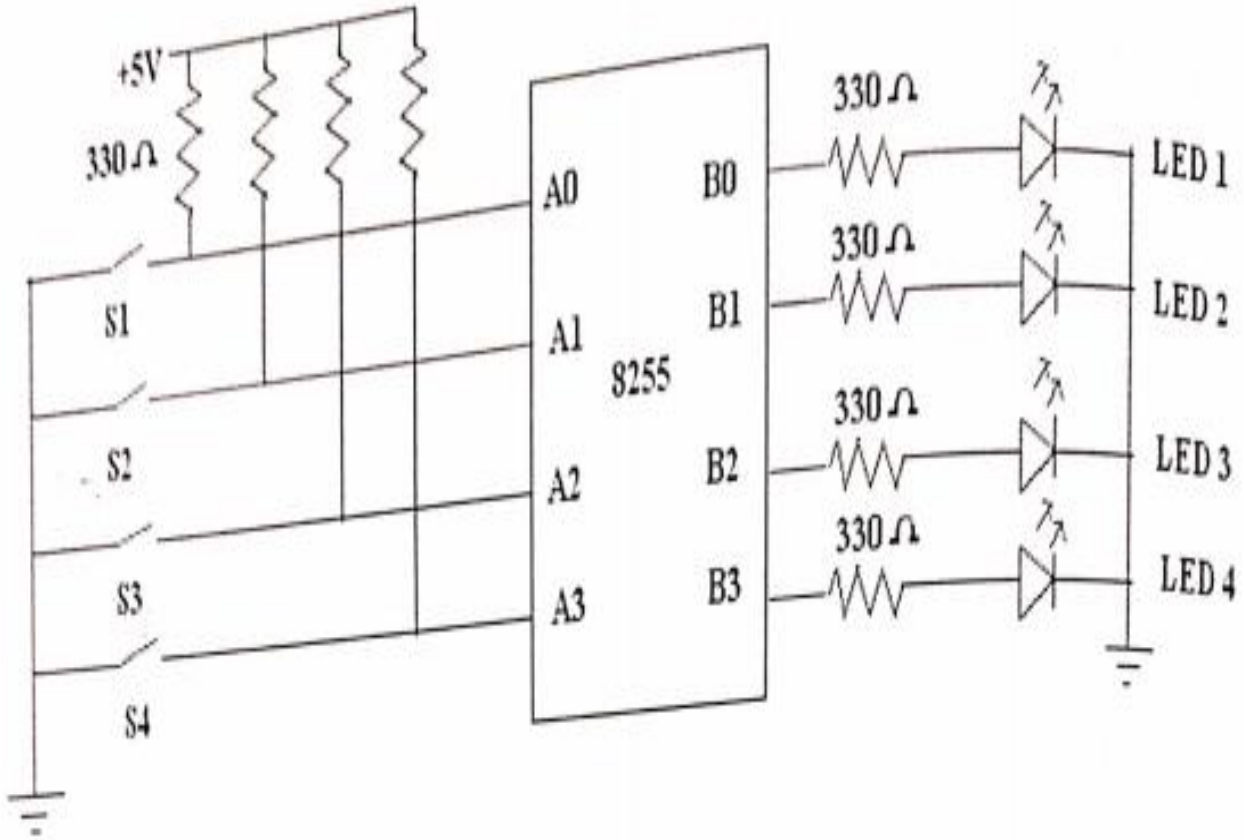
CONTROL WORD:

1	0	0	1	0	0	0	0	$\rightarrow 90_H$

THEORY




CONTROL WORD
D7 D6 D5 D4 D3 D2 D1 D0
GROUP B
Port C lower (PC3 - PC0) 1 = i/p , 0 = o/p
Port B , 1 = i/p , 0 = o/p
Port B , Mode selection , 0 = mode 0, 1 = mode
GROUP A
Port C upper (PC4 - PC7) , 1 = o/p , 0 = o/p
Port A , 1 = i/p , 0 = o/p
Port A , Mode selection 00 = mode 0
01 = mode 1
1X = mode 2
1 = I/O Mode , BSR Mode = 0


+5V
330Ω
S1
S2
S3
S4
A0
A1
A2
A3
8255
B0
B1
B2
B3
330Ω
330Ω
330Ω
330Ω
LED 1
LED 2
LED 3
LED 4

PROGRAM

Address	Hex code	Label	Mnemonics & Operands	Comments
6000	3E		MVI A, 90_H	CW is moved to A
6001	90			
6002	D3		OUT 03	Out CW to control register
6003	03			
6004	DB	START	IN 00	Read port A
6005	00			
6006	D3		OUT 01	Display port B
6007	01			
6008	C3		JMP START	
6009	04			
600A	60			

Thank You

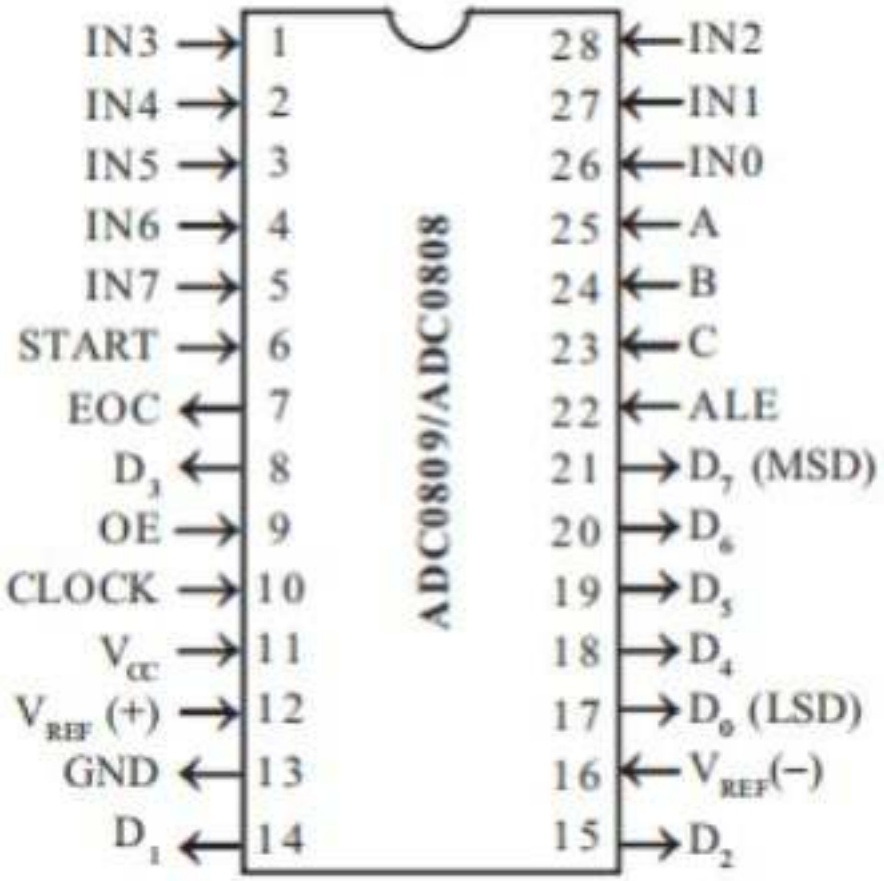
MPES
Module 3_6

ADC interfacing with 8085:

- In many applications, an analog device has to be interfaced to digital system. But, the digital devices cannot accept the analog signals directly. So, the analog signals are converted to equivalent digital signal (data) using Analog-to-Digital Converter (ADC).
- TheA/D conversion is the process by which the analog signal is represented by an equivalent binary data
- If the digital data is represented by n-bit binary then it can have 2^n different values.
- In A/D conversion the given analog signal has to be divided into steps of 2^n values, and each step is represented by one of the 2^n values.
- The resolution of the converter is the minimum analog value that can be represented by the digital data. If the ADC gives n-bit digital output and the full scale analog input is X volts, then the resolution is X / 2^n volts.
- The conversion time is defined as the total time required to convert an analog signal into its digital equivalent.

ADC 0808/ 0809

The ADC0809/0808 is an 8-bit ADC with an inbuilt 8-channel multiplexer.
The ADC0809/0808 is available as a 28-pin IC in DIP (Dual In-line Package).
The analog to digital converter is treated as an input device by the microprocessor.



SIGNAL DESCRIPTION OF ADC0809/ADC0808

Signals	Description
IN0-IN7	Eight single ended analog input to ADC.
A, B, C	3-bit binary input to select one of the eight analog signals for conversion at any one time.
ALE	Address latch enable. Used to latch the 3-bit address input to an internal latch.
START	Start of conversion pulse input. To start ADC process this signal should be asserted **high** and then **low**. This signal should remain **high** for atleast 100 ns.
CLOCK	Clock input and the frequency of clock can be in the range of 10 kHz to 1280 kHz. Typical clock input is 640 kHz.
$V_{REF}(+), V_{REF}(-)$	Reference voltage input. The positive reference voltage can be less than or equal to V_{cc} and the negative reference voltage can be greater than or equal to ground.
D_0-D_7	The 8-bit digital output. The reference voltages will decide the mapping of analog input to digital data.
EOC	End of conversion. This signal is asserted **high** by the ADC to indicate the end of conversion process and it can be used as interrupt signal to processor.
OE	Output buffer Enable. This signal is used to read the digital data from output buffer after a valid EOC.
V_{cc}	Power supply, +5-V
GND	Power supply ground, 0-V

- Step Size or quantization step in ADC is ,

$$Q_{step} = \frac{V_{REF}}{2^8} = \frac{V_{REF}(+) - V_{REF}(-)}{256_{10}}$$

The digital data corresponding to an analog input (V_{in}) is given by,

$$\text{Digital data} = \left(\frac{V_{in}}{Q_{step}} - 1 \right)_{10}$$

Address input			Selected channel
C	B	A	
0	0	0	IN0
0	0	1	IN1
0	1	0	IN2
0	1	1	IN3
1	0	0	IN4
1	0	1	IN5
1	1	0	IN6
1	1	1	IN7

Input Channel selection based on ABC signals.

EXAMPLE

Let, $V_{REF}(+) = 3.84$ V, $V_{REF}(-) = 0$ V

$$\therefore\ Q_{step} = \frac{V_{REF}(+) - V_{REF}(-)}{256_{10}} = \frac{3.84}{256} = 0.015 \text{ V} = 15 \text{ mV}$$

Let the input analog voltage be 2.56 V. Now the digital data corresponding to 2.56 V is given by,

$$\text{Digital data} = \frac{V_{in}}{Q_{step}} - 1 = \frac{2.56}{0.015} - 1 = 169_{10} = A9_H = 1010\ 1001_2$$

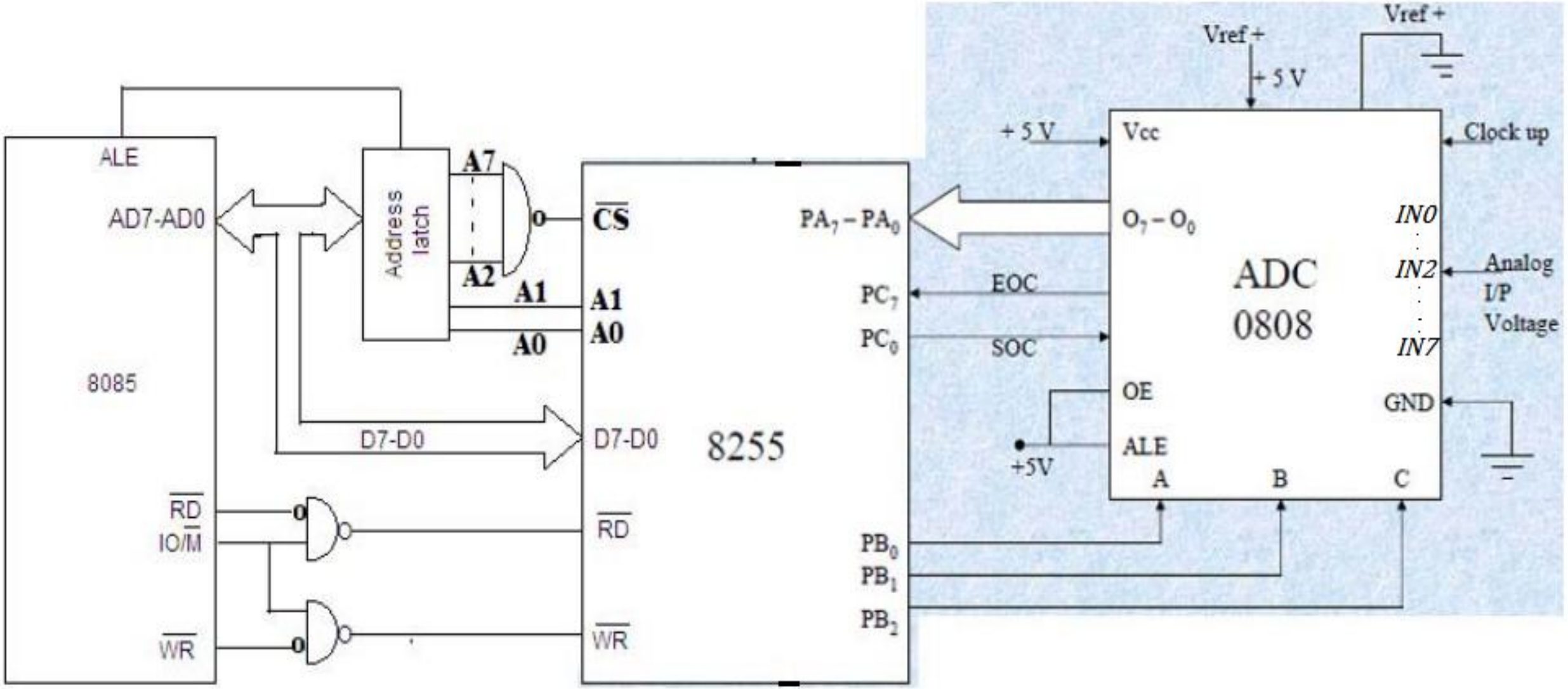


Figure: Interfacing ADC with 8085

- During the analog to digital conversion process, Initially, microprocessor sends an initializing signal (start of conversion-SOC) to the ADC to start the analog to digital data conversion process. The start of conversion signal is a pulse of a specific duration.
- The microprocessor has to wait for the digital data till the conversion is over. After the conversion is over, the ADC sends end of conversion EOC signal to inform the microprocessor that the conversion is over and the result is ready at the output buffer of the ADC.
- These tasks of issuing an SOC pulse to ADC, reading EOC signal from the ADC and reading the digital output of the ADC are carried out by the CPU using 8255 I/O ports.

Example: Interfacing ADC 0808 with 8085 using 8255 ports. Use port A of 8255 for transferring digital data output of ADC to the CPU and port C for control signals. Assume that an analog input is present at I/P2 of the ADC and a clock input of suitable frequency is available for ADC.

Solution: The analog input I/P2 is used and therefore address pins A,B,C should be 0,1,0 respectively to select I/P2.

The OE and ALE pins are already kept at +5V to select the ADC and enable the outputs.

Port C upper acts as the input port to receive the EOC signal while port C lower acts as the output port to send SOC to ADC .

Port A acts as a 8-bit input data port to receive the digital data output from the ADC.

- The 8255 control word is written as follows:

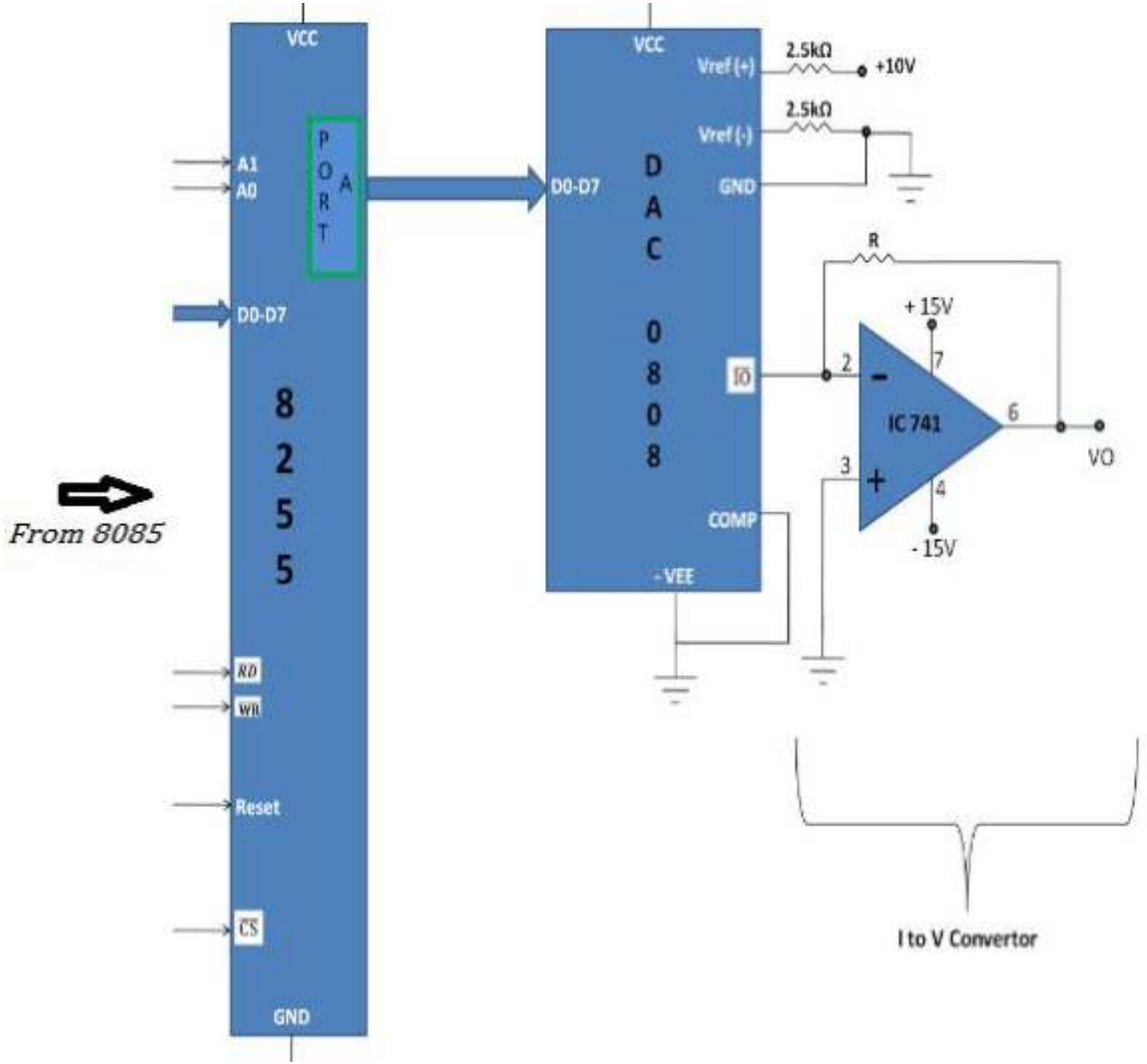
D7	D6	D5	D4	D3	D2	D1	D0
1	0	0	1	1	0	0	0

DAC Interfacing to 8085:

- In many applications, the microprocessor has to produce analog signals for controlling certain analog devices. Basically the microprocessor system can produce only digital signals. In order to convert the digital signal to analog signal a Digital-to-Analog Converter (DAC) has to be employed.
- The DAC will accept a digital (binary) input and convert to analog voltage or current.
- Every DAC will have "n" input lines and an analog output.
- The DAC requires a reference analog voltage (Vref) or current (Iref) source. The smallest possible analog value that can be represented by the n-bit binary code is called resolution. The resolution of DAC with n-bit binary input is $1/2^n$ of reference analog value. Every analog output will be a multiple of the resolution.
- For example, consider an 8-bit DAC with reference analog voltage of 5 volts.

  Now the resolution of the DAC is $(1/2^8) \times 5$ volts. The 8-bit digital input can take, $2^8 = 256$ different values.
- The analog values for all possible digital input are as shown in Table

Digital input	Analog output
0000 0000	$\frac{0}{2^8} \times 5$ Volts
0000 0001	$\frac{1}{2^8} \times 5$ Volts
0000 0010	$\frac{2}{2^8} \times 5$ Volts
0000 0011	$\frac{3}{2^8} \times 5$ Volts
1111 1111	$\frac{255}{2^8} \times 5$ Volts


VCC
A1
A0
PORT A
D0-D7
8255
RD
WR
Reset
CS
GND
From 8085
VCC
Vref (+)
2.5kΩ
+10V
Vref (-)
2.5kΩ
GND
D0-D7
DAC 0808
IO
COMP
-VEE
R
+15V
7
2
IC 741
6
VO
3
4
-15V
I to V Convertor

Thank You

MPES
Module 5_1

Embedded System:

An Embedded system is a system that has a software embedded in to a computer hardware for doing a dedicated task. It may be an independent system or a part of large system.

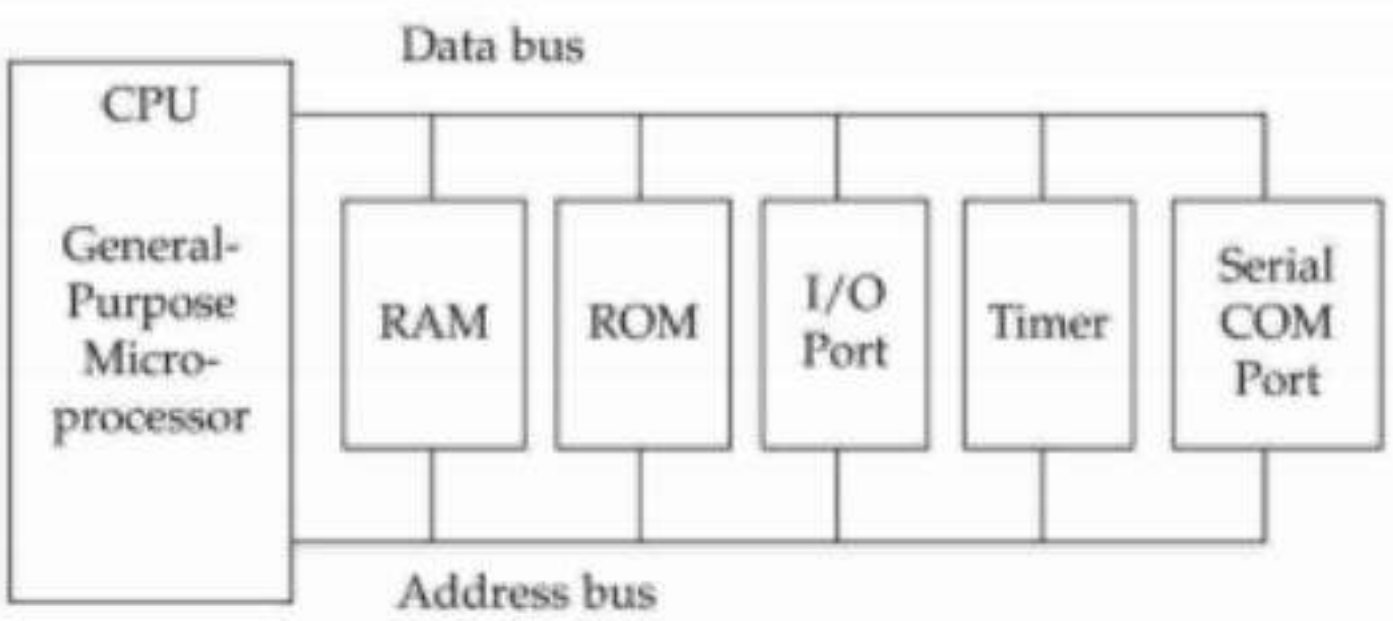
Eg: Mobile phones, TV remotes, Printers etc.

Characteristics of Embeddded systems:

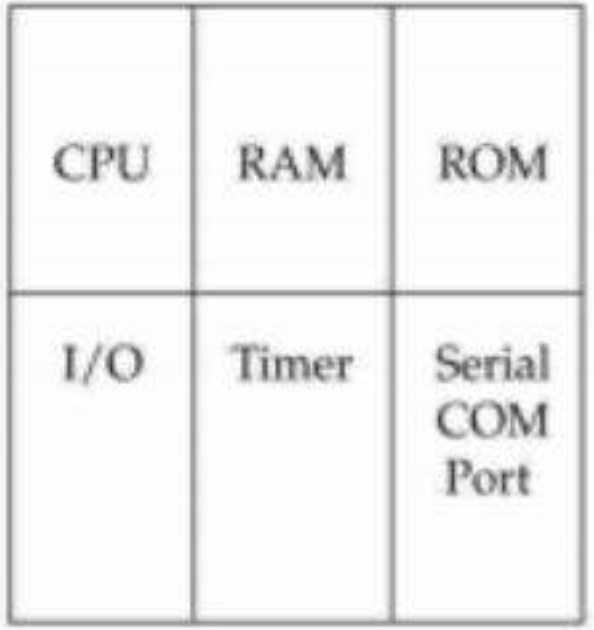
- Reliability
- Cost effectiveness
- Low power consumption
- Fast execution time
- Efficient use of memory
- Processing power is more

CPU of the embedded system can be Micro processor, Micro Controller or DSP or any Application Specific Processor (ASP).

Micro processor based System Vs Micro Controller based System:



(a) General-Purpose Microprocessor System

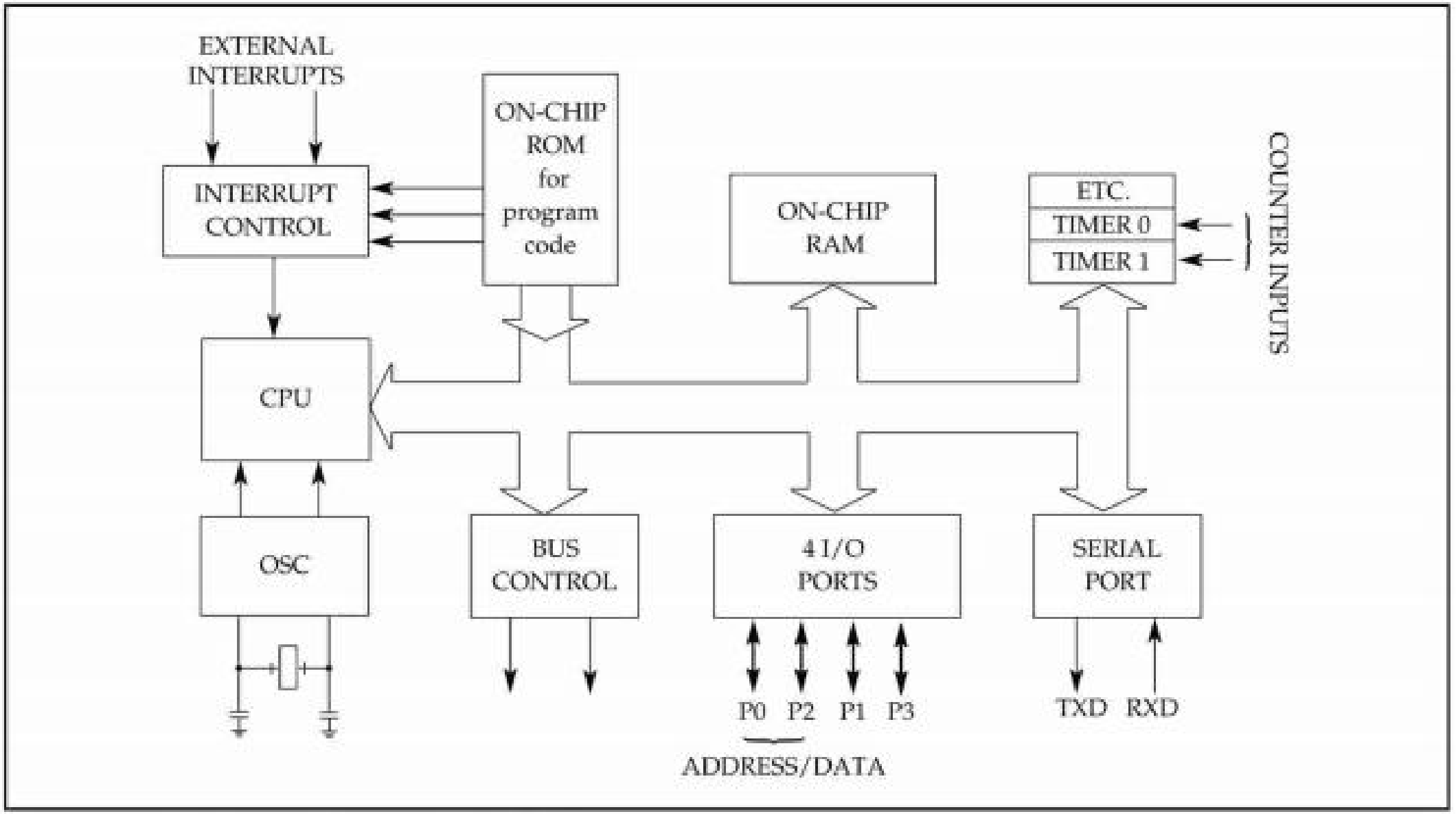


(b) Microcontroller

Comparison:

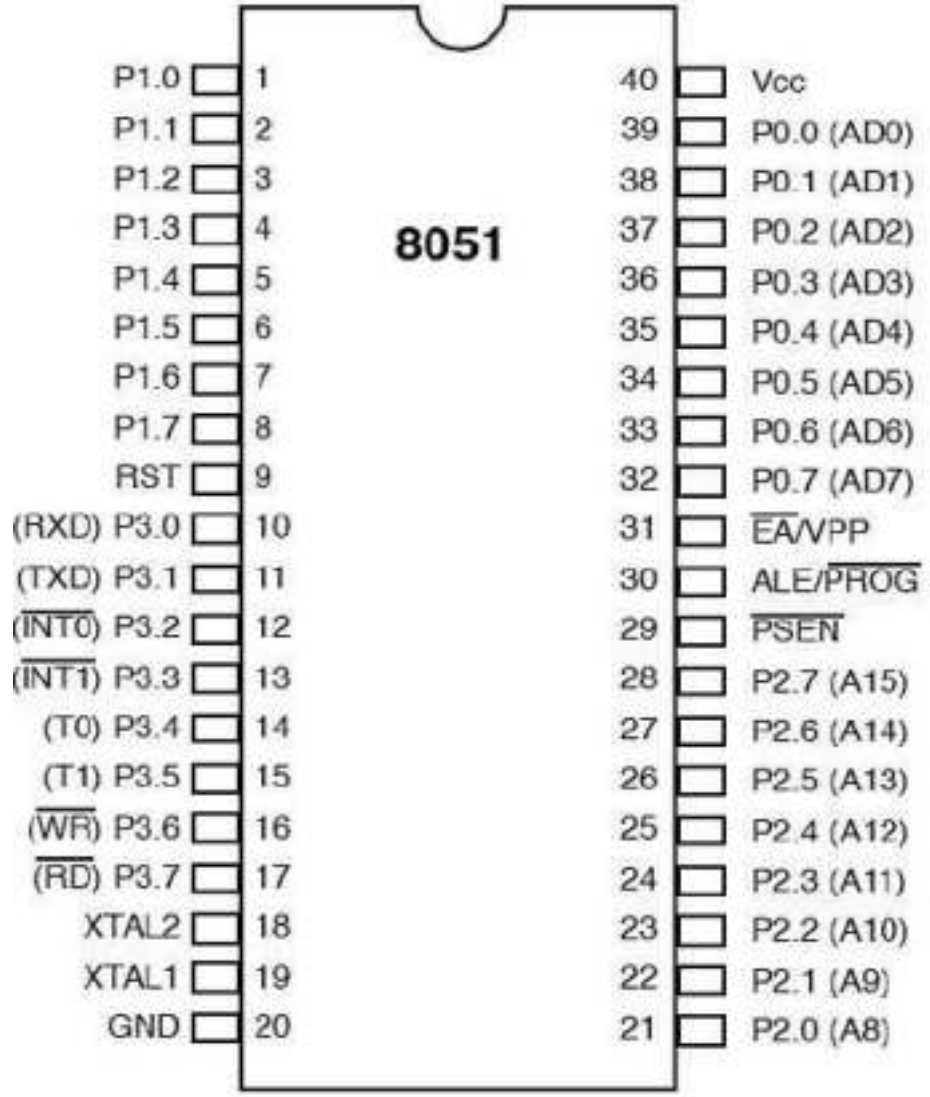
A Microprocessor does not contain RAM, ROM, I/O ports, Timer/ Counters,Serial Communication ports internal to it. So for a system these are to be to interfaced externally.	A Micro Controller contains the circuitary of Micro Processor and in addition to that it has built in ROM, RAM, I/O ports,Timer/ Counters,Serial Communication ports etc.
Bulkier System	Less Size
More time to design PCB as more Hardware	Les Time to design.
Expensive	Less Expensive
More versatile as the designer can choose the amount of memory and required peripherals.	Less versatile
Access time for Memory and IO devices are high	Less access time.
Less No. of pins are Multifuctional	More pins are Multi functional
More Instructions required to move data between memory and CPU	Less Instructions required to move data between memory and CPU
Less No. of bit handling instructions	More No. of bit handling instructions
8085, 8086, Pentium series etc	8051,8052, PIC, ARM etc..
Desktop, Laptop etc	Mobile phones,Printers, TV remotes etc..

Internal Architecture of 8051:



Features of 8051:

- 8 bit Micro controller
- 40 pin DIP
- 128 bytes of internal RAM
- 4K bytes of onchip ROM
- Two 16 bit Timers / Counters
- One Full duplex Serial port
- Four 8 bit I/O ports.
- 6 interrupt sources
- Harvard Architecture



Thank You

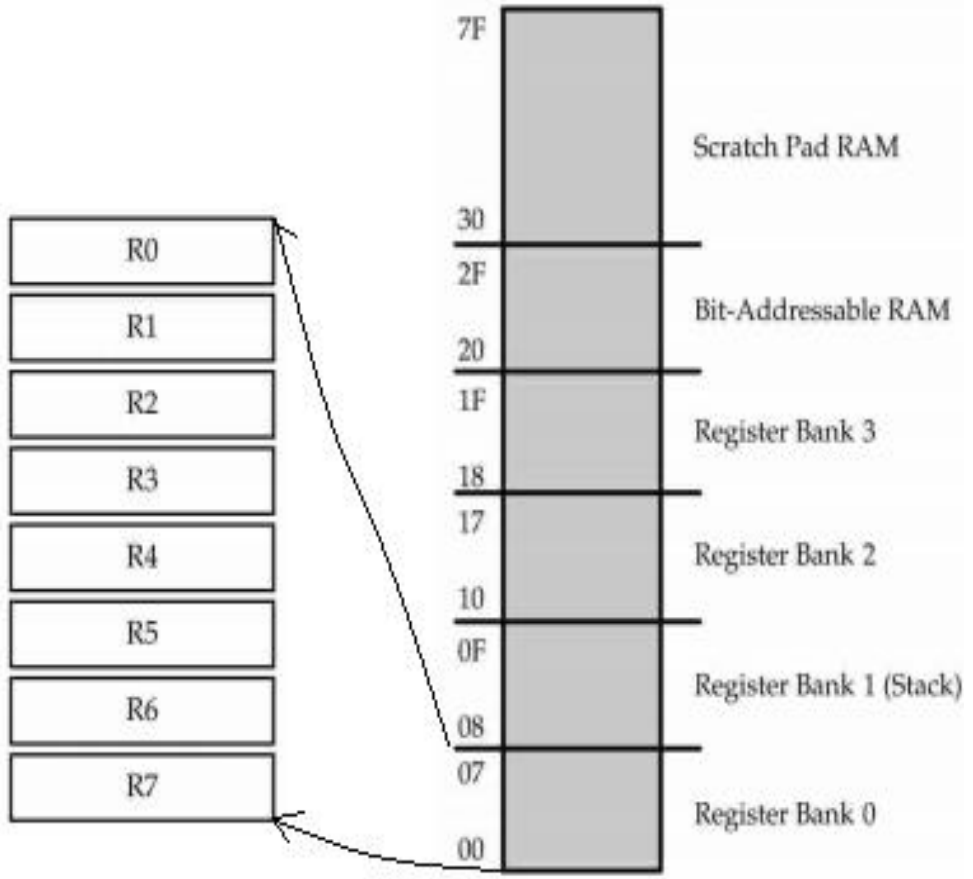
MPES
Module_5_2

Register Organisation in 8051:

- In 8051 , there is only one data type which is 8 bits.
- Since there is a large number of registers in 8051, they can be classified as General purpose registers and Special function registers.
- Most widely used registers of 8051 are, A [accumulator], B, R0 to R7 of each bank of registers. There are 4 such Banks of registers.
- These 32 registers (8 X 4 = 32) R0 to R7 of each bank, is known as General purpose registers .
- All of these general purpose registers are of 8 bit size.
- Registers A and B are used to hold results of mathematical and logical operations by CPU.
- Register B is used along with A for multiplication and division operations and has no other function other than as a location where data is stored.
- In addition to hold the operands and results, Register A is used for all data transfers between the 8051 and the external memory.
- Except Program Counter [PC] and DPTR register, all are 8 bit registers. PC and DPTR are 16 bit registers.

RAM memory space allocation in 8051:

- 128 bytes of RAM in 8051. ie, addresses drom 00 H to 7F H.
- The total RAM space is divided in to three.
  1. A total of 32 bytes from locations 00 H to 1F H are set aside for Register banks and Stack.
     - [0001 1111 = 1F H = 31 d is equivalent to 31 + 1 = 32 since we start from 00 H.]
     - These 32 registers are organised as Four register banks of eight registers each. R0 to R7 in each bank.
     - The four register banks are numbered from 0 to 3.
     - Each register can be addressed by its name (R0 to R7), if the bank is selected, else by its RAM address.
     - If the bank 3 is selected then R0 will point to R0 of bank 3, else address 18 H will point to the same location.
     - Bank selection can be done using the bits RS0 and RS1 in the program status word register. [PSW]
     - upon Reset Bank 0 is selected.



	RS1 [PSW.4]	RS0 [PSW.3]
Bank 0	0	0
Bank 1	0	1
Bank 2	1	0
Bank 3	1	1

Register bank selection using PSW bits

2. A total of 16 bytes from location 20 H to 2F H are set aside for Bit addressable Read/ Write memory.
   - If only one bit is needed, we can use the bit address ie, 00 H to 7F H along with instructions for bit operation.
   - If one byte is needed then address 20 H to 2F H along with instructions for byte operation.
3. A total of 80 bytes from locations 30 H to 7F H are used for read / write storage operations known as scratch pad RAM.

STACK in 8051:

Stack is a section of RAM used by the CPU to store information temporarily. the information could be data or address.

- The register used to point towards the stack is called Stack pointer which is of 8 bit in 8051. The address held in the SP is the location in internal RAM where the last byte of data was stored by the stack operation.
- When the 8051 is powered up, the SP points to the location 07 H, (which can be changed with in the program.) This means that the RAM location 08 H is the first location being used as stack of 8051 by default. ie, in Bank 1.
- Generally locations 08 H to 1F H can used as stack. If a given program needs more stack, we can change the SP to RAM locations 30 H to 7F H using the program instructions. Can't use locations 20 H to 2F H, as it is reserved for bit addressable memory.

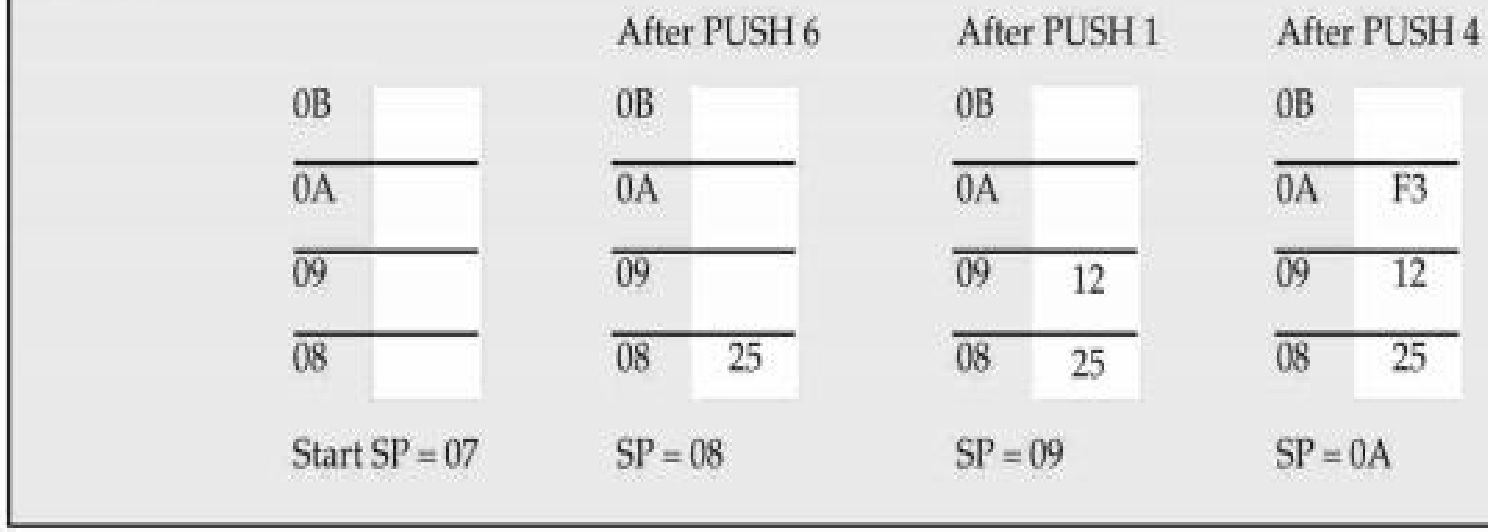
PUSHing in to the Stack:

- When we PUSH data in to the stack, the SP is incremented by one before storing the data in to the stack, so that stack grows up as data is stored.

Show the stack and stack pointer for the following. Assume the default stack area and register 0 is selected.

```
MOV     R6, #25H
MOV     R1, #12H
MOV     R4, #0F3H
PUSH    6
PUSH    1
PUSH    4
```

Solution:



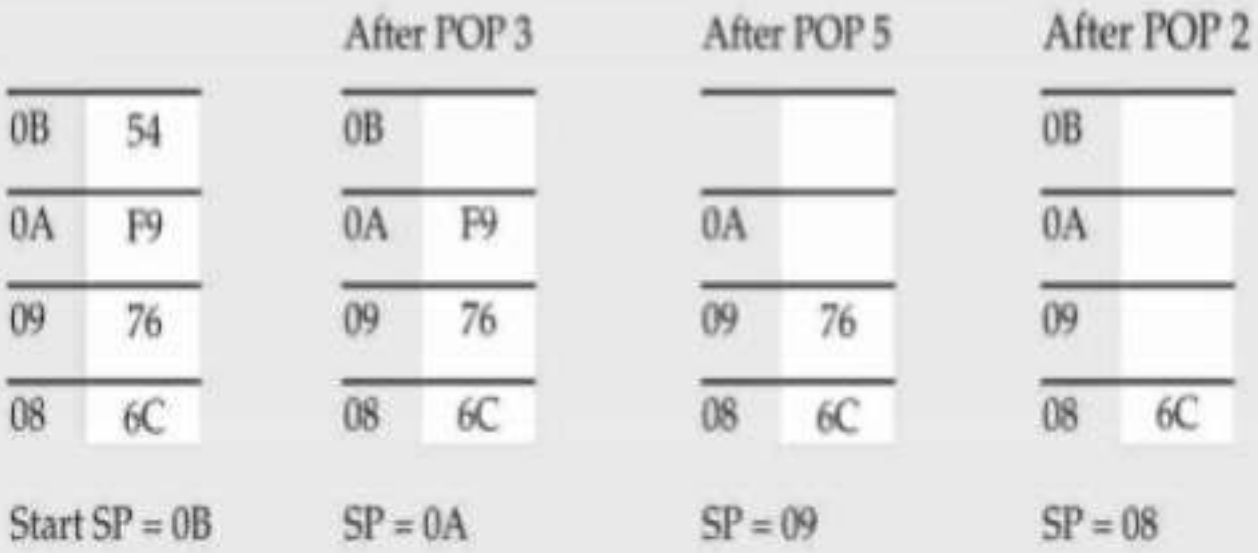
POPping from Stack:

- With every POP or retrieval, top byte of stack [LIFO] is copied to the register specified in the instruction and the the SP is decremented by one.

Examining the stack, show the contents of the registers and SP after execution of the following instructions. All values are in hex.

```
POP 3     ;POP stack into R3
POP 5     ;POP stack into R5
POP 2     ;POP stack into R2
```

Solution:



- The CPU can also use the use the stack to save the address of the instruction that comes just after the CALL instruction so that the CPU can know where to resume after the execution of the called sub routine.

Stack and Bank 1 Conflict:

- When 8051 is powered up, SP = 07 H. There fore the first location of stack is RAM location 08 H which belongs R0 of register bank 1. ie, Register bank 1 and Stack is using the same memory space. So if in a program we need to use register bank 1 and 2 , it is needed to relocate stack to another section of RAM.

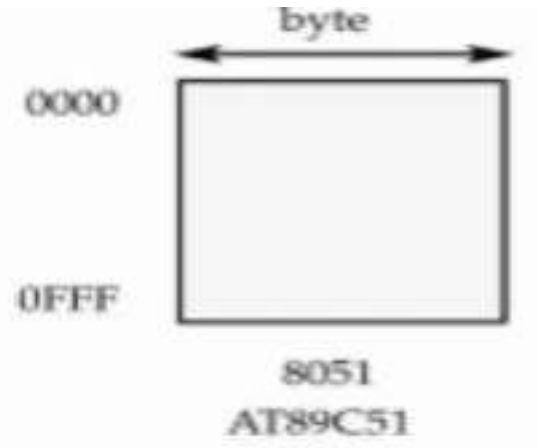
Table : 8051 Special Function Register (SFR) Addresses

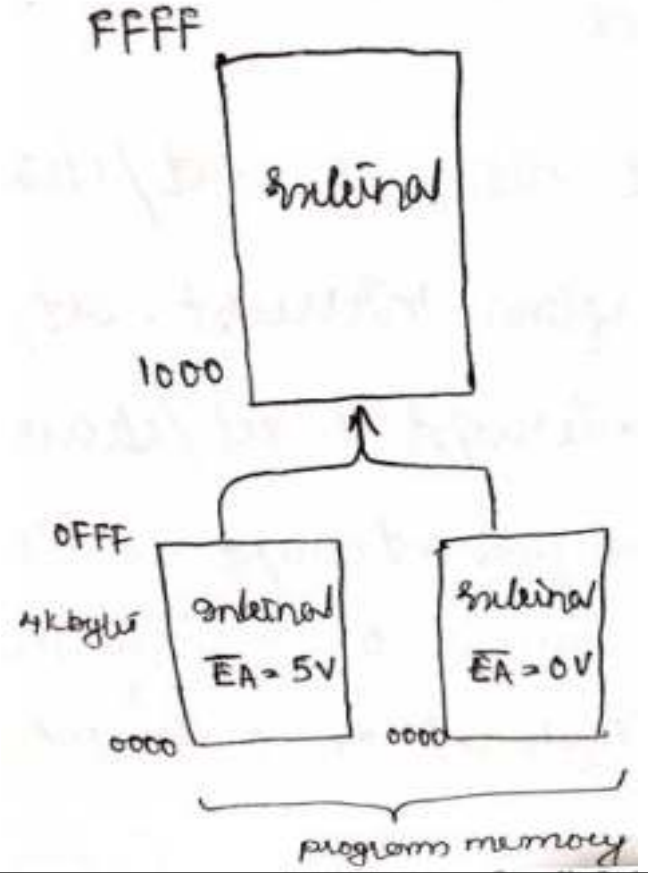
Symbol	Name	Address
ACC*	Accumulator	0E0H
B*	B register	0F0H
PSW*	Program status word	0D0H
SP	Stack pointer	81H
DPTR	Data pointer 2 bytes	
DPL	Low byte	82H
DPH	High byte	83H
P0*	Port 0	80H
P1*	Port 1	90H
P2*	Port 2	0A0H
P3*	Port 3	0B0H
IP*	Interrupt priority control	0B8H
IE*	Interrupt enable control	0A8H
TMOD	Timer/counter mode control	89H
TCON*	Timer/counter control	88H
T2CON*	Timer/counter 2 control	0C8H
T2MOD	Timer/counter mode control	0C9H
TH0	Timer/counter 0 high byte	8CH
TL0	Timer/counter 0 low byte	8AH
TH1	Timer/counter 1 high byte	8DH
TL1	Timer/counter 1 low byte	8BH
TH2	Timer/counter 2 high byte	0CDH
TL2	Timer/counter 2 low byte	0CCH
RCAP2H	T/C 2 capture register high byte	0CBH
RCAP2L	T/C 2 capture register low byte	0CAH
SCON*	Serial control	98H
SBUF	Serial data buffer	99H
PCON	Power control	87H

* Bit-addressable

ROM in 8051:

- 8051 has 4K bytes of internal ROM.
- Internal ROM of 8051 starts at 0000H and goes upto 0FFF H which corresponds to 4K.
- Externally if needed it can interface upto 64K bytes of memory externally. But the total of internal and external ROM should be equal to 64K bytes. ie, if we are using internal ROM and external ROM then, internal from 0000 H to 0FFF H and external from 1000 H to FFFF H.
- If only external ROM is used for total 64K bytes, then EA pin should be kept low for that. if EA pin is high then first internal ROM then remaining can be external ROM.





Program Status Word in 8051 [PSW]:

The program status word (PSW) register is an 8-bit register. It is also referred to as the *flag register*. Although the PSW register is 8 bits wide, only 6 bits of it are used by the 8051. The two unused bits are user-definable flags. Four of the flags are called *conditional flags*, meaning that they indicate some conditions that result after an instruction is executed. These four are CY (carry), AC (auxiliary carry), P (parity), and OV (overflow).

CY	AC	F0	RS1	RS0	OV	--	P

CY	PSW.7	Carry flag.
AC	PSW.6	Auxiliary carry flag.
F0	PSW.5	Available to the user for general purpose.
RS1	PSW.4	Register Bank selector bit 1.
RS0	PSW.3	Register Bank selector bit 0.
OV	PSW.2	Overflow flag.
--	PSW.1	User-definable bit.
P	PSW.0	Parity flag. Set/cleared by hardware each instuction cycle to indicate an odd/even number of 1 bits in the accumulator.

An Assembly language instruction consists of four fields:

```
[label:] mnemonic [operands] [;comment]
```

Brackets indicate that a field is optional, and not all lines have them. Brackets should not be typed in. Regarding the above format, the following points should be noted.

1. The label field allows the program to refer to a line of code by name. The label field cannot exceed a certain number of characters. Check your assembler for the rule.
2. The Assembly language mnemonic (instruction) and operand(s) fields together perform the real work of the program and accomplish the tasks for which the program was written. In Assembly language statements such as

```
ADD A,B
MOV A,#67
```

ADD and MOV are the mnemonics, which produce opcodes; and "A, B" and "A, #67" are the operands.

```
MOV destination,source ;copy source to dest.
```

8051 Data Types and Directives:

- Data Type in 8051 is of 8 bits.

Assembler Directives:

ORG (origin)

The ORG directive is used to indicate the beginning of the address. The number that comes after ORG can be either in hex or in decimal. If the number is not followed by H, it is decimal and the assembler will convert it to hex.

EQU (equate)

This is used to define a constant without occupying a memory location. The EQU directive does not set aside storage for a data item but associates a constant value with a data label so that when the label appears in the program, its constant value will be substituted for the label. The following uses EQU for the counter constant and then the constant is used to load the R3 register.

```
COUNT    EQU 25
...      ....
MOV      R3,#COUNT
```

When executing the instruction "`MOV R3,#COUNT`", the register R3 will be loaded with the value 25 (notice the # sign). What is the advantage of using EQU? Assume that there is a constant (a fixed value) used in many different places in the program, and the programmer wants to change its value throughout. By the use of EQU, the programmer can change it once and the assembler will change all of its occurrences, rather than search the entire program trying to find every occurrence.

END directive

Another important pseudocode is the END directive. This indicates to the assembler the end of the source (asm) file. The END directive is the last line of an 8051 program, meaning that in the source code anything after the END directive is ignored by the assembler.

DB (define byte)

The DB directive is the most widely used data directive in the assembler. It is used to define the 8-bit data. When DB is used to define data, the numbers can be in decimal, binary, hex, or ASCII formats. For decimal, the "D" after the decimal number is optional, but using "B" (binary) and "H" (hexadecimal) for the others is required. Regardless of which is used, the assembler will convert the numbers into hex. To indicate ASCII, simply place the characters in quotation marks ('like this'). The assembler will assign the ASCII code for the numbers or characters automatically.

```
            ORG     500H
DATA1:      DB      28                  ;DECIMAL(1C in hex)
DATA2:      DB      00110101B           ;BINARY (35 in hex)
DATA3:      DB      39H                 ;HEX
            ORG     510H
DATA4:      DB      "2591"              ;ASCII NUMBERS
            ORG     518H
DATA6:      DB      "My name is Joe"    ;ASCII CHARACTERS
```

Either single or double quotes can be used around ASCII strings. This can be useful for strings, which contain a single quote such as "O'Leary". DB is also used to allocate memory in byte-sized chunks.

Addressing Modes in 8051:

The various ways of accessing data by the CPU is known as Addressing modes.

The five different addressing modes in 8051 are,

1. **Immediate**
2. **Direct**
3. **Register**
4. **Register indirect**
5. **Indexed**

Thank You

MPES
Module_5_3

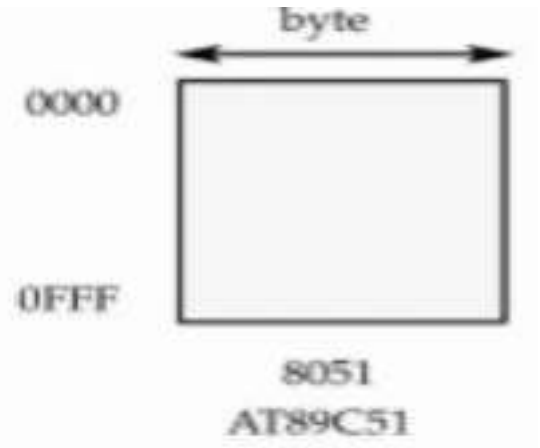
Table : 8051 Special Function Register (SFR) Addresses

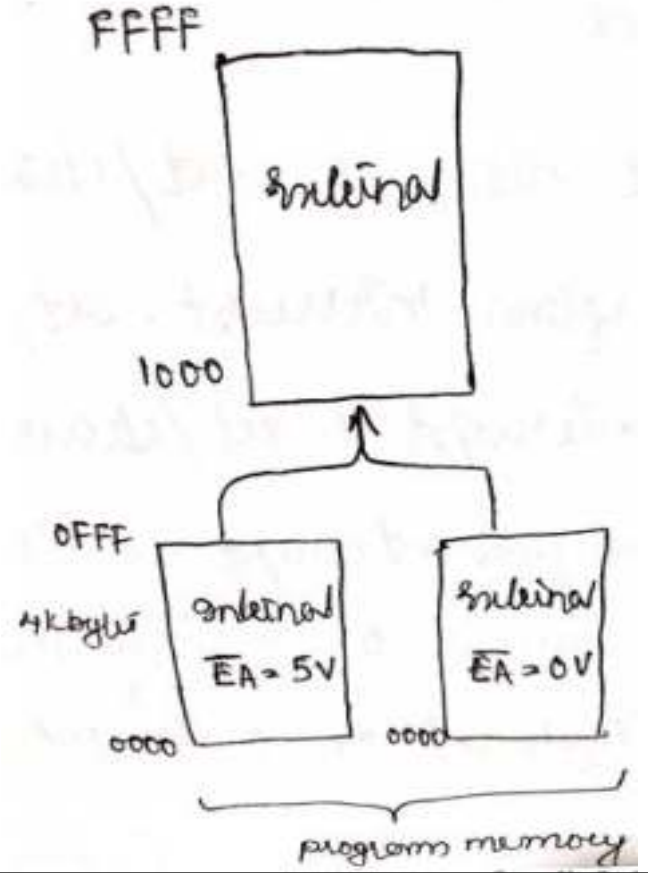
Symbol	Name	Address
ACC*	Accumulator	0E0H
B*	B register	0F0H
PSW*	Program status word	0D0H
SP	Stack pointer	81H
DPTR	Data pointer 2 bytes	
DPL	Low byte	82H
DPH	High byte	83H
P0*	Port 0	80H
P1*	Port 1	90H
P2*	Port 2	0A0H
P3*	Port 3	0B0H
IP*	Interrupt priority control	0B8H
IE*	Interrupt enable control	0A8H
TMOD	Timer/counter mode control	89H
TCON*	Timer/counter control	88H
T2CON*	Timer/counter 2 control	0C8H
T2MOD	Timer/counter mode control	0C9H
TH0	Timer/counter 0 high byte	8CH
TL0	Timer/counter 0 low byte	8AH
TH1	Timer/counter 1 high byte	8DH
TL1	Timer/counter 1 low byte	8BH
TH2	Timer/counter 2 high byte	0CDH
TL2	Timer/counter 2 low byte	0CCH
RCAP2H	T/C 2 capture register high byte	0CBH
RCAP2L	T/C 2 capture register low byte	0CAH
SCON*	Serial control	98H
SBUF	Serial data buffer	99H
PCON	Power control	87H

* Bit-addressable

ROM in 8051:

- 8051 has 4K bytes of internal ROM.
- Internal ROM of 8051 starts at 0000H and goes upto 0FFF H which corresponds to 4K.
- Externally if needed it can interface upto 64K bytes of memory externally. But the total of internal and external ROM should be equal to 64K bytes. ie, if we are using internal ROM and external ROM then, internal from 0000 H to 0FFF H and external from 1000 H to FFFF H.
- If only external ROM is used for total 64K bytes, then EA pin should be kept low for that. if EA pin is high then first internal ROM then remaining can be external ROM.





Program Status Word in 8051 [PSW]:

The program status word (PSW) register is an 8-bit register. It is also referred to as the *flag register*. Although the PSW register is 8 bits wide, only 6 bits of it are used by the 8051. The two unused bits are user-definable flags. Four of the flags are called *conditional flags*, meaning that they indicate some conditions that result after an instruction is executed. These four are CY (carry), AC (auxiliary carry), P (parity), and OV (overflow).

CY	AC	F0	RS1	RS0	OV	--	P

CY	PSW.7	Carry flag.
AC	PSW.6	Auxiliary carry flag.
F0	PSW.5	Available to the user for general purpose.
RS1	PSW.4	Register Bank selector bit 1.
RS0	PSW.3	Register Bank selector bit 0.
OV	PSW.2	Overflow flag.
--	PSW.1	User-definable bit.
P	PSW.0	Parity flag. Set/cleared by hardware each instuction cycle to indicate an odd/even number of 1 bits in the accumulator.

An Assembly language instruction consists of four fields:

```
[label:] mnemonic [operands] [;comment]
```

Brackets indicate that a field is optional, and not all lines have them. Brackets should not be typed in. Regarding the above format, the following points should be noted.

1. The label field allows the program to refer to a line of code by name. The label field cannot exceed a certain number of characters. Check your assembler for the rule.
2. The Assembly language mnemonic (instruction) and operand(s) fields together perform the real work of the program and accomplish the tasks for which the program was written. In Assembly language statements such as

```
ADD A,B
MOV A,#67
```

ADD and MOV are the mnemonics, which produce opcodes; and "A, B" and "A, #67" are the operands.

```
MOV destination,source ;copy source to dest.
```

8051 Data Types and Directives:

- Data Type in 8051 is of 8 bits.

Assembler Directives:

ORG (origin)

The ORG directive is used to indicate the beginning of the address. The number that comes after ORG can be either in hex or in decimal. If the number is not followed by H, it is decimal and the assembler will convert it to hex.

EQU (equate)

This is used to define a constant without occupying a memory location. The EQU directive does not set aside storage for a data item but associates a constant value with a data label so that when the label appears in the program, its constant value will be substituted for the label. The following uses EQU for the counter constant and then the constant is used to load the R3 register.

```
COUNT    EQU 25
...      ....
MOV      R3,#COUNT
```

When executing the instruction "`MOV R3,#COUNT`", the register R3 will be loaded with the value 25 (notice the # sign). What is the advantage of using EQU? Assume that there is a constant (a fixed value) used in many different places in the program, and the programmer wants to change its value throughout. By the use of EQU, the programmer can change it once and the assembler will change all of its occurrences, rather than search the entire program trying to find every occurrence.

END directive

Another important pseudocode is the END directive. This indicates to the assembler the end of the source (asm) file. The END directive is the last line of an 8051 program, meaning that in the source code anything after the END directive is ignored by the assembler.

DB (define byte)

The DB directive is the most widely used data directive in the assembler. It is used to define the 8-bit data. When DB is used to define data, the numbers can be in decimal, binary, hex, or ASCII formats. For decimal, the "D" after the decimal number is optional, but using "B" (binary) and "H" (hexadecimal) for the others is required. Regardless of which is used, the assembler will convert the numbers into hex. To indicate ASCII, simply place the characters in quotation marks ('like this'). The assembler will assign the ASCII code for the numbers or characters automatically.

```
            ORG     500H
DATA1:      DB      28                  ;DECIMAL(1C in hex)
DATA2:      DB      00110101B           ;BINARY (35 in hex)
DATA3:      DB      39H                 ;HEX
            ORG     510H
DATA4:      DB      "2591"              ;ASCII NUMBERS
            ORG     518H
DATA6:      DB      "My name is Joe"    ;ASCII CHARACTERS
```

Either single or double quotes can be used around ASCII strings. This can be useful for strings, which contain a single quote such as "O'Leary". DB is also used to allocate memory in byte-sized chunks.

Addressing Modes in 8051:

The various ways of accessing data by the CPU is known as Addressing modes.

The five different addressing modes in 8051 are,

1. **Immediate**
2. **Direct**
3. **Register**
4. **Register indirect**
5. **Indexed**

Thank You

MPES
Module 5_4

Addressing modes in 8051:

The various ways of accessing data by the CPU is known as Addressing modes.

The five different addressing modes in 8051 are,

1. **Immediate**
2. **Direct**
3. **Register**
4. **Register indirect**
5. **Indexed**

Immediate addressing mode

In this addressing mode, the source operand is a constant. In immediate addressing mode, as the name implies, when the instruction is assembled, the operand comes immediately after the opcode. Notice that the immediate data must be preceded by the pound sign, "#". This addressing mode can be used to load information into any of the registers, including the DPTR register. Examples follow.

```
MOV A,#25H          ;load 25H into A
MOV R4,#62          ;load the decimal value 62 into R4
MOV B,#40H          ;load 40H into B
MOV DPTR,#4521H     ;DPTR=4512H
```

Although the DPTR register is 16-bit, it can also be accessed as two 8-bit registers, DPH and DPL, where DPH is the high byte and DPL is the low byte. Look at the following code.

```
MOV DPTR,#2550H
```

is the same as:

```
MOV DPL,#50H
MOV DPH,#25H
```

We can use the EQU directive to access immediate data as shown below.

```
          COUNT     EQU 30
          ...       ..
          MOV       R4,#COUNT       ;R4=1E(30=1EH)
          MOV       DPTR,#MYDATA    ;DPTR=200H

          ORG 200H
MYDATA:   DB  "America"
```

Notice that we can also use immediate addressing mode to send data to 8051 ports. For example, "`MOV P1, #55H`" is a valid instruction.

Register addressing mode

Register addressing mode involves the use of registers to hold the data to be manipulated. Examples of register addressing mode follow.

```
MOV A,R0     ;copy the contents of R0 into A
MOV R2,A     ;copy the contents of A into R2
ADD A,R5     ;add the contents of R5 to contents of A
ADD A,R7     ;add the contents of R7 to contents of A
MOV R6,A     ;save accumulator in R6
```

It should be noted that the source and destination registers must match in size. In other words, coding "`MOV DPTR,A`" will give an error, since the source is an 8-bit register and the destination is a 16-bit register. See the following.

```
MOV DPTR,#25F5H
MOV R7,DPL
MOV R6,DPH
```

Notice that we can move data between the accumulator and Rn (for n = 0 to 7) but movement of data between Rn registers is not allowed. For example, the instruction "`MOV R4,R7`" is invalid.

Direct addressing mode

In the direct addressing mode, the data is in a RAM memory location whose address is known, and this address is given as a part of the instruction. Contrast this with immediate addressing mode, in which the operand itself is provided with the instruction. The "#" sign distinguishes between the two modes. See the examples below, and note the absence of the "#" sign.

```
MOV R0,40H      ;save content of RAM location 40H in R0
MOV 56H,A       ;save content of A in RAM location 56H
MOV R4,7FH      ;move contents of RAM location 7FH to R4
```

Regarding direct addressing mode, notice the following two points: (a) the address value is limited to one byte, 00 - FFH, which means this addressing mode is limited to accessing RAM locations and registers located inside the 8051.

Stack and direct addressing mode

Another major use of direct addressing mode is the stack. In the 8051 family, only direct addressing mode is allowed for pushing onto the stack. Therefore, an instruction such as "`PUSH A`" is invalid. Pushing the accumulator onto the stack must be coded as "`PUSH 0E0H`" where 0E0H is the address of register A. Similarly, pushing R3 of bank 0 is coded as "`PUSH 03`". Direct addressing mode must be used for the POP instruction as well. For example, "`POP 04`" will pop the top of the stack into R4 of bank 0.

Register indirect addressing mode

In the register indirect addressing mode, a register is used as a pointer to the data. If the data is inside the CPU, only registers R0 and R1 are used for this purpose.

```
MOV A,@R0       ;move contents of RAM location whose
                ;address is held by R0 into A
MOV @R1,B       ;move contents of B into RAM location
                ;whose address is held by R1
```

Notice that R0 (as well as R1) is preceded by the "@" sign. In the absence of the "@" sign, MOV will be interpreted as an instruction moving the contents of register R0 to A, instead of the contents of the memory location pointed to by R0.

Limitation of register indirect addressing mode in the 8051

As stated earlier, R0 and R1 are the only registers that can be used for pointers in register indirect addressing mode. Since R0 and R1 are 8 bits wide, their use is limited to accessing any information in the internal RAM (scratch pad memory of 30H - 7FH, or SFR). However, there are times when we need to access data stored in external RAM or in the code space of on-chip ROM. Whether accessing externally connected RAM or on-chip ROM, we need a 16-bit pointer. In such cases, the DPTR register is used

Indexed addressing mode and on-chip ROM access

Indexed addressing mode is widely used in accessing data elements of look-up table entries located in the program ROM space of the 8051. The instruction used for this purpose is "`MOVC A,@A+DPTR`". The 16-bit register DPTR and register A are used to form the address of the data element stored in on-chip ROM. Because the data elements are stored in the program (code) space ROM of the 8051, the instruction MOVC is used instead of MOV. The "C" means code. In this instruction the contents of A are added to the 16-bit register DPTR to form the 16-bit address of the needed data.

Example

The word "SAM" is to be burned in the flash ROM location starting from 0400H of an AT89C51. Write a program to do this and to read this data into internal RAM locations starting from 60H.

Solution:

```
ORG 0000H              ;program starts at location 0000H
CLR A                  ;A=0
MOV DPTR,#0400H        ;DPTR=400H (points to first source location)
MOVC A,@A+DPTR         ;get 'S' from location 400H
MOV 60H,A              ;move it to RAM location 60H
```

Example

Assuming that ROM space starting at 250H contains "America", write a program to transfer the bytes into RAM locations starting at 40H.

Solution:

```
;(a) This method uses a counter
        ORG   0000
        MOV   DPTR,#MYDATA      ;load ROM pointer
        MOV   R0,#40H           ;load RAM pointer
        MOV   R2,#7             ;load counter
BACK:   CLR   A                 ;A = 0
        MOVC  A,@A+DPTR         ;move data from code space
        MOV   @R0,A             ;save it in RAM
        INC   DPTR              ;increment ROM pointer
        INC   R0                ;increment RAM pointer
        DJNZ  R2,BACK           ;loop until counter=0
HERE:   SJMP  HERE


; ----------On-chip code space used for storing data
        ORG   250H
MYDATA  DB    "AMERICA"
        END
```

Example

Write a program to get the x value from P1 and send x^2 to P2, continuously.

Solution:

```
            ORG   0
            MOV   DPTR,#300H        ;load look-up table address
            MOV   A,#0FFH           ;A=FF
            MOV   P1,A              ;configure P1 as input port
BACK:       MOV   A,P1              ;get X
            MOVC  A,@A+DPTR         ;get X squared from table
            MOV   P2,A              ;issue it to P2
            SJMP  BACK              ;keep doing it
            ORG   300H
XSQR_TABLE:
            DB    0,1,4,9,16,25,36,49,64,81
            END
```

Notice that the first instruction could be replaced with "`MOV DPTR, #XSQR_TABLE`"

Example

External data ROM has a look-up table for the squares of numbers 0 - 9. Since the internal RAM of the 8031/51 has a shorter access time, write a program to copy the table elements into internal RAM starting at address 30H. The look-up table address starts at address 0 of external ROM.

Solution:

```
TABLE     EQU   000H
RAMTBLE   EQU   30H
COUNT     EQU   10
          ...
          MOV   DPTR,#TABLE     ;pointer to external data
          MOV   R5,#COUNT       ;counter
          MOV   R0,#RAMTBLE     ;pointer to internal RAM
BACK:     MOVX  A,@DPTR         ;get byte from external mem
          MOV   @R0,A           ;store it in internal RAM
          INC   DPTR            ;next data location
          INC   R0              ;next RAM location
          DJNZ  R5,BACK         ;until all are read
```

Thank You

MPES
Module 5_5

I/O ports in 8051:

- 8051 has four 8 bit bidirectional ports, ie, 32 I/O pins.
- Port 0, Port 1, Port 2, Port 3
- All ports upon reset is configured as output port.

Port 0:

- Designated as AD0 to AD7
- Port address (SFR) is 080 H.
- Port 0 can be used for Address/ Data when connected to an external memory.
- For memory addressing, the lower byte of 16 bit address is in P0.
- When used as an output port, Port 0 needs as external Pull up resistor to sourcea high value to the output circuit.
- When used as an output Port, whatever value comes in the port 0 SFR latch will be given as the output.
- When external memory is accesed Port 0 (input) does not need a Pull up resistor.

Port 0 Internal Structure:

- To work Port 0 as output port, Write the latch (SFR) of port 0 with zero, so that what wver comes at the latch can be given as output since the lower N channel FET is in on state. (refer second fig.) By default, ie, without external puul up resistor ,it can't source high output value.
- To make Port 0 as an input port , write a '1' to the port 0 SFR, which will turn off the lower N channel FET . This will make pin in the high impedance or floating state.(refer first fig.) So that whatever value sourced by the external circuit can be read by activating 'Read Pin'.
- Ports 1 ,2 and 3 have internal pull ups.
- Bit addressing of Port 0 is possible.
- Port 0 is known is True bidirectional port as it floats or at high impefance state when configured as input port.

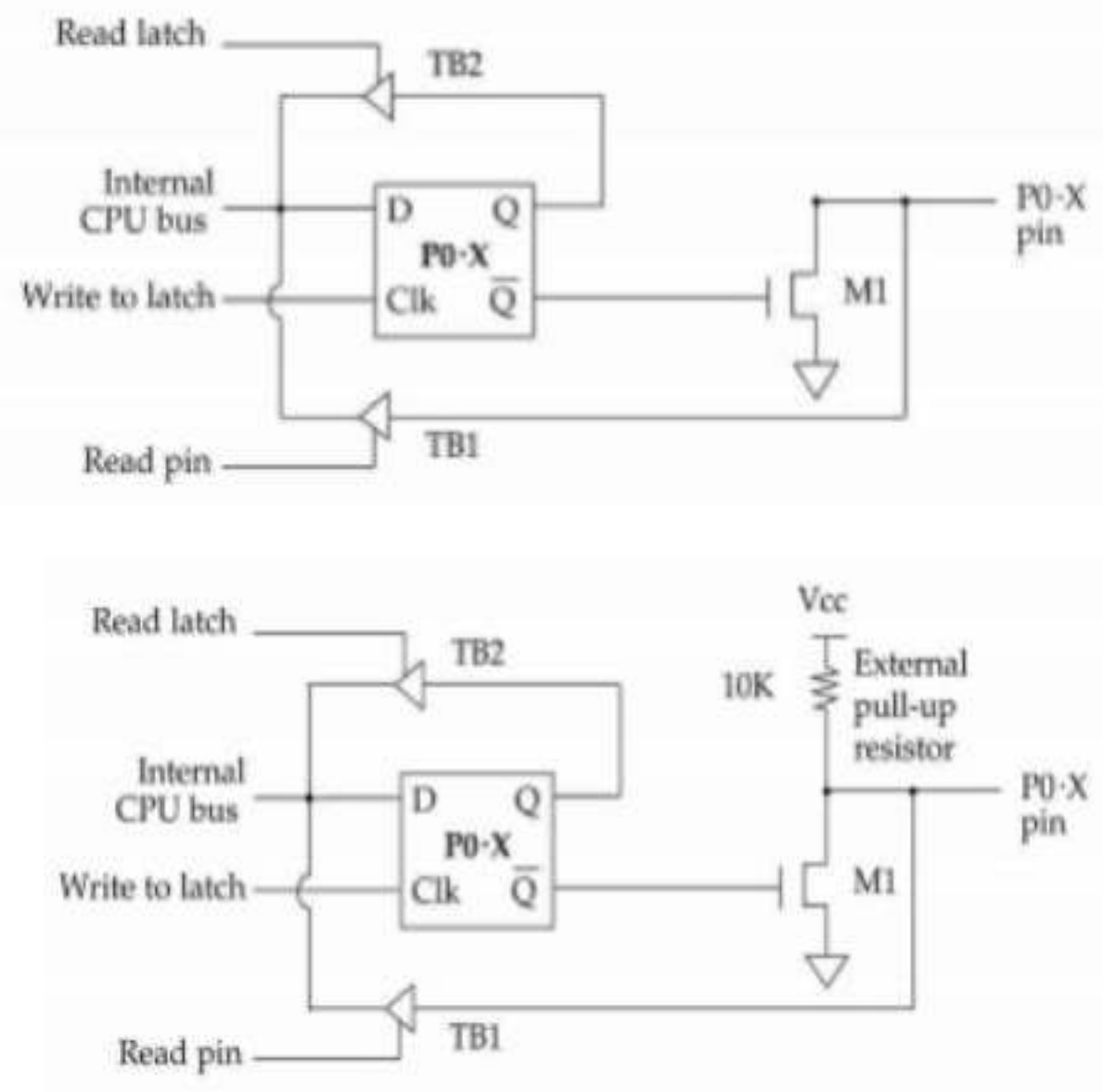
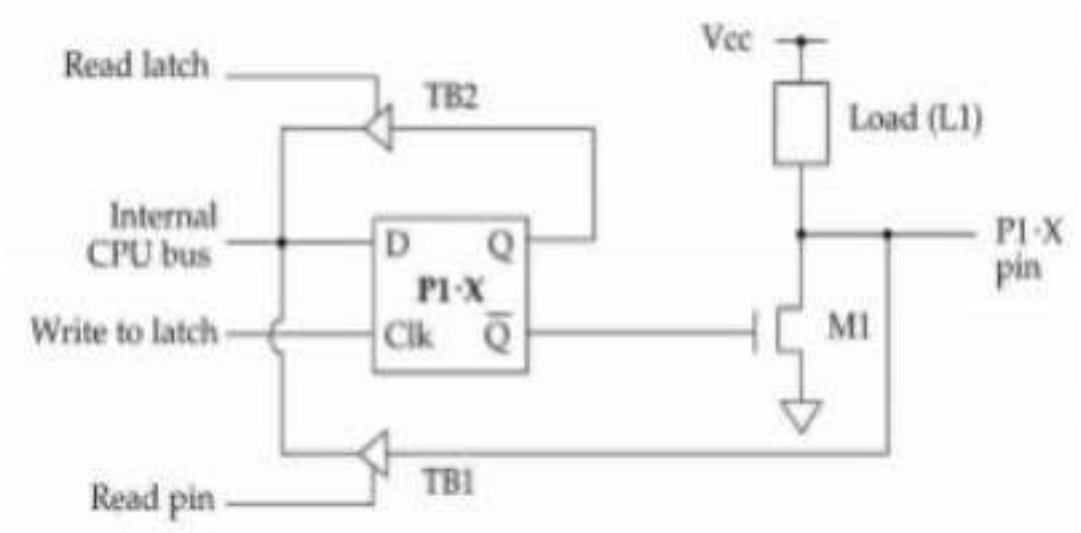
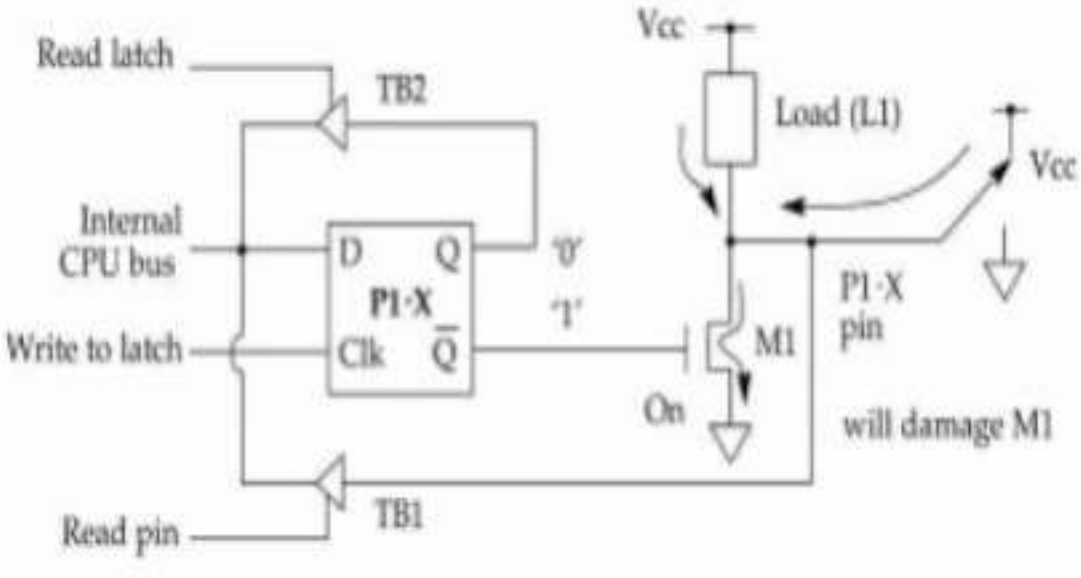


Fig. Internal structure of each Port 0 pin with and without external Pull up resistor.

Port 1 Internal Structure:

- 8 bit Bidirectional Port which is used only for I/O operations.
- Does not need external pull up resistor as it have internal pull ups.
- Bit addressable.
- Writing '1' to SFR latch turns off the lower N cahnnel FET and makes the pin suitable for input operations.
- To make it as an output port , whatever comes in latch can be given as output.
- Now what happens when we write a '0' to the port pin by mistake which is actually configured and wired as an input?
- To solve the above problem connecta current limiting resistor to the external Vcc.
- When configured as input port, Port1, 2 and 3 will pull the status of the pin to high due to its internal pullup resistors so that it will source currents when externally pulled low. Due to this property Port 1, 2 and 3 are known as **Quasi Bidirectional.**
- CPL P1.2 is a instruction which read the Port latch, compliments it and then written back to latch.





Port 2:

- 8 bit Bidirectional I/O port
- Time multiplexed between Higher order address and data.
- Similar port structure to Port 1.
- Bit addressable
- No need of external pull ups.

Port 3:

- 8 bit bidirectional I/O port.
- Bit addressable
- Alternate functions in addition to normal I/O operations.
- No need for external pull ups.

Port 3 Alternate Functions

P3 Bit	Function	Pin
P3.0	RxD	10
P3.1	TxD	11
P3.2	$\overline{\text{INT0}}$	12
P3.3	$\overline{\text{INT1}}$	13
P3.4	T0	14
P3.5	T1	15
P3.6	$\overline{\text{WR}}$	16
P3.7	$\overline{\text{RD}}$	17

Thank you

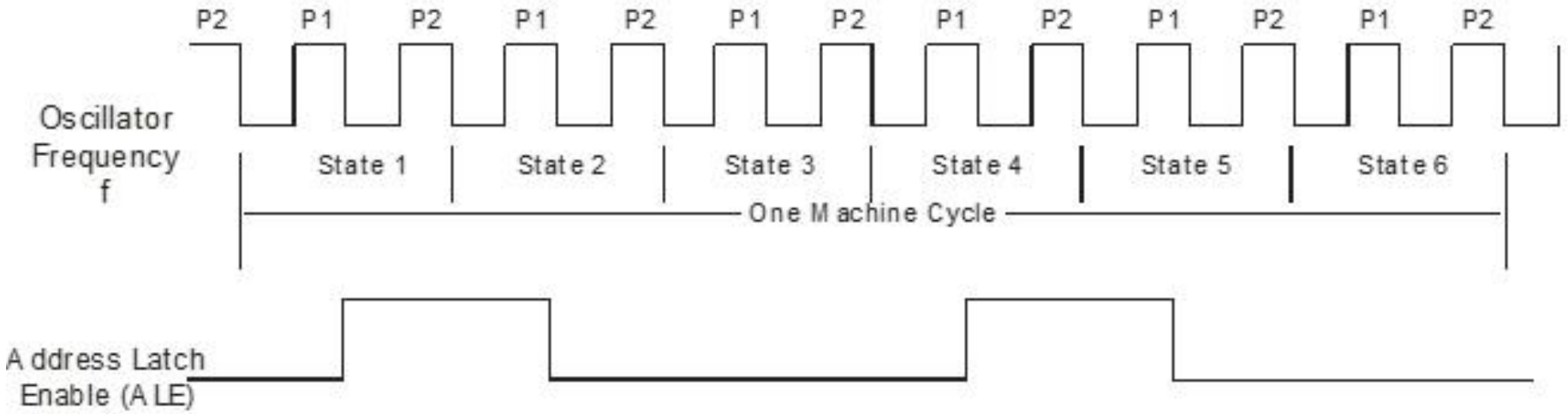
MPES
Module 5_6

Machine Cycle in 8051:

- The Smallest interval of time to ccomplish any simple instruction, or part of a complex instruction is called Machine cycle.
- A Machine Cycle is made up of 6 states. One machine cycle is 12 clock cycles.
- A **State** is the basic time interval for discrete operations of the Micro controller such as Fetching, Decoding and Executing an opcode or writing a data byte.
- Two oscillator pulses , two cycles, define a state.
- Program instructions may require one, two or four machine cycles to get executed depending on the type of the instruction.

$$Time\ for\ one\ Machine\ cycle\ = \frac{1}{clock\ frequency} X\ 12$$

Contd...



As in the figure there are two ALE pulses per Machine cycle. The ALE pulse is used as atiming pulse for external memory access, indicates when every instruction byte is fetched. thus two bytes of a single instruction can be fetched in one machine cycle. But single byte instructions are not executed in half cycle. ie, the second ALE pulse is discarded for single byte instruction.

Timers and Counters in 8051:

- Two 16 bit up counters named Timer 0 and Timer 1 are in 8051.
- Each can be programmed to count internal clock pulses for generating time delay as a Timer and external events as a Counter.
- Each Timer is divided in to two 8 bit registers called TL0 and TH0 for Timer 0 and TL1 and TH1 for Timer 1.
- All Timer / Counter actions are controlled by the bit states in the Timer Mode Control Register TMOD and Timer / Counter Control Register TCON along with certain Program instructions. Four modes of operation are there.
- Timer increases by one in every Machine cycle.ie, in every 12 clock cycles.
- Therefore the frequency of the Timer is always $\frac{1}{12^{th}}$ of oscillator frequency.
- As an example, for a crystal frequency of 12 Mhz, Timer frequency is $\frac{1}{12} X 12Mhz = 1Mhz$

$$Therefore\ time\ for\ one\ count\ is\ \frac{1}{1Mhz} = 1\ Micro\ sec$$

TMOD Register

- TMOD is not Bit addressable.

(MSB) (LSB)

GATE	C/T	M1	M0	GATE	C/T	M1	M0
Timer 1				Timer 0			

GATE Gating control when set. The timer/counter is enabled only while the INTx pin is high and the TRx control pin is set. When cleared, the timer is enabled whenever the TRx control bit is set.

C/T Timer or counter selected cleared for timer operation (input from internal system clock). Set for counter operation (input from Tx input pin).

M1 Mode bit 1

M0 Mode bit 0

M1	M0	Mode	Operating Mode
0	0	0	13-bit timer mode 8-bit timer/counter THx with TLx as 5-bit prescaler
0	1	1	16-bit timer mode 16-bit timer/counters THx and TLx are cascaded; there is no prescaler
1	0	2	8-bit auto reload 8-bit auto reload timer/counter; THx holds a value that is to be reloaded into TLx each time it overflows.
1	1	3	Split timer mode

TMOD Register

Timer 0 Registers

Timer 1 Registers

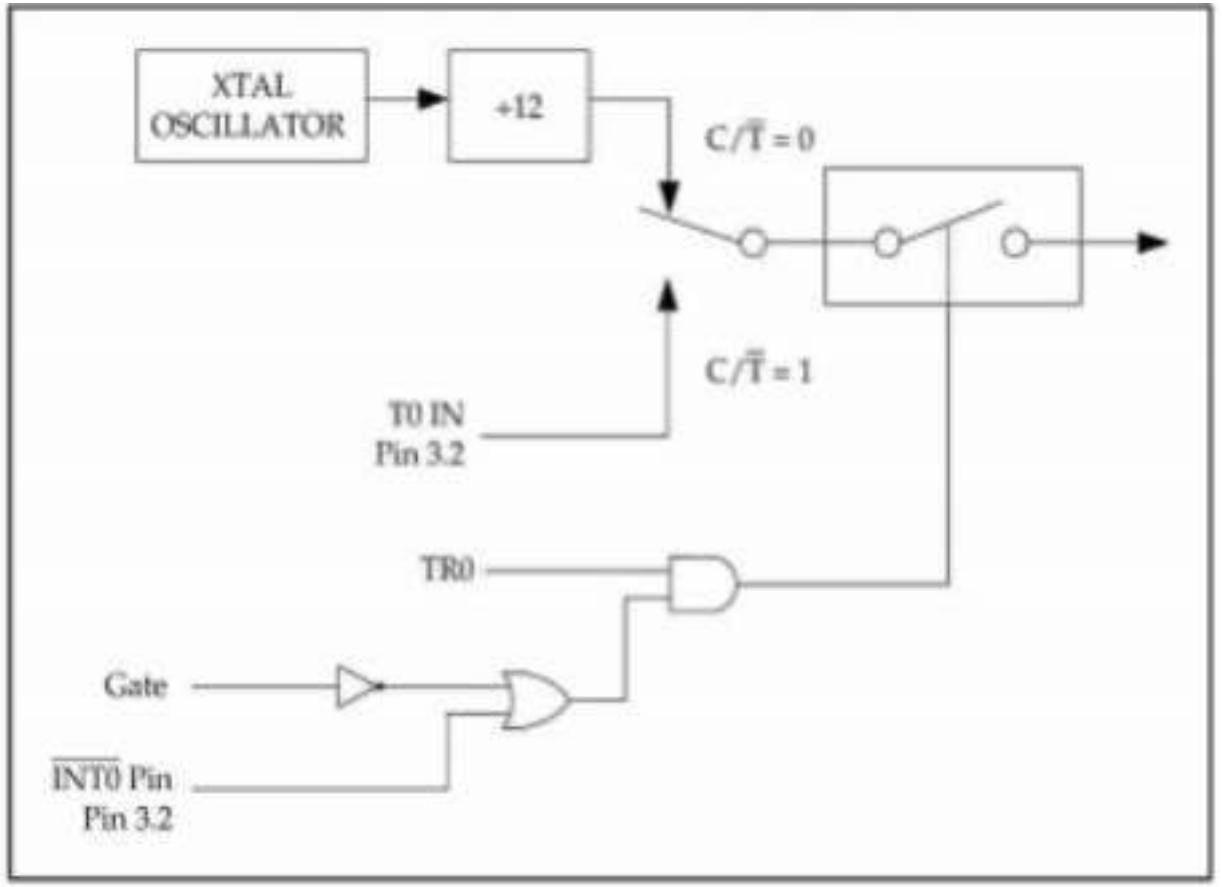
TCON Register:

D7							D0
TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0

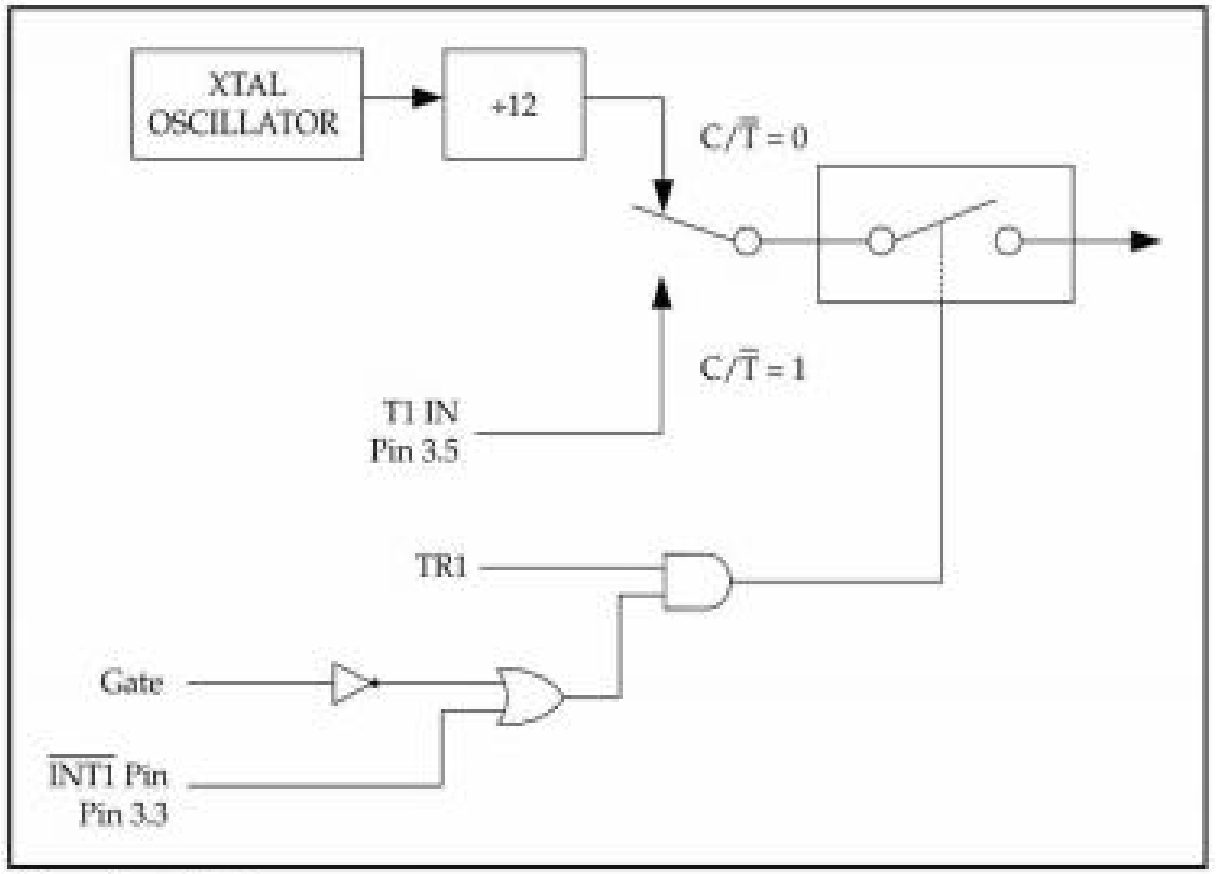
TF1	TCON.7	Timer 1 overflow flag. Set by hardware when timer/counter 1 overflows. Cleared by hardware as the processor vectors to the interrupt service routine.
TR1	TCON.6	Timer 1 run control bit. Set/cleared by software to turn timer/counter 1 on/off.
TF0	TCON.5	Timer 0 overflow flag. Set by hardware when timer/counter 0 overflows. Cleared by hardware as the processor vectors to the service routine.
TR0	TCON.4	Timer 0 run control bit. Set/cleared by software to turn timer/counter 0 on/off.
IE1	TCON.3	External interrupt 1 edge flag. Set by CPU when the external interrupt edge (H-to-L transition) is detected. Cleared by CPU when the interrupt is processed. *Note:* This flag does not latch low-level triggered interrupts.
IT1	TCON.2	Interrupt 1 type control bit. Set/cleared by software to specify falling edge/low-level triggered external interrupt.
IE0	TCON.1	External interrupt 0 edge flag. Set by CPU when external interrupt (H-to-L transition) edge is detected. Cleared by CPU when interrupt is processed. *Note:*This flag does not latch low-level triggered interrupts.
IT0	TCON.0	Interrupt 0 type control bit. Set/cleared by software to specify falling edge/low-level triggered external interrupt.

TCON (Timer/Counter) Register (Bit-addressable)

Timer / Counter Control Logic using GATE bit:



Timer/Counter 0



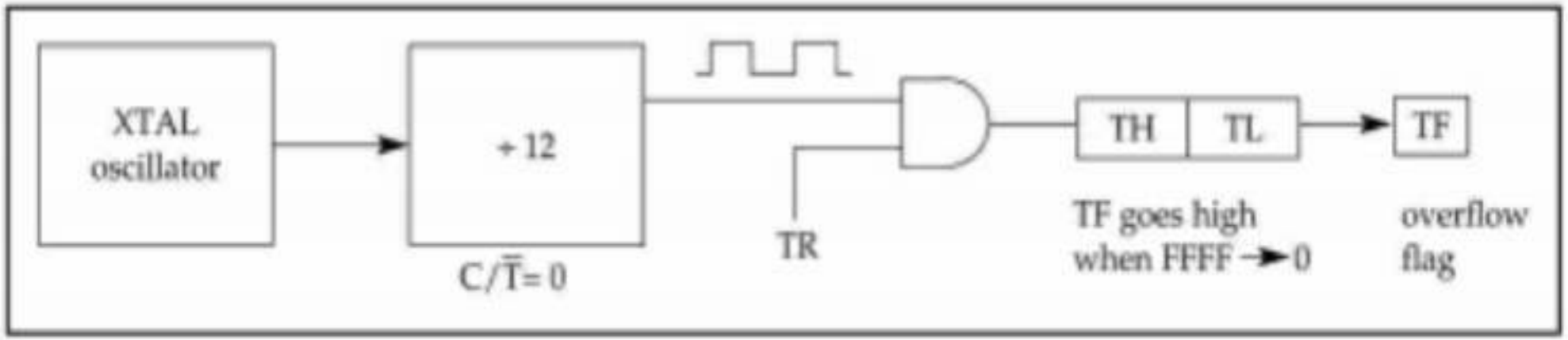
Timer/Counter 1

Thank You

MPES
Module 5_7

Timer Mode 1 operation:

1. It is a 16-bit timer; therefore, it allows values of 0000 to FFFFH to be loaded into the timer's registers TL and TH.
2. After TH and TL are loaded with a 16-bit initial value, the timer must be started. This is done by "`SETB TR0`" for Timer 0 and "`SETB TR1`" for Timer 1.
3. After the timer is started, it starts to count up. It counts up until it reaches its limit of FFFFH. When it rolls over from FFFFH to 0000, it sets high a flag bit called TF (timer flag). This timer flag can be monitored. When this timer flag is raised, one option would be to stop the timer with the instructions "`CLR TR0`" or "`CLR TR1`", for Timer 0 and Timer 1, respectively. Again, it must be noted that each timer has its own timer flag: TF0 for Timer 0, and TF1 for Timer 1.
4. After the timer reaches its limit and rolls over, in order to repeat the process the registers TH and TL must be reloaded with the original value, and TF must be reset to 0.



Steps to program Timer in Mode 1:

1. Load the TMOD value register indicating which timer (Timer 0 or Timer 1) is to be used and which timer mode (0 or 1) is selected.
2. Load registers TL and TH with initial count values.
3. Start the timer.
4. Keep monitoring the timer flag (TF) with the "JNB TFx, target" instruction to see if it is raised. Get out of the loop when TF becomes high.
5. Stop the timer.
6. Clear the TF flag for the next round.
7. Go back to Step 2 to load TH and TL again.

In the following program, we are creating a square wave of 50% duty cycle (with equal portions high and low) on the P1.5 bit. Timer 0 is used to generate the time delay. Analyze the program.

```
          MOV   TMOD,#01          ;Timer 0, mode 1(16-bit mode)
HERE:     MOV   TL0,#0F2H         ;TL0 = F2H, the Low byte
          MOV   TH0,#0FFH         ;TH0 = FFH, the High byte
          CPL   P1.5              ;toggle P1.5
          ACALL DELAY
          SJMP  HERE              ;load TH, TL again
DELAY:
          SETB TR0                ;start Timer 0
AGAIN:    JNB TF0,AGAIN           ;monitor Timer 0 flag until
                                  ;it rolls over
          CLR   TR0               ;stop Timer 0
          CLR   TF0               ;clear Timer 0 flag
          RET
```

In the above program notice the following steps.

1. TMOD is loaded.
2. FFF2H is loaded into TH0 - TL0.
3. P1.5 is toggled for the high and low portions of the pulse.
4. The DELAY subroutine using the timer is called.
5. In the DELAY subroutine, Timer 0 is started by the "`SETB TR0`" instruction.
6. Timer 0 counts up with the passing of each clock, which is provided by the crystal oscillator. As the timer counts up, it goes through the states of FFF3, FFF4, FFF5, FFF6, FFF7, FFF8, FFF9, FFFA, FFFB, and so on until it reaches FFFFH. One more clock rolls it to 0, raising the timer flag (TF0 = 1). At that point, the JNB instruction falls through.
7. Timer 0 is stopped by the instruction "`CLR TR0`". The DELAY subroutine ends, and the process is repeated.

Notice that to repeat the process, we must reload the TL and TH registers and start the timer again.

In Example calculate the amount of time delay in the DELAY subroutine generated by the timer. Assume that XTAL = 11.0592 MHz.

Solution:

The timer works with a clock frequency of 1/12 of the XTAL frequency; therefore, we have 11.0592 MHz / 12 = 921.6 kHz as the timer frequency. As a result, each clock has a period of T = 1 / 921.6 kHz = 1.085 µs. In other words, Timer 0 counts up each 1.085 µs resulting in delay = number of counts × 1.085 µs.

The number of counts for the rollover is FFFFH – FFF2H = 0DH (13 decimal). However, we add one to 13 because of the extra clock needed when it rolls over from FFFF to 0 and raises the TF flag. This gives 14 × 1.085 µs = 15.19 µs for half the pulse. For the entire period T = 2 × 15.19 µs = 30.38 µs gives us the time delay generated by the timer.

(a) in hex	(b) in decimal
(FFFF - YYXX + 1) × 1.085 µs where YYXX are TH, TL initial values respectively. Notice that values YYXX are in hex.	Convert YYXX values of the TH,TL register to decimal to get a NNNNN decimal number, then(65536 - NNNNN) × 1.085 µs

Timer Delay Calculation for XTAL = 11.0592 MHz

To find the Values to be loaded in to Timer in mode 1:

Assuming that we know the amount of timer delay we need, the question is how to find the values needed for the TH, TL registers. To calculate the values to be loaded into the TL and TH registers look at Example where we use crystal frequency of 11.0592 MHz for the 8051 system.

Assuming XTAL = 11.0592 MHz from Example 9-10 we can use the following steps for finding the TH, TL registers' values.

1. Divide the desired time delay by 1.085 µs.
2. Perform 65536 – *n*, where *n* is the decimal value we got in Step 1.
3. Convert the result of Step 2 to hex, where *yyxx* is the initial hex value to be loaded into the timer's registers.
4. Set TL = *xx* and TH = *yy*.

Example

Assume that XTAL = 11.0592 MHz. What value do we need to load into the timer's registers if we want to have a time delay of 5 ms (milliseconds)? Show the program for Timer 0 to create a pulse width of 5 ms on P2.3.

Solution:

Since XTAL = 11.0592 MHz, the counter counts up every 1.085 µs. This means that out of many 1.085 µs intervals we must make a 5 ms pulse. To get that, we divide one by the other. We need 5 ms / 1.085 µs = 4608 clocks. To achieve that we need to load into TL and TH the value 65536 – 4608 = 60928 = EE00H. Therefore, we have TH = EE and TL = 00.

```
        CLR   P2.3            ;clear P2.3
        MOV   TMOD,#01        ;Timer 0, mode 1 (16-bit mode)
BACK:   MOV   TL0,#0          ;TL0 = 0, Low byte
        MOV   TH0,#0EEH       ;TH0 = EE(hex), High byte
        SETB  P2.3            ;SET P2.3 high
        SETB  TR0             ;start Timer 0
AGAIN:  JNB   TF0,AGAIN       ;monitor Timer 0 flag
                              ;until it rolls over
        CLR   P2.3            ;clear P2.3
        CLR   TR0             ;stop Timer 0
        CLR   TF0             ;clear Timer 0 flag
        SJMP  BACK            ;reload timer
```

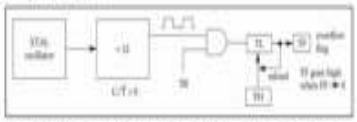
Thank You

MPES
Module 5_8

Timer Mode 2 Programming in 8051:

8 bit Timer, with auto reload feature. Used for Baud rate setting in serial communication.

1. It is an 8-bit timer, therefore, it allows only values of 00 to FFH to be loaded into the timer's register TH.
2. After TH is loaded with the 8-bit value, the 8051 gives a copy of it to TL. Then the timer must be started. This is done by the instruction "SETB TR0" for Timer 0 and "SETB TR1" for Timer 1. This is just like mode 1.
3. After the timer is started, it starts to count up by incrementing the TL register. It counts up until it reaches its limit of FFH. When it rolls over from FFH to 00, it sets high the TF (timer flag). If we are using Timer 0, TF0 goes high; if we are using Timer 1, TF1 is raised.



4. When the TL register rolls from FFH to 0 and TF is set to 1, TL is reloaded automatically with the original value kept by the TH register. To repeat the process, we must simply clear TF and let it go without any need by the programmer to reload the original value. This makes mode 2 an auto-reload, in contrast with mode 1 in which the programmer has to reload TH and TL.

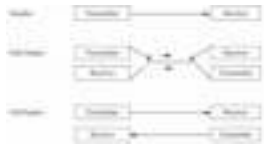
- T0 and T1 pins , ie, P3.4 and P3.5 respectively are used to connect the external events.
- In comparison to Timer operation, here the clock source is from external events. Otherwise similar to timer only.
- 1/24th of the crystal frequency is the maximum count rate that can be achieved without loosing accuracy as it needs two machine cycle to sense the changes by the micro controller.
- The change in external input ie, in pins T0 and T1 should hold for atleast one machine cycle.

Thank You

MPES
Module 5_9

Basics of Data communication:

- 8051 is a parallel device that transfers eight bits of data simultaneously over eight data lines to parallel I/O devices such as Printers, DAC etc.
- Parallel data transfer over long distance is very expensive, hence Serial data communication is widely used for long distance communication.
- In serial data communication, one bit of information , ata time is transferred over a single line.
- The data byte is always transmitted with the least significant bit first.
- Serial communication in 8051 is Full Duplex in nature, ie, data is transmitted in both ways at the same time.



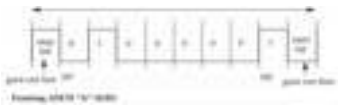
Types of Serial Data communication:

Synchronous Serial Data Communication:

- In which Transmitter and Receiver are synchronised with the common clock signal.

Asynchronous Serial data Communication:

- In which different clock sources are used for transmitter and receiver.
- Data is transmitted using Start and Stop bits.
- Transmision begins with start bit, then data and end with the stop bit.



Serial Port in 8051:

- Full Duplex Serial communication using TxD (P3.1) and RxD (P3.0) pins.
- RS232 I/O interfacing standard is used. Where '1' is represented using -3 V to -25 V and '0' is represented by 3 V to 25 V. MAX 232 drivers are used to interface it with TTL compatable devices.
- 8051 uses SBUF register to hold data during Serial communication.
- SBUF is physically two registers. One is write only used to hold data to be transmitted out and the other is Read only used to hold the received data . Both mutually exclusive registers can be accessed using same address 99 H, differentiated with the help of instruction, for transmit or receive.
- Four modes of Serial communication in 8051, selected with the SMx bits in SCON register. [Serial Port Control]
- PCON is another special function register used to control data rates.

SCON Register:

SM0	SCON.7	Serial port mode specifier
SM1	SCON.6	Serial port mode specifier
SM2	SCON.5	Used for multiprocessor communication
REN	SCON.4	Set/cleared by software to enable/disable reception
TB8	SCON.3	Not widely used.
RB8	SCON.2	Not widely used.
TI	SCON.1	Transmit interrupt flag. Set by hardware at the beginning of the stop bit in mode 1. Must be cleared by software.
RI	SCON.0	Receive interrupt flag. Set by hardware halfway through the stop bit time in mode 1. Must be cleared by software.

Note: Make SM2, TB8, and RB8 = 0.

SCON: Serial Port Control Register (Bit-Addressable)

SM0	SM1	
0	0	Serial Mode 0
0	1	Serial Mode 1, 8-bit data, 1 stop bit, 1 start bit
1	0	Serial Mode 2
1	1	Serial Mode 3

(If the 4 serial modes, only mode 1 is of interest to us.
In the SCON register, when serial mode 1 is chosen, the data framing is 8 bits: 1 stop bit, and 1 start bit.
More importantly, serial mode 1 allows the baud rate to be variable and is set by Timer 1 of the 8051.
In serial mode 1, for each character a total of 10 bits are transferred, where the first bit is the start bit, followed by 8 bits of data, and finally 1 stop bit.

Baud Rate :

- For Synchronous data communication, Baud rate is Bits/ Second.
- For Asynchronous Data communiaction, Baud rate is the reciprocal of the time to send one bit, because the data is preceded by a start and followed by a stop bit. it need not be equal to the bits/ second always.
- For error free communication, the baud rate , no. of data bits, No. of start and stop bits , presence or absence of parity bit etc should be same for Transmitter and Receiver.

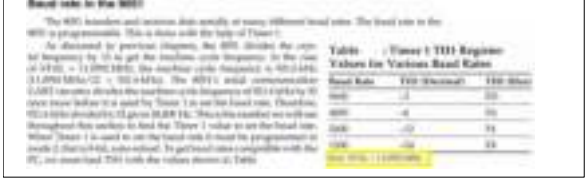
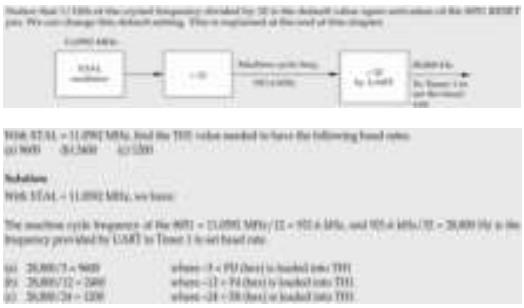


Table : Timer 1 TH1 Register Values for Various Baud Rates

Baud Rate	TH1 (Decimal)	TH1 (Hex)
9600	-3	FD
4800	-6	FA
2400	-12	F4
1200	-24	E8



PCON Register:

- By default SMOD bit is Zero. Using program we can make it as '1' to double the baud rate.

Table : Baud Rate Comparison for SMOD = 0 and SMOD = 1

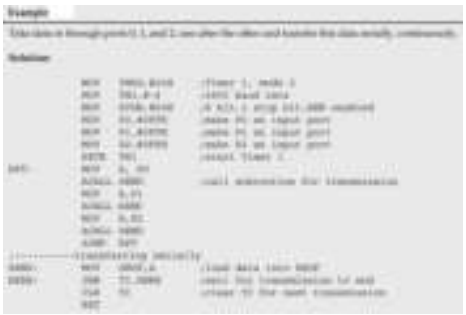
TH1	(Decimal)	(Hex)	SMOD = 0	SMOD = 1
	-3	FD	9,600	19,200
	-6	FA	4,800	9,600
	-12	F4	2,400	4,800
	-24	E8	1,200	2,400

Thank You

MPES
Module 5_10

Steps to Program 8051 to transmit serial data:

1. The TMOD register is loaded with the value 20H, indicating the use of Timer 1 in mode 2 (8-bit auto-reload) to set the baud rate.
2. The TH1 is loaded with one of the values in Table 10-4 to set the baud rate for serial data transfer (assuming XTAL = 11.0592 MHz).
3. The SCON register is loaded with the value 50H, indicating serial mode 1, where an 8-bit data is framed with start and stop bits.
4. TR1 is set to 1 to start Timer 1.
5. TI is cleared by the "CLR TI" instruction.
6. The character byte to be transferred serially is written into the SBUF register.
7. The TI flag bit is monitored with the use of the instruction "JNB TI, xx" to see if the character has been transferred completely.
8. To transfer the next character, go to Step 5.



Importance of TI Flag:

By monitoring the TI flag, we make sure that we are not overloading the SBUF register. If we write another byte into the SBUF register before TI is raised, the untransmitted portion of the previous byte will be lost. In other words, when the 8051 finishes transferring a byte, it raises the TI flag to indicate it is ready for the next character.

After SBUF is loaded with a new byte, the TI flag bit must be forced to 0 by the "CLR TI" instruction in order for this new byte to be transferred.

Steps to Program 8051 to Receive data Serially:

1. The TMOD register is loaded with the value 20H, indicating the use of Timer 1 in mode 2 (8-bit auto-reload) to set the baud rate.
2. TH1 is loaded with one of the values in Table 10-4 to set the baud rate (assuming XTAL = 11.0592 MHz).
3. The SCON register is loaded with the value 50H, indicating serial mode 1, where 8-bit data is framed with start and stop bits and receive enable is turned on.
4. TR1 is set to 1 to start Timer 1.
5. RI is cleared with the "CLR RI" instruction.
6. The RI flag bit is monitored with the use of the instruction "JNB RI, xx" to see if an entire character has been received yet.
7. When RI is raised, SBUF has the byte. Its contents are moved into a safe place.
8. To receive the next character, go to Step 6.

Importance of RI Flag:

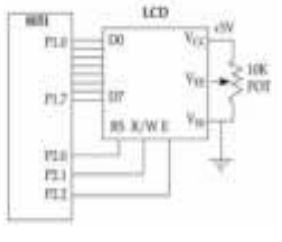
By checking the RI flag bit when it is raised, we know that a character has been received and is sitting in the SBUF register. We copy the SBUF contents to a safe place in some other register or memory before it is lost.

After the SBUF contents are copied into a safe place, the RI flag bit must be forced to 0 by the "CLR RI" instruction in order to allow the next received character byte to be placed in SBUF. Failure to do this causes loss of the received character.

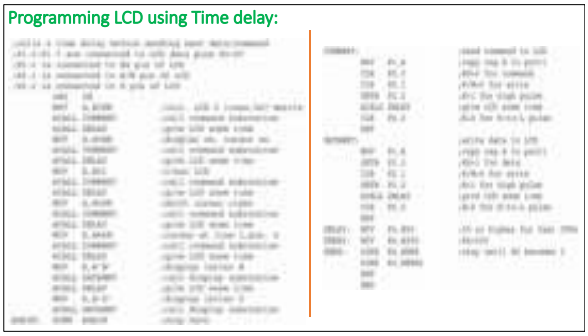
Thank You

MPES
Module 5_11

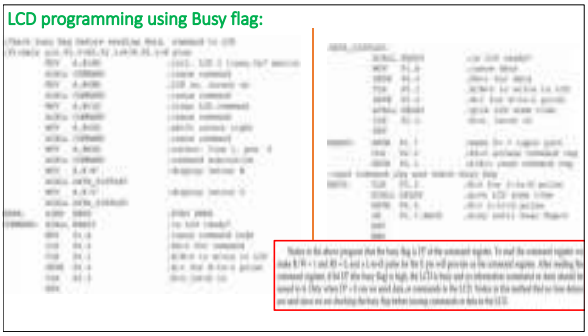
LCD Interfacing to 8051:



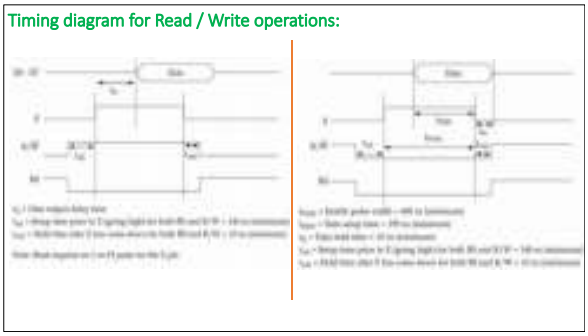
Programming LCD using Time delay:



LCD programming using Busy flag:



Timing diagram for Read / Write operations:



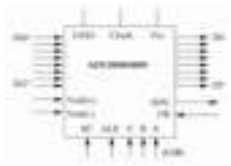
Thank You

MPES
Module 5_12

ADC Interfacing with 8051:

The resolution of the converter is the minimum analog value that can be represented by the digital data.

If the ADC gives n-bit digital output and the full scale analog input is X volts, then the resolution is $1/2^n \times X$ volts.

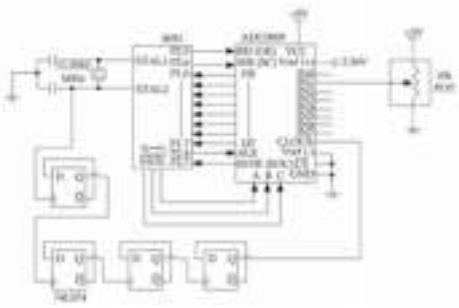


$$\text{Resolution} = \frac{V_{max} - V_{min}}{2^n} = \frac{V_{max}(+) - V_{max}(-)}{2^n}$$

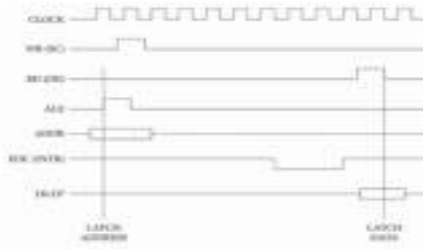
The digital data corresponding to an analog input (V_{in}) is given by:

$$\text{Digital data} = \left[\frac{V_{in}}{\text{Resolution}} - 1\right]_{10}$$

EXAMPLE



Timing diagram for ADC:



Steps to program the ADC0808/0809

The following are steps to get data from an ADC0808/0809:

1. Select an analog channel by providing bits to A, B, and C addresses according to Table 11-1.
2. Activate the ALE (address latch enable) pin. It needs an L-to-H pulse to latch in the address. See Figure 11-6.
3. Activate SC (start conversion) by an L-to-H pulse to initiate conversion.
4. Monitor EOC (end of conversion) to see whether conversion is finished. If no L output indicates that the data is converted and is ready to be picked up. If we do not use EOC, we can read the converted digital data after a fixed time delay. The delay time depends on the speed of the external clock we connect to the CLK pin. Notice that the EOC is the same as the INTR pin in other ADC chips.
5. Activate OE (output enable) to read data out of the ADC chip. An L-to-H pulse to the OE pin will bring digital data out of the chip. Also notice that the OE is the same as the RD pin in other ADC chips.

Notice in the ADC0808/0809 that there is no self-clocking and this clock must be provided from an external source to the CLK pin. Although the speed of conversion depends on the frequency of the clock connected to the CLK pin, it cannot be faster than 100 microseconds.

Programming ADC0808/0809 in Assembly

Thank You

MPES
Module 5_13

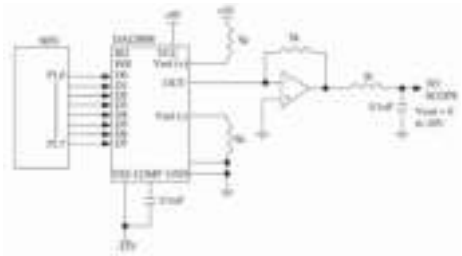
DAC Interfacing to 8051:

- Used to convert digital signals to analog signals.
- Conversion is done with R/2R ladder network logic.

$$I_{out} = I_{ref}\left(\frac{D7}{2} + \frac{D6}{4} + \frac{D5}{8} + \frac{D4}{16} + \frac{D3}{32} + \frac{D2}{64} + \frac{D1}{128} + \frac{D0}{256}\right)$$

- The number of output levels that an n bit input DAC can produce is 2^n
- The DAC requires a reference analog voltage (Vref) or current (Iref) source. The smallest possible analog value that can be represented by the n-bit binary code is called resolution. The resolution of DAC with n-bit binary input is $1/2^n$ of reference analog value. Every analog output will be a multiple of the resolution.

DAC Interfacing:



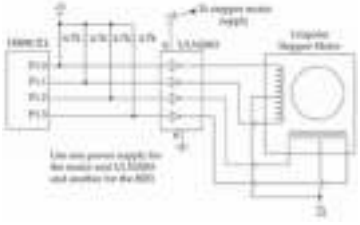
Thank You

MPES
Module 5_14

Stepper Motor Interfacing with 8051:

- Stepper motor translates electrical pulses to mechanical movement, used for position control.
- Examples are in Robotics, Dot matrix printers, Disc drives etc.
- Stepper motors generally have a permanent magnet rotor (shaft) and stator surrounding it. There are variable reluctance type stepper motors also.
- The rotor of stepper motor runs in precise steps.
- Step angle is the minimum degree of rotation associated with a single step.
- Steps per revolution is the total number of steps needed to rotate one complete revolution or 360 degrees. ie, if there are 180 steps per revolution means each step is of 2 degrees.
- Unipolar, Bipolar and Universal configurations, 6 terminals, 4 terminals and 8 terminals respectively.

when coil A is energised



Example

Describe the 8051 connection to the stepper motor of Figure and code a program to rotate it continuously.

```
        MOV   A,#99H        ;load 4-bit sequence
BACK:   MOV   P1.0          ;output sequence to motor
        RR    A             ;rotate right continuously
        ACALL DELAY         ;wait
        SJMP  BACK          ;keep going
;
DELAY:
        MOV   R1,#255
D1:     MOV   R2,#255
D2:     DJNZ  R2,$
        DJNZ  R1,D1
        RET
```

Change the value of DELAY to set the speed of rotation.
We can use the single bit instructions SETB and CLR instead of RR A to rotate the sequence.

Thank You

MPES
Module 5_15

8051 Programming in 'C' :

Why Program the 8051 in C?

Compilers produce hex files that are download into the ROM of the microcontroller. The size of the hex file produced by the compiler is one of the main concerns of microcontroller programmers, for two reasons:

1. Microcontrollers have limited on-chip ROM.
2. The code space for the 8051 is limited to 64K bytes.

How does the choice of programming language affect the compiled program size? While Assembly language produces a hex file that is much smaller than C, programming in Assembly language is tedious and time consuming. C programming, on the other hand, is less time consuming and much easier to write, but the hex file size produced is much larger than if we used Assembly language. The following are some of the major reasons for writing programs in C instead of Assembly:

1. It is easier and less time consuming to write in C than Assembly.
2. C is easier to modify and update.
3. You can use code available in function libraries.
4. C code is portable to other microcontrollers with little or no modification.

Data Types in 8051 C Programming:

Unsigned char

Since the 8051 is an 8-bit microcontroller, the character data type is the most natural choice for many applications. The unsigned char is an 8-bit data type that takes a value in the range of 0 – 255 (00 – FFH). It is one of the most widely used data types for the 8051. In many situations, such as using a counter value, where there is no need for signed data we should use the unsigned char instead of the signed char. Remember that C compilers use the signed char as the default if we do not put the keyword unsigned in front of the char.

Signed char

The signed char is an 8-bit data type that uses the most significant bit (D7 of D7 – D0) to represent the + or – value. As a result, we have only 7 bits for the magnitude of the signed number, giving us values from –128 to +127. In situations where + and – are needed to represent a given quantity such as temperature, the use of the signed char data type is a must.

Unsigned int

The unsigned int is a 16-bit data type that takes a value in the range of 0 to +65535 (0000 – FFFFH). In the 8051, unsigned int is used to define 16-bit variables such as memory addresses. It is also used to get counter values of more than 256. Since the 8051 is an 8-bit microcontroller and the int data type takes two bytes of RAM, we must not use the int data type unless we have to. Since registers and memory accesses are 8-bit (1byte), the manipulation of int variables will result in a larger hex file.

Signed int

Signed int is a 16-bit data type that uses the most significant bit (D15 of D15 – D0) to represent the + or – value. As a result, we have only 15 bits for the magnitude of the number, or values from –32,768 to +32,767.

Bit (single bit)

The bit keyword is a widely used 8051 C data type designed specifically to access single-bit addressable registers. It allows access to the single bits of the SFR registers.
Among the SFRs that are widely used and are bit-addressable are ports P0 – P3. We can use bit to access the individual bits of the ports.

Bit and sfr

The bit data type allows access to single bits of bit-addressable memory space 20 – 2FH. Notice that while the sfr data type is used for the addressable SFRs, the bit data type is used for the bit-addressable section of RAM space 20 – 2FH. To access the byte-wide SFR registers, we use the sfr data type.

Table: Some Widely Used Data Types for 8051 C

Data Type	Size in Bits	Data Range/Usage
unsigned char	8-bit	0 to 255
signed char	8-bit	-128 to +127
unsigned int	16-bit	0 to 65535
signed int	16-bit	-32,768 to +32,767
sfr	1-bit	SFR bit addressable only
bit	1-bit	RAM bit addressable only
sfr	8-bit	RAM addresses 80 – FFH only

Example

Write an 8051 C program to send values 00 – FF to port P1.

Solution:

```
#include <reg51.h>
void main(void)
  {
    unsigned char x;
    for(x=0;x<=255;x++)
      P1=x;
  }
```

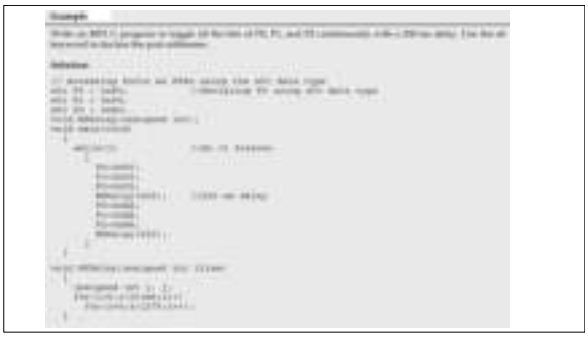
Run the above program on your simulator to see how P1 displays values 00 – FFH in binary.

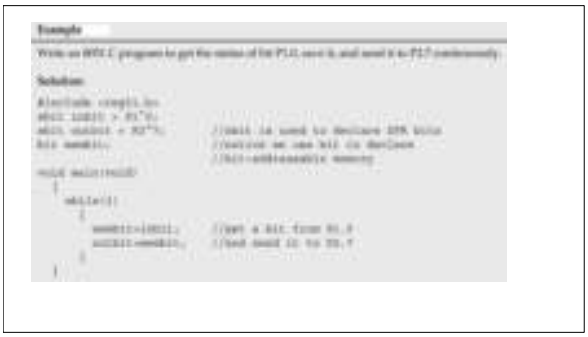
Example

Write an 8051 C program to get a byte of data from P1, wait 1/2 second, and then send it to P2.

Solution:

```
#include <reg51.h>
void MSDelay(unsigned int);
void main(void)
{
  unsigned char mybyte;
  P1=0xFF;              //make P1 an input port
  while(1)
    {
      mybyte=P1;        //get a byte from P1
      MSDelay(500);
      P2=mybyte;        //send it to P2
    }
}
void MSDelay(unsigned int itime)
{
  unsigned int i, j;
  for(i=0;i<itime;i++)
    for(j=0;j<1275;j++);
}
```





Thank You

MPES
Module 5_16

Embedded System:

An Embedded system is a system that has a software embedded in to a computer hardware for doing a dedicated task. It may be an independent system or a part of large system.

Eg: Mobile phones, TV remotes, Printers etc.

Characteristics of Embeddded systems:

- Reliability
- Cost effectiveness
- Low power consumption
- Fast execution time
- Efficient use of memory
- Processing power is more

CPU of the embedded system can be Micro processor, Micro Controller or DSP or any Application Specific Processor (ASP).

Application Domain of Embedded Systems:

- In every element of modern day life, we could find an application of Embedded systems. Following are a few among them,
  - **Consumer electronics:** Mobile phones, Digital cameras, Printers, Washing machines, Remote controls, Toys etc.
  - **Home security systems:** Intruder and Fire alarm system etc
  - **Automobile:** Anti lock Braking (ABS), Engine control unit, Electronic fuel injection,Door and wiper controls etc.
  - **Medical equipments:** Scanners, ECG and EEG, Testing and monitoring equipments.
  - **Banking:** ATM, Currency counting machine etc.
  - **Networking:** Routers, Switches etc.
  - **Factories:** Control, Instrumentation and Alarm systems.

The list is incomplete and we could new domains for embedded system applications every where.

Features and characteristics of Embedded systems:

- It should perform one or a small number of dedicated functions efficiently.
- Low power consumption as most devices are battery powered.
- It should have limited number of memory and peripherals.
- Application is not supposed to get altered by the user.
- Many of them are not directly accessible as it may be a part fo the large system.
- It should be highly reliable.
- Many of the need to operate with time contraints (Speed of response).
- Physical size is a constraint for many systems.
- Code size is another constraint for many systems.

Current Trends:

- Multi-core processors:

It has become very clear that trying to improve processor performance by increasing clock frequencies is fraught with difficulties, because the direct result of higher clock frequency is high power dissipation. Thus, the option of using more than one processor core (at lower clock frequencies) is being tried out. Thus, the current smart phones and gaming consoles use multi-core processors.

- Embedded and real-time operating systems:

With the emergence of complex applications, many new embedded and real-time operating systems have become popular. Linux has emerged as a popular embedded OS, and others like Android and newer versions of Symbion have came up for mobile applications and handheld devices.

- Newer areas of deployment of embedded devices:

Embedded devices have applications in the entertainment, healthcare and automotive segments. Besides that, there are applications in the communication and military fields as well. Research and development in these fields is going ahead.

Thank You

MPES
Module 5_17

Real time Task:

- It is a task in which the performance is judged on the basis of time.
- It means that the result of computation is 'correct' only it has produced the correct output within the specified time constraint , failure to that is considered as a system failure or as a reduced 'quality of service'.
- Process control systems in Industries, Weapon guidance system, Air traffic control, Anti lock braking systems etc are few examples for Real time systems.
- All embedded systems are not Real time systems, eg. Printer, Mobile phones etc.

Types of Real time tasks:

- **Hard Real time systems:**

In Hard real time systems the failure to meet the time deadline is a fatal fault. Eg. Air traffic control, ABS etc.

- **Soft real time systems:**

Breaking the time deadline is unwanted, but not immediately critical.

Eg. Navigation systems

- **Firm Real time systems:**

If the deadline is missed occassionally, the system won't fail. The results produced after the deadline is discarded.

Gaming console etc can be an example.

Real time operating systems [RTOS]:

Characteristics:

- Time constraints
- Correctness [logical and Time]
- Task scheduling
- Safety and Reliability
- Task Criticality - Cost of failure of task.
- Custom hardware
- Responsive and reactive
- Stable
- Exception handling

Eg. RT Linux, Vx Works, Windows NT, Solaris etc....

Thank You

MPES
Module 5_18

Embedded Product Development Cycle [EDLC]:

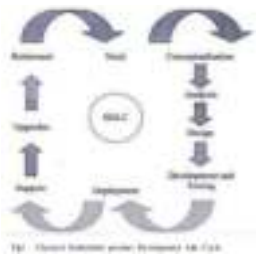
- It is an Analysis - Design - Implementation based problem solving approach for Embedded product development.
- First analyse what product is to be developed, next to find the good the design to build it and finally to Implement or develop it.
- EDLC is essential for understanding the scope and complexity of the work involved in any embedded product development.

Objectives of EDLC:

- Ensure that high quality products are delivered to end user.
- Risk minimisation and defect prevention in product development by project management.
- Maximise the productivity.

Different phases of EDLC:

- The life cycle of product development is commonly reffered to as models.
- The classic embedded life cycle management contains the phases, Need - Conceptualisation - Analysis - Design - Development and Testing - Deployment - Support - upgrades - Retirement/disposal.



EDLC Models:

Water fall or Linear Model:

- In this model or approach each phase of EDLC is executed in sequence.
- In linear model each phases are well documented, which gives a clear insight to what to be done and how it is to be done in the next phase.
- One significant feature in linear model is, even if a fault is identified in a phase, the corrective measures is not done immediately as there is no feedback given in the model.
- Good documentation, easy project management, Good control over cost and schedule.
- Well suited for the product development where the requirements are well defined and within the scope, no changes are expected till the end of the cycle.

← Water fall or Linear model

Thank You

MPES
Module 5_19

Iterative/ Incremental or Fountain Model of EDLC:

- It can be viewed as a cascaded model of linear models, where the cycles are repeated till the requirements are met completely.
- The major advantage is that it provides a very good deployment cycle feedback after each function or feature implementation and hence the data can be used as reference for development of similar products in future.
- Since each cycle acts as a maintenance phase of the previous cycle, changes in functions or features can be easily incorporated and hence more responsive to changing user needs.
- Risk is spread across each cycle with limited features and can be minimised easily.
- Project management and testing is much simpler to linear model.

Tool Chain System:

- A toolchain is the set of tools that compiles source code into executables that can run on your target device, and includes a compiler, a linker, and

Loaders:

In computer systems a loader is the part of an operating system that is responsible for loading programs and libraries. It is one of the essential stages in the process of starting a program, as it places programs into memory and prepares them for execution.

Loading a program involves reading the contents of the executable file containing the program instructions into memory, and then carrying out other required preparatory tasks to prepare the executable for running. Once loading is complete, the operating system starts the program by passing control to the loaded program code.

Debuggers:

- A debugger or debugging tool is a computer program used to test and debug other programs (the "target" program). The main use of a debugger is to run the target program under controlled conditions that permit the programmer to track its operations in progress and monitor changes in computer resources (most often memory areas used by the target program or the computer's operating system) that may indicate malfunctioning code.

- Typical debugging facilities include the ability to run or halt the target program at specific points, display the contents of memory, CPU registers or storage devices (such as disk drives), and modify memory or register contents in order to enter selected test data that might be a cause of faulty program execution.

Profilers:

- Profiling is the process of measuring an application or system by running an analysis tool called a profiler. Profiling tools can focus on many aspects: functions call times and count, memory usage, cpu load, and resource usage. Used to optimise the system.

Test Coverage:

Test coverage is defined as a technique which determines whether our test cases are actually covering the application code and how much code is exercised when we run those test cases. If there are 10 requirements and 100 tests created and if 90 tests are executed then test coverage is 90%.

Thank You