**NOTES**

**SUBJECT: MICROPROCESSORS**

**SUBJECT CODE: NEC-503**

**BRANCH: ECE**

**SEM: V SEM**

**SESSION: 2016-17**

**Er. Ajay Dwivedi**

**Assistant Professor**

**ECE Dept.**

**INDEX**

**CHAPTER 2**

1. Basic interfacing concepts
2. Interfacing output displays
3. Interfacing input devices
4. Memory mapped I/O
5. Flow chart symbols
6. Data Transfer operations
7. Arithmetic operations
8. Logic Operations
9. Branch operation
10. Writing assembly language programs
11. Programming techniques: looping, counting and indexing

**8085 Bus Structure:**

Address Bus: The address bus is a group of 16 lines generally identified as A0 to A15.

The address bus is unidirectional: bits flow in one direction-from the MPU to peripheral devices.The MPU uses the address bus to perform the first function: identifying a peripheral or a memory location.

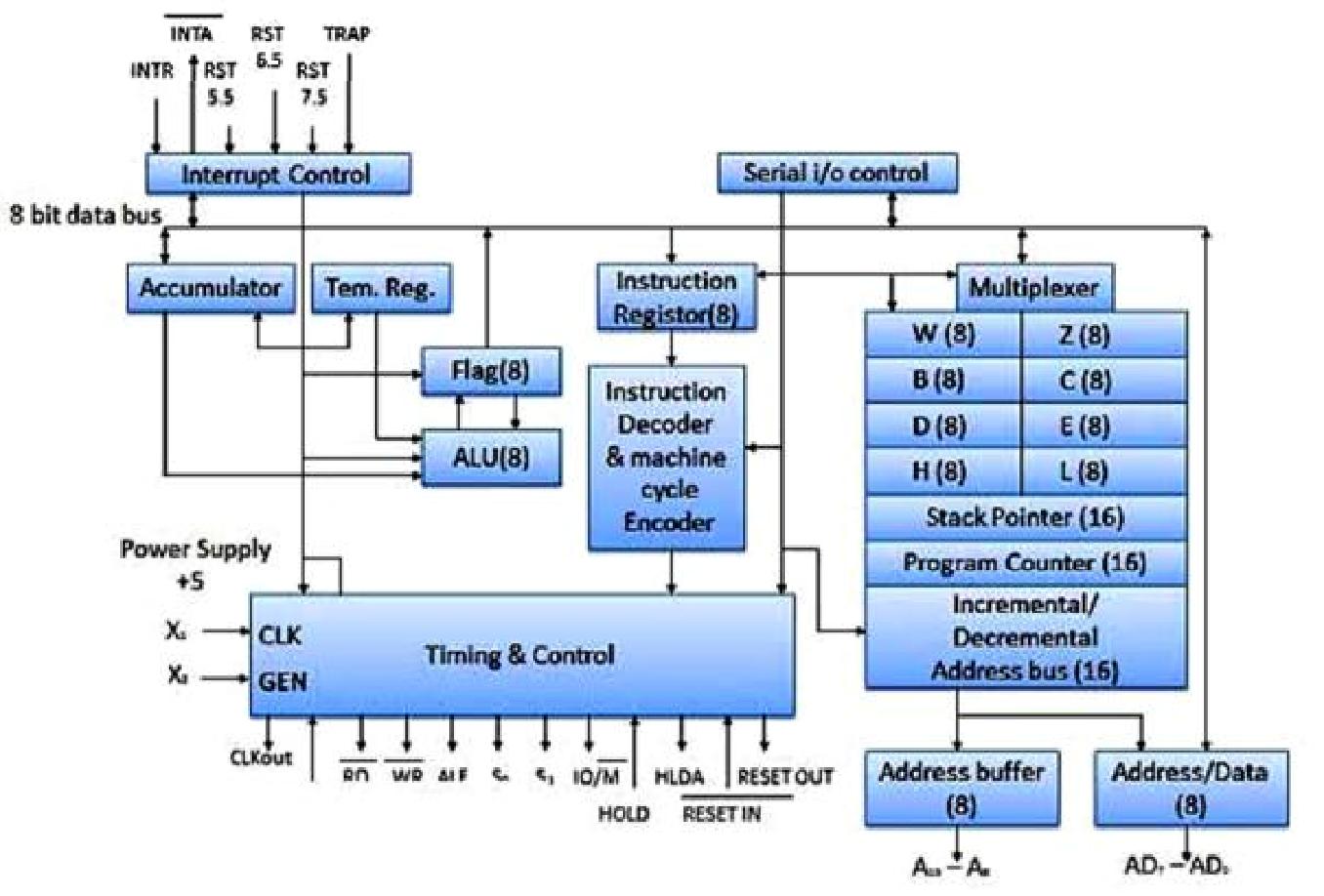


Fig. 1.19: Block Diagram of 8085 Microprocessor

Data Bus: The data bus is a group of eight lines used for data flow. These lines are bi-directional - data flow in both directions between the MPU and memory and peripheral devices. The MPU uses the data bus to perform the second function: transferring binary information. The eight data lines enable the MPU to manipulate 8-bit data ranging from 00 to FF (28 = 256 numbers).The largest number that can appear on the data bus is 11111111.

Control Bus: The control bus carries synchronization signals and providing timing signals. The MPU generates specific control signals for every operation it performs. These signals are used to identify a device type with which the MPU wants to communicate.

Registers of 8085:The 8085 have six general-purpose registers to store 8-bit data during program execution.These registers are identified as B, C, D, E, H, and L.They can be combined as register pairs-BC, DE, and HL-to perform some 16-bit operations.

Accumulator (A):

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

Flags:

* The ALU includes five flip-flops that are set or reset according to the result of an operation.
* The microprocessor uses the flags for testing the data conditions.
* They are Zero (Z), Carry (CY), Sign (S), Parity (P), and Auxiliary Carry (AC) flags. The most commonly used flags are Sign, Zero, and Carry.

The bit position for the flags in flag register is,



Fig.1.20 Flag register

1. Sign Flag (S): After execution of any arithmetic and logical operation, if D7 of the result is 1, the sign flag is set. Otherwise it is reset.D7 is reserved for indicating the sign; the remaining is the magnitude of number. If D7 is 1, the number will be viewed as negative number. If D7 is 0, the number will be viewed as positive number.
2. Zero Flag (z): If the result of arithmetic and logical operation is zero, then zero flag is set otherwise it is reset.
3. Auxiliary Carry Flag (AC): f D3 generates any carry when doing any arithmetic and logical operation, this flag is set. Otherwise it is reset.
4. Parity Flag (P): If the result of arithmetic and logical operation contains even number of 1's then this flag will be set and if it is odd number of 1's it will be reset.
5. Carry Flag (CY):If any arithmetic and logical operation result any carry then carry flag is set otherwise it is reset.

Arithmetic and Logic Unit (ALU):It is used to perform the arithmetic operations like addition, subtraction, multiplication, division, increment and decrement and logical operations like AND, OR and EX-OR.It receives the data from accumulator and registers. According to the result it set or reset the flags.

Program Counter (PC): This 16-bit register sequencing the execution of instructions. It is a memory pointer. Memory locations have 16-bit addresses, and that is why this is a 16-bit register. The function of the program counter is to point to the memory address of the next instruction to be executed. When an opcode is being fetched, the program counter is incremented by one to point to the next memory location.

Stack Pointer (Sp): The stack pointer is also a 16-bit register used as a memory pointer. It points to a memory location in R/W memory, called the stack. The beginning of the stack is defined by loading a 16-bit address in the stack pointer (register).

Temporary Register: It is used to hold the data during the arithmetic and logical operations.

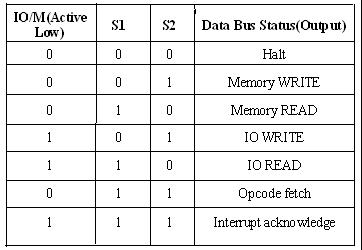
* Instruction Register: When an instruction is fetched from the memory, it is loaded in the instruction register.
* Instruction Decoder: It gets the instruction from the instruction register and decodes the instruction. It identifies the instruction to be performed.

* Serial I/O Control: It has two control signals named SID and SOD for serial data transmission.

Timing and Control unit.

* It has three control signals ALE, RD (Active low) and WR (Active low) and three status signals IO/M(Active low), S0 and S1.
* ALE is used for provide control signal to synchronize the components of microprocessor and timing for instruction to perform the operation.
* RD (Active low) and WR (Active low) are used to indicate whether the operation is reading the data from memory or writing the data into memory respectively.
* IO/M(Active low) is used to indicate whether the operation is belongs to the memory or peripherals.

Table 1.3 Machine cycle status and control signals



**1.5.1The 8085 machine cycles and bus timings:** Introduction: A machine cycle is the timerequired to complete one operation of accessing the memory, I/O or acknowledge an external signal or request. Usually machine cycle consists of 3 to 6 T-states. In this article let us discuss about their different types and how they are being classified.

Types of machine cycle

There are various types of machine cycles which are classified based onStatus signals (IO/M’, S1 and S0) ,Control Signals (RD’, WR’, INTA).

The different types of machine cycle available in 8085 microprocessor are:

Opcode Fetch

Memory Read

Memory write

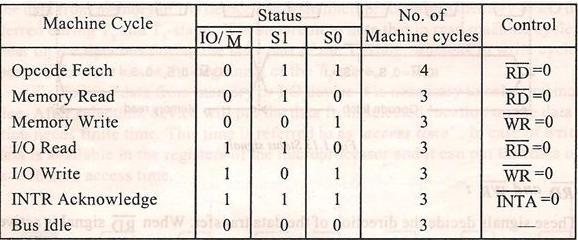
I/O Read

I/O Write

INTR Acknowledge

Bus Idle

Table 1.4: 8085 machine cycle status and control signals



**Opcode fetch machine cycle:** The Opcode fetch cycle, fetches the instructions from memory anddelivers it to the instruction register of the microprocessor. For any instruction cycle, Opcode fetch is the first machine cycle. We know that each machine cycle may have 3 to 6 T-states. This Opcode fetch machine cycle consists of 4 T-states.

T1 State: During the T1 state, the contents of the program counter are placed on the 16 bit address bus. The higher order 8 bits are transferred to address bus (A8-A15) and lower order 8 bits are transferred to multiplexed A/D (AD0-AD7) bus.

After the address bits are transferred, the ALE (address latch enable) signal goes high. As soon as ALE goes high, the memory latches the AD0-AD7 bus. At the middle of the T state the ALE goes low and the complete 16-bit address is made available for the Opcode fetch machine cycle.

T2 State: During the beginning of this state, the RD’ signal goes low to enable memory. It is during this state, the selected memory location is placed on D0-D7 of the Address/Data multiplexed bus.

T3 State: In the previous state the Opcode is placed in D0-D7 of the A/D bus. In this state of the cycle, the Opcode of the A/D bus is transferred to the instruction register of the microprocessor. Now the RD’ goes high after this action and thus disables the memory from A/D bus.

T4 State: In this state the Opcode which was fetched from the memory is decoded.

Thus the cycle completes after 4 T-states. This very well explains the Opcode fetch machine cycle. For better understanding of the concept, a diagram explaining Opcode fetch cycle is shown below.

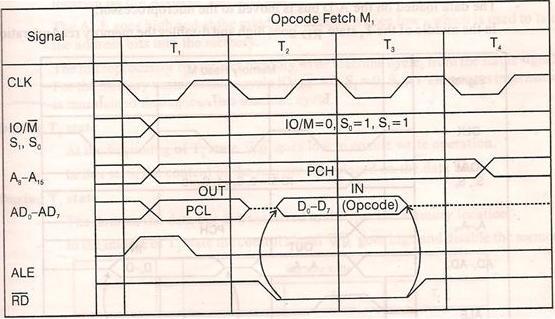


Fig 1.21 Memory Read machine cycle

If the instruction is only one byte in length, then one machine cycle is enough to complete the process (Opcode fetch cycle). When the instruction has more than one byte of information to be processed, then the microprocessor may require more than one machine cycle to complete the process. The machine cycle in this case would require reading of address or data from memory or any other I/O devices. Hence these are known as memory read or I/O read machine cycles. These machine cycles have 3 T-states.

A simple example for memory read machine cycle is MVI D, 24H

For the above example there are 2 machine cycles involved. One is the Opcode fetch cycle and the second one is the memory read cycle which transfers the operand 24H from the memory to the microprocessor.

T1 state: In this state the contents of the program counter is placed on the higher order address bus (A8-A15) and lower order address and data multiplexed (AD0-AD7) bus. ALE goes high so that the memory latches the (AD0-AD7) and then during the middle of the T1 state ALE goes low, so that complete 16-bit address are available.

The microprocessor then identifies the memory read machine cycle from the status signals IO/M’=0, S1=1, S0=0. This condition indicates the memory read cycle.

T2 state: Selected memory location is placed on the (D0-D7) of the A/D multiplexed bus.

T3 State: The data which was loaded on the previous state is transferred to the microprocessor. In the middle of the T3 state RD’ goes high and disables the memory read operation. The data which was obtained from the memory is then decoded.

The concept can be understood better with the aid of the diagram shown below.

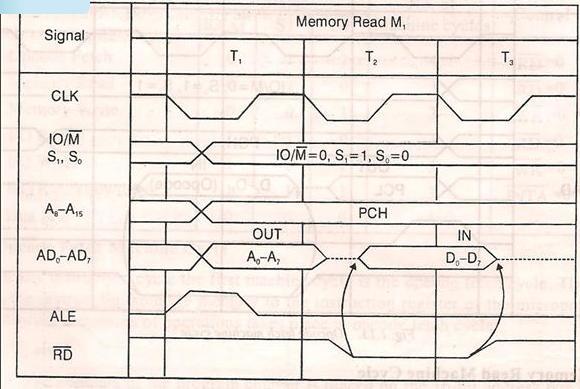


Fig.1.22 : I/O read machine cycle

This machine cycle is very similar to memory read machine cycle. It is a 2 byte-I/O read instruction.

A simple example is IN 22H.

The first machine cycle is same as the memory read machine cycle, which is the Opcode fetch cycle. The second machine cycle is the I/O read machine cycle, where the content of port addresses (22H in this case) is transferred to the microprocessor.

The status signal for the I/O read machine cycle is different. The status signal values are IO/M’=1, S1=1, S0=0.

In the next article let us continue with the other 4 types of the machine cycle.

**1.6 MEMORY INTERFACING :** Microprocessor need to access memory quite frequently to readinstructions and data stored in memory; the interface circuit enables that access.

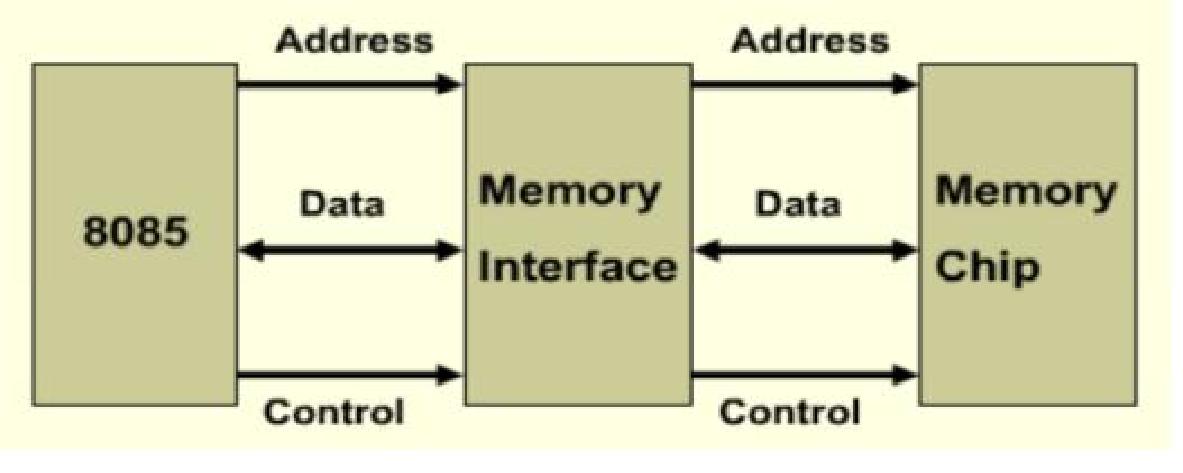


Fig.1.23 Memory Interfacing

The interface process involves designing a circuit that will match the memoryrequirements with the microprocessor signal.[Memory has certain signalrequirements to read from and write into memory. SimilarlyMicroprocessor initiates the set of signals when it wants to read fromand write into memory].

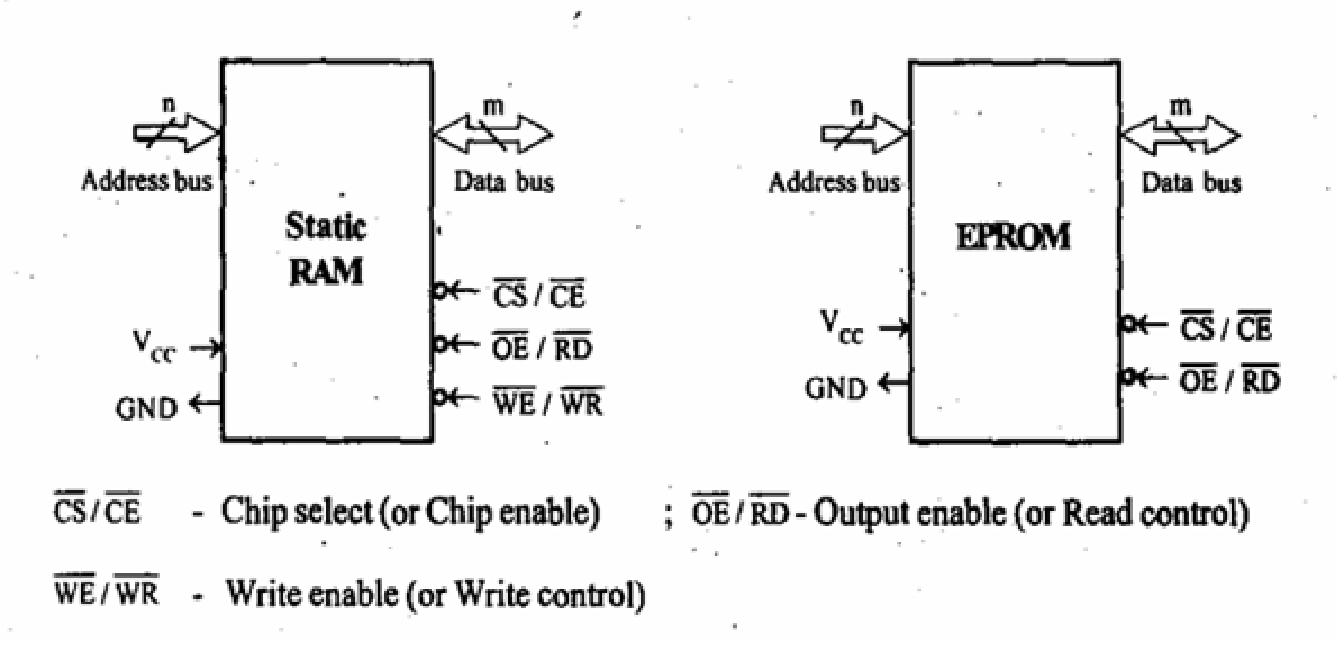


Fig 1.24: Static RAM and EPROM memory

Accessing memory can be summarized into the following three steps:

* Select the chip.
* Identify the memory register.
* Enable the appropriate buffer.

And to Translating this to microprocessor domain following steps are required:

* The microprocessor places a 16-bit address on the address bus.
* Part of the address bus will select the chip and the other part will go through the address decoder to select the register.
* The signals IO/M and RD combined indicate that a memory read operation is in progress. The MEMR signal can be used to enable the RD line on the memory chip.

Address Decoding : The result of ‘address decoding’ is the identification of a register for a given address. A large part of the address bus is usually connected directly to the address inputs of the memory chip. This portion is decoded internally within the chip. What concerns us is the other part that must be decoded externally to select the chip. This can be done either using logic gates or a decoder.

Interfacing circuit :

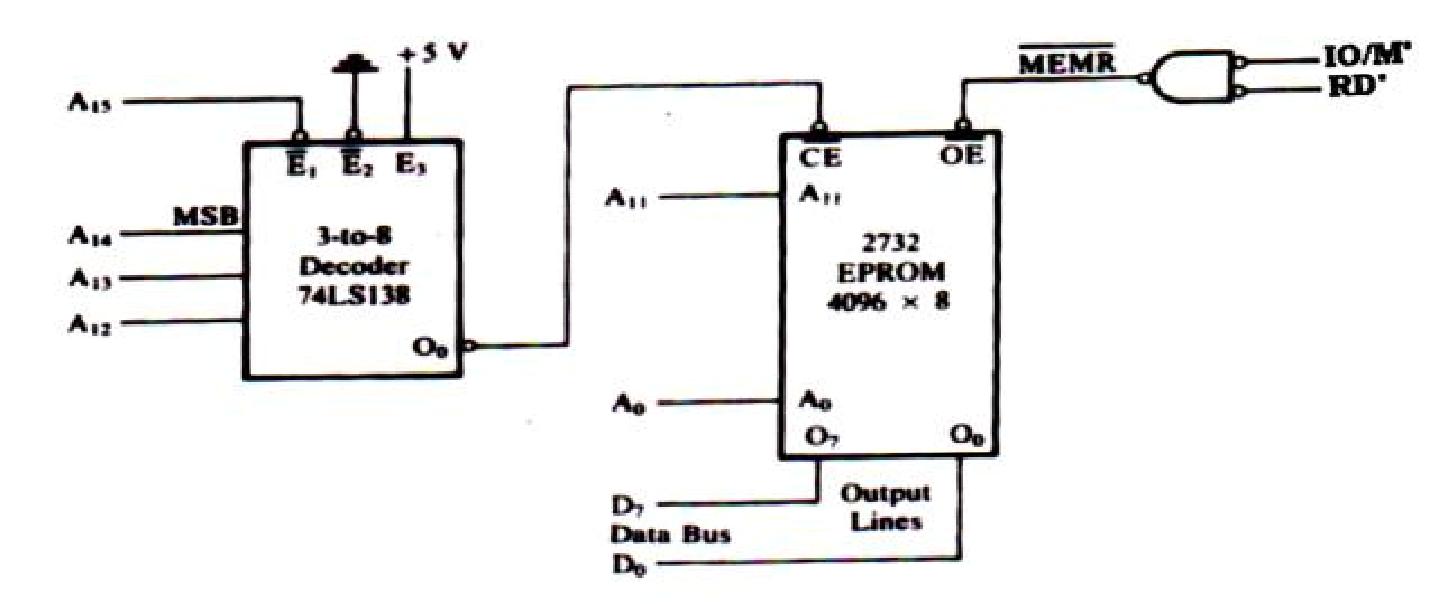


Fig 1.25 Interfacing 2732 EPROM with 8085 Microprocessor

The 8085 address lines A11-A0 are connected to the pins A11-A0 of the memory chip.Decoder decode A15-A12 and output O0 is connected to CE’ which is asserted onlywhen A15-A12 is 0000 (A15 low enables decoder and input 000 asserts the output O0).One control signal MEMR’ is connected to OE’ to enable output buffer.

Example: Interface a 4K EPROM, one 4K RAM and one 8K RAM to a microprocessor with the following Memory Map.

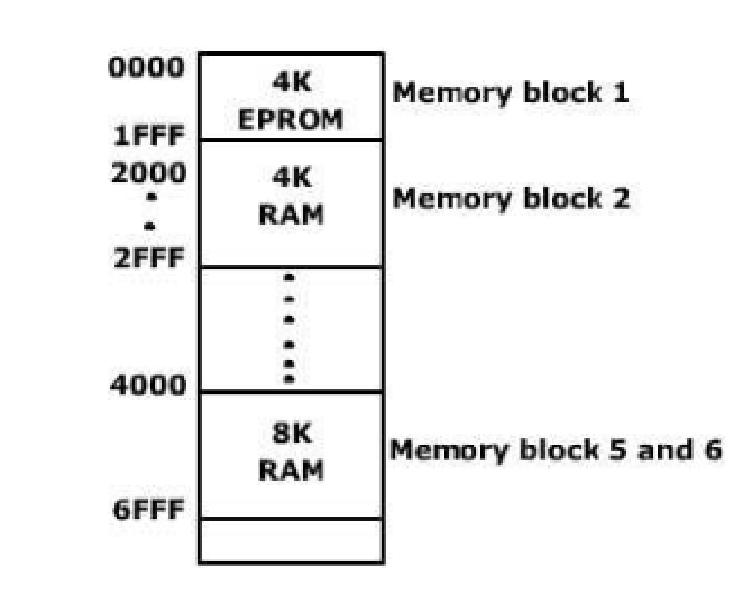


Fig. 1.26 Memory Map

A memory chip select decoder is used to provide chip select signal for eachmemory device (IC). This will decide the address range that is allotted for each memoryIC. 74LS138 is a 3 to 8 decoder and it can be used for this purpose. In this example theminimum memory block size is 4K. To access 1K locations 10 address lines must be used(210 =1K = 1024 locations). So to access 4K locations (4 X 1K = 22 X 210 = 212) 12address lines (A0 – A11) must be used. Since 8085 has 16 address lines the decoding can

be indicated as shown below.

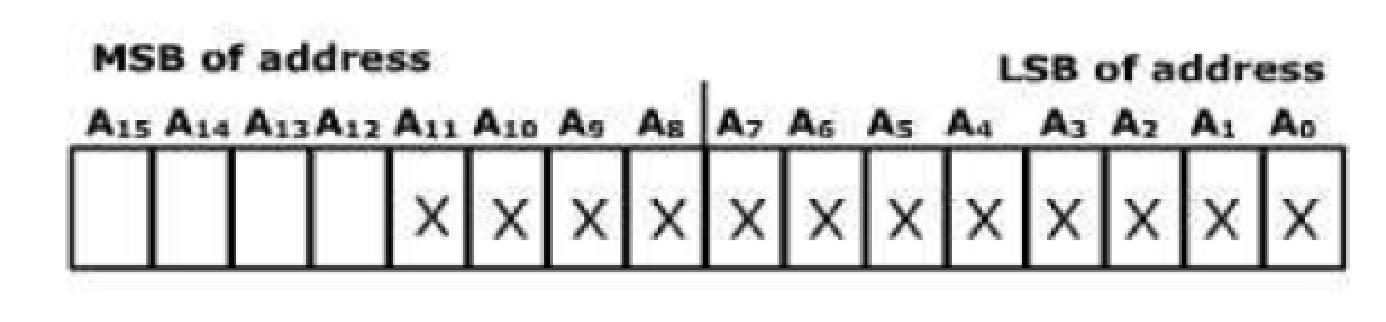


Fig 1.27 Variable address lines

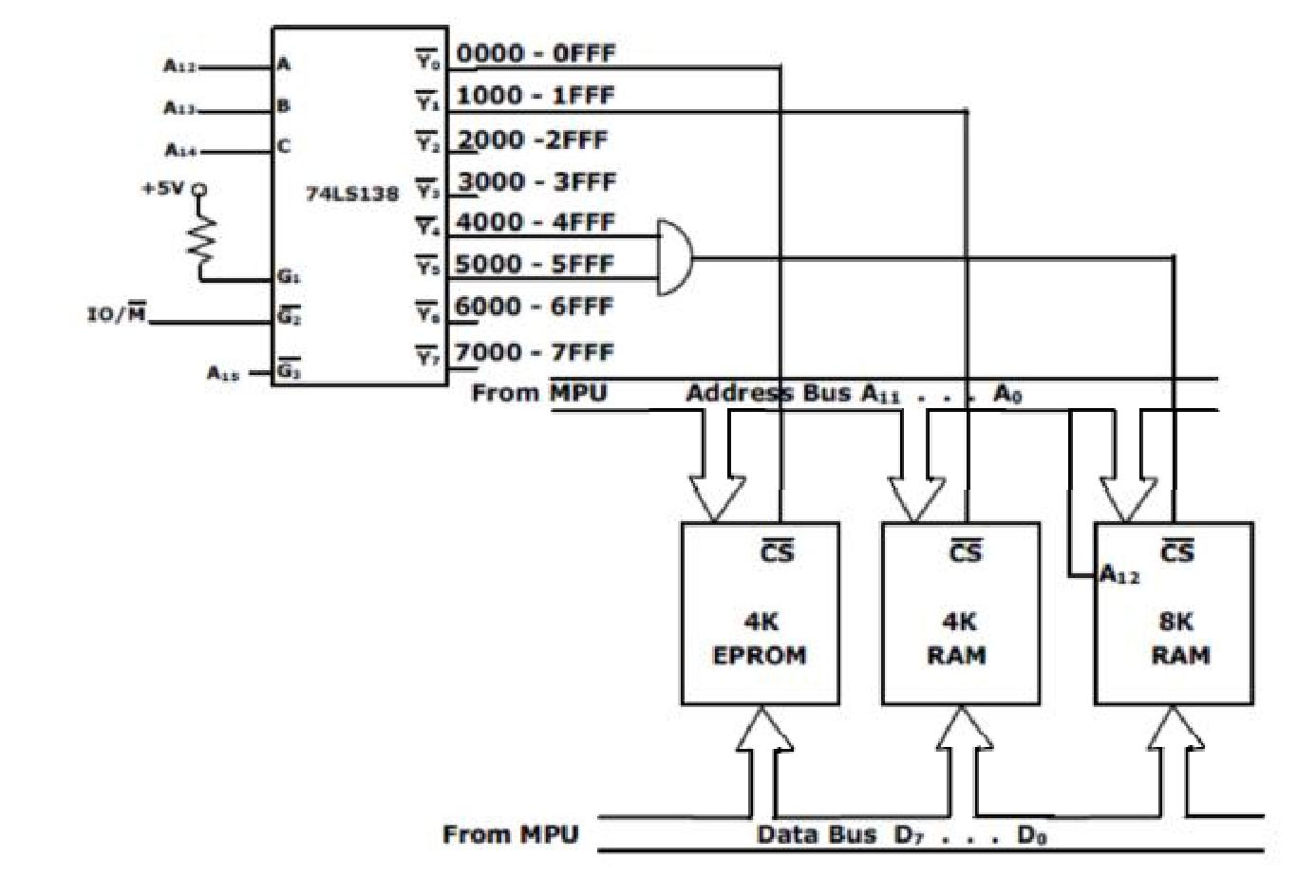


Fig. 1.28: Memory Interface Circuit

**2.1 BASIC INTERFACING CONCEPTS**

Memory is made up of (registers). Each register consists of one storage location. Each location consists of an address. The number of storage locations from few hundred to several mega or giga locations. The total number of memory storage is called memory capacity and measured in Bytes. Each register consists of storage element (FF, capacitor for semiconductor). A storage element is called cell. The data could be read from or written to memory.

**2.1.1 Memory structure and its requirements**

Read/write memories consist of an array of registers, in which each register has unique address. The size of the memory is N x M as shown below where N is the number of registers and M is the word length, in number of bits.



Fig 2.1 Logic diagram for RAM

If memory is having 12 address lines and 8 data lines, then Number of registers/ memory locations (capacity) = 2N= 212 = 4096

Word length = M bit = 8 bit

Example 2: If memory has 8192 memory locations, then it has 13 address lines. Table 2.1 shows how the number of address lines is decided depending on the size of memory.

Table 2.1 summarizes capacity with address

|  |  |  |
| --- | --- | --- |
|  | Memory Capacity | Address lines |
|  |  | required |
| 1k | = 1024 memory locations | 10 |
| 2k | = 2048 memory locations | 11 |
| 4k | = 4096 memory locations | 12 |
| 8k | = 8192 memory locations | 13 |
| 16k | = 16384 memory locations | 14 |
| 32k | = 32768 memory locations | 15 |
| 64k | = 65536 memory locations | 16 |

**2.2 INTERFACING OUTPUT DISPLAYS**

The output devices are usually slow. Also the output is usually expected to continue appearing on the output device for a long period of time. Given that the data will be present on the data lines for a very short period (microseconds), it has to be latched externally. To do this external latch should be enabled when the port’s address is present on the address bus, the IO/M signal is high and WR is set low. The resulting signal would be active when the output device is being accessed by the microprocessor.

**2.3 INTERFACING INPUT DEVICES**

The basic concepts are similar to interfacing of output devices. The address lines are decoded to generate a signal that is active when the particular port is being accessed. An IORD signal is generated by combining the IO/M and the RD signals from the microprocessor. A tristate buffer is used to connect the input device to the data bus. The control (enable) for these buffers is connected to the result of combining the address signal and the signal IORD.

**2.4 MEMORY MAPPED I/O**

Basic Memory Interfacing with 8085

For interfacing memory devices to µp 8085, following points should be considered:

* µp 8085 can access 64KB memory since address bus is 16-bit.
* Generally EPROM (or EPROMs) is used as a program memory and RAM (or RAMs) as data memory.
* The capacity of program memory and data memory depends on the application.
* It is not always necessary to select 1 EPROM and 1 RAM. We can have multiple EPROMs and multiple RAMs as per the requirement of application.

For Example

We have to implement 32 KB of program memory and 4KB EPROMs are available. In this case we can connect 8 EPROMs in parallel. We can place EPROM/RAM anywhere in full 64 KB address space. But program memory (EPROM) should be located from address 0000 H. It is not always necessary to locate EPROM and RAM in consecutive memory address. The memory interfacing requires to:

* Select the chip
* Identify the register
* Enable the appropriate buffer.
* µp system includes memory and I/O devices.
* It is important to note that µp can communicate (read/write) with only one device at a time, so address decoding needed.

Address Decoding techniques

There are two main techniques:

* Absolute decoding/ Full Decoding
* Linear decoding / Partial Decoding

**Absolute Decoding**: All the higher address lines are decoded to select the memory chip, and thememory chip is selected only for the specified logic level on these high-order address, no other logic levels can select the chip. The Fig 2.2 shows the memory interface with absolute decoding. This addressing technique is normally used in large memory systems.

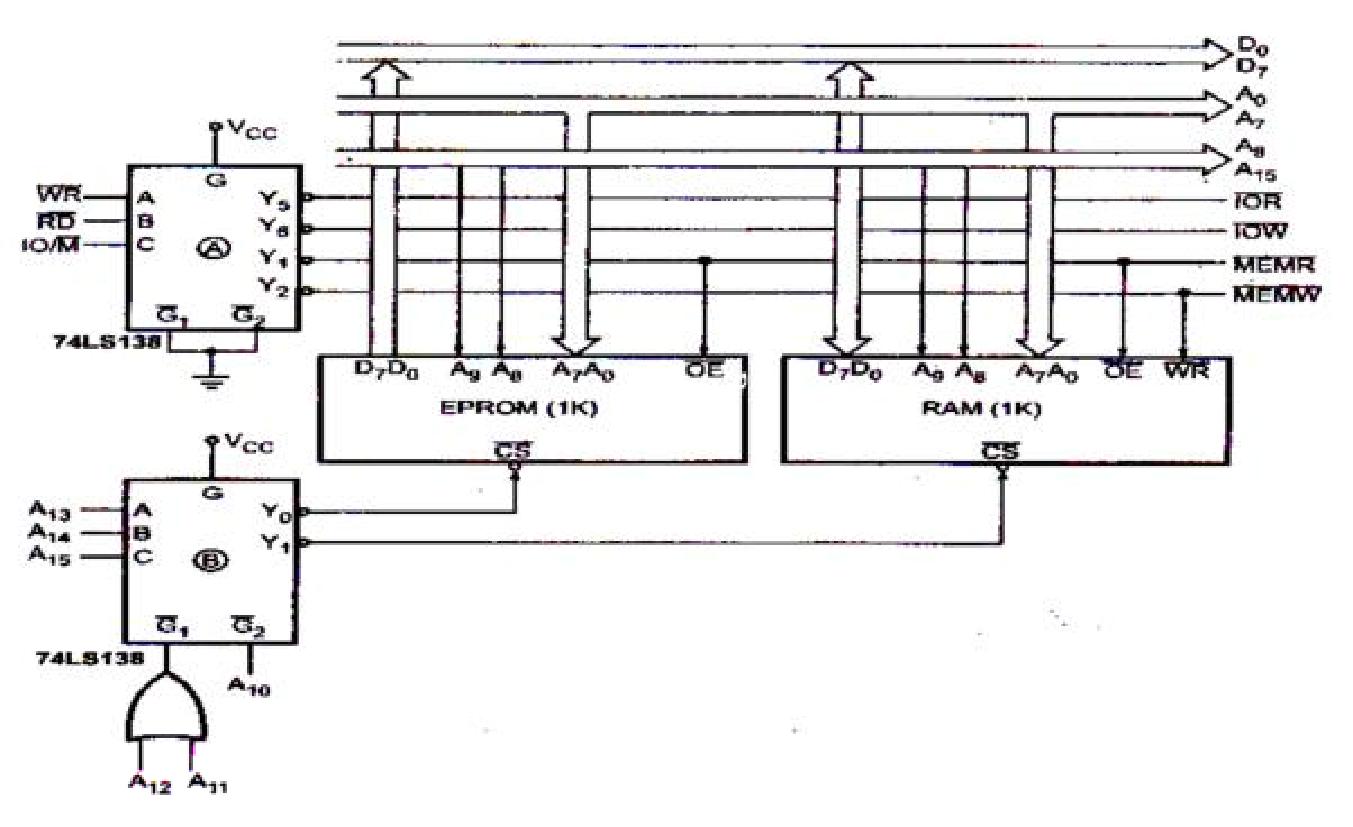


Fig 2.2 Absolute Address Decoding

**Linear Decoding**- In small systems, h/w for the decoding logic can be eliminated by usingindividual high-order address lines to select memory chips. This is referred to as linear decoding. The Fig 2.3 below shows the addressing of RAM with linear decoding technique.This technique is also called partial decoding. It reduces the cost of the decoding circuit, but it has a drawback of multiple address (shadow addresses).

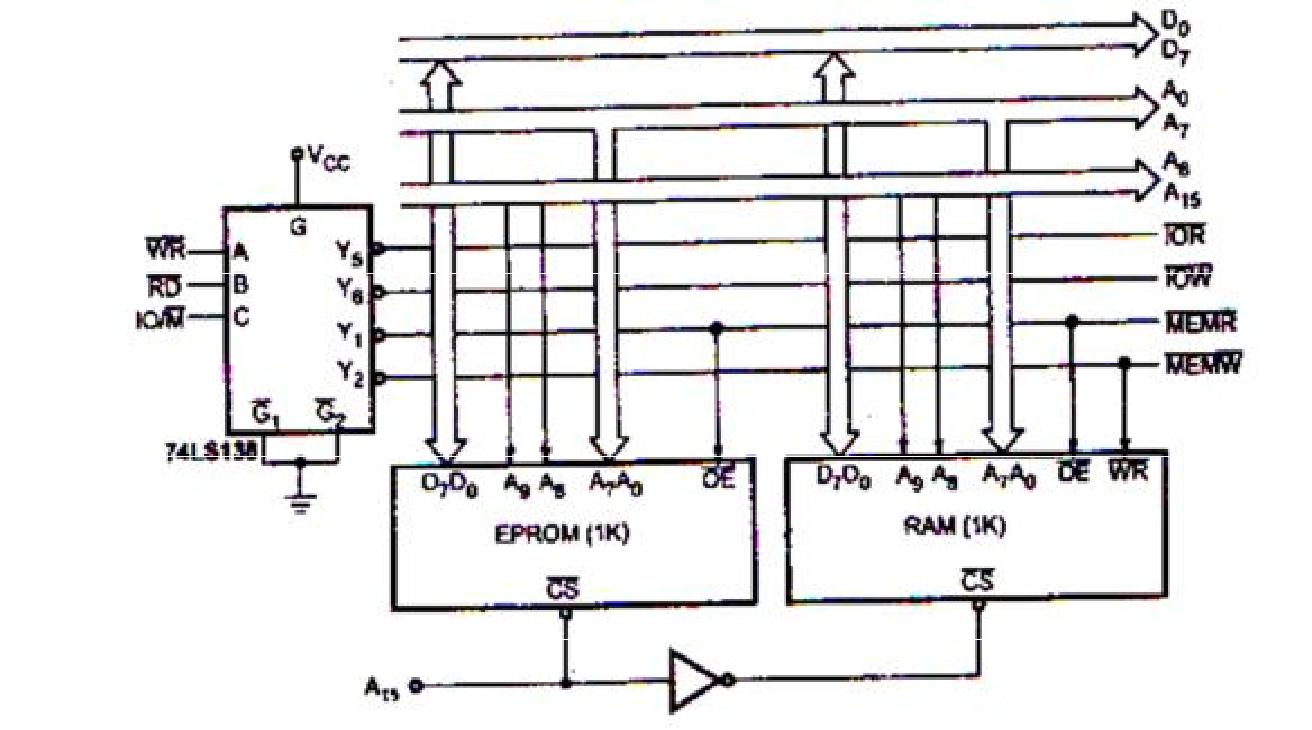


Fig 2.3 Linear Decoding

**Interfacing Examples:**

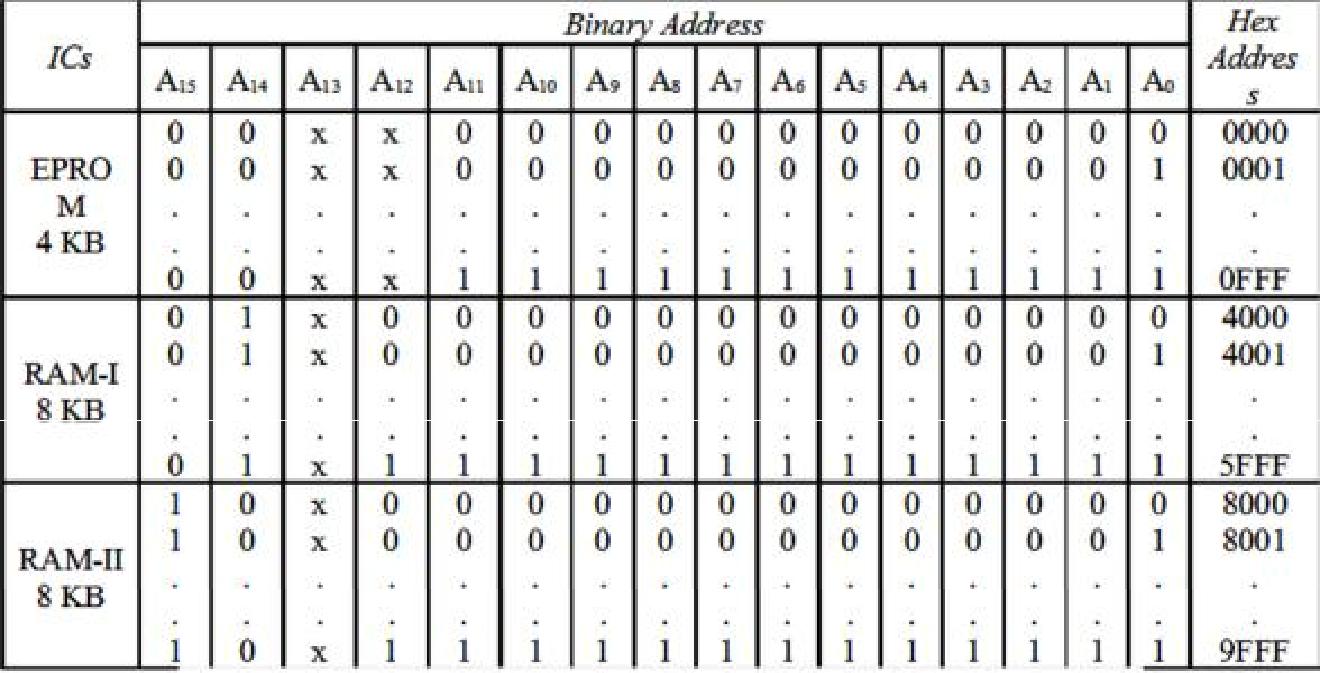
Draw the circuit diagram of an 8085 system, having a 4 KB EPROM and two 8 KB RAM ICs. The starting address of the EPROM is 0000H and that of RAM is 8000H. The address of the decoder circuits should be clearly shown.

Answer :

EPROM-4 KB (Address lines required is 12 – A0 to A11) RAM-I-8 KB (Address lines required is 13 – A0 to A12) RAM-II-8 KB (Address lines required is 13 – A0 to A12)

Mapping of Addresses to Memory Ics

Table 2.2 Address Mapping



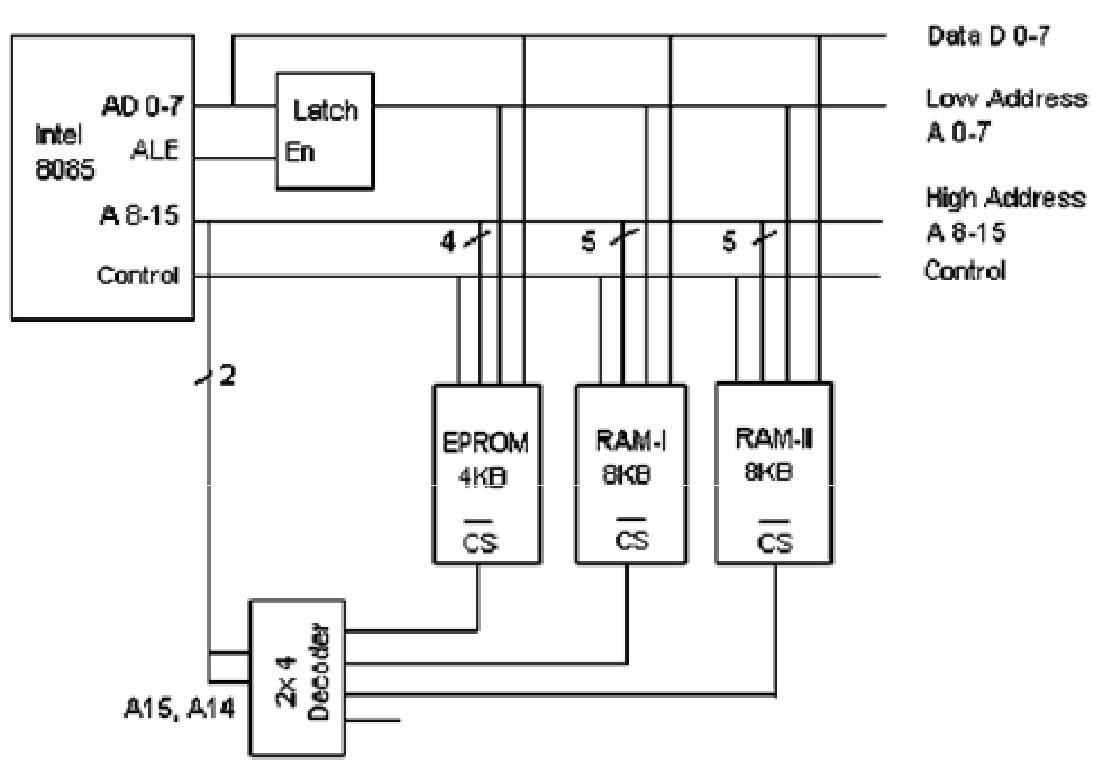


Fig 2.4 Address Decoding

**2.5 Flow chart symbols**

To develop the programming logic, programmer has to write down various actions which are to be performed in proper sequence. The flow chart is a graphical tool that allows programmer to represent various actions which are to be performed. Figure 2.5 shows the graphical symbols used in flow chart

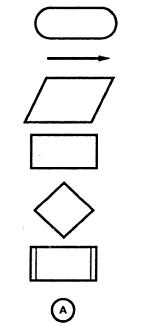


Fig 2.5 Graphical symbols used in flow chart

1. Oval: indicates start or stop operation.
2. Arrow: indicates flow with direction
3. Parallelogram: indicates input/output operation.
4. Rectangle: indicates process operation
5. Diamond: indicates decision making operation
6. Double sided rectangle: indicates execution of subroutine

7. Circle with alphabet: indicates continuation.

**2.6 DATA TRANSFER INSTRUCTIONS**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Opcode | Operand | Description |  |  |
| Copy from source to destination |  |  |  |  |
| MOV | Rd, Rs | This instruction copies the contents of the source | |  |
|  | M, Rs | register into the destination register; the contents | |  |
|  | Rd, M | of the source register are not altered. If one of the | |  |
|  |  | operands is a memory location, its location is | |  |
|  |  | specified by the contents of the HL registers. | |  |
| Move immediate 8-bit |  | Example: MOV B, C or MOV B, M | |  |
|  |  |  |  |
| MVI | Rd, data | The 8-bit data is stored in the destination register | |  |
|  | M, data | or memory. If the operand is a memory location, its | |  |
|  |  | location is specified by the contents of the HL | |  |
|  |  | registers. |  |  |
|  |  | Example: MVI B, 57 or MVI M, 57 | |  |
| Load accumulator |  |  |  |  |
| LDA | 16-bit | The contents of a memory location, specified by | |  |
|  | address | a16-bit address in the operand, are copied to the | |  |
|  |  | accumulator. The contents of the source are not | |  |
|  |  | altered. |  |  |
|  |  | Example: LDA 2034 or LDA XYZ | |  |
| Load accumulator indirect |  |  |  |  |
| LDAX | B/D Reg. | The contents of the designated register pair point to a | |  |
|  | pair | memory location. This instruction copies the | |  |
|  |  | contents of that memory location into the | |  |
|  |  | accumulator. The contents of either the register | |  |
|  |  | pair or the memory location are not altered. | |  |
| Store accumulator direct |  | Example: LDAX B |  |  |
|  |  |  |  |
| STA | 16-bit | The contents of the accumulator are copied into the | |  |
|  | address | memory location specified by | the operand. |  |
|  |  | This is a 3-byte instruction, | the second byte |  |
|  |  | specifies the low-order address and the third byte | |  |
|  |  | specifies the high-order address. |  |  |
| Store accumulator indirect |  | Example: STA 4350 or STA XYZ | |  |
|  |  |  |  |
| STAX | Reg. pair | The contents of the accumulator are copied into the | |  |
|  |  | memory location specified by the contents of the | |  |
|  |  | operand (register pair). The contents of the | |  |
|  |  | accumulator are not altered. |  |  |
|  |  | Example: STAX B |  |  |

Load register pair immediate

LXI Reg. pair, 16-bit data

Load H and L registers direct LHLD 16-bit address

Store H and L registers direct

SHLD 16-bit address

Exchange H and L with D and E XCHG none

The instruction loads 16-bit data in the register pair designated in the operand.

Example: LXI H, 2034

The instruction copies the contents of the memory location pointed out by the 16-bit address into register L and copies the contents of the next memory location into register H. The contents of source memory locations are not altered.

Example: LHLD 2040

The contents of register L are stored into the memory location specified by the 16-bit address in the operand and the contents of H register are stored into the next memory location by incrementing the operand. The contents of registers HL are not altered. This is a 3-byte instruction, the second byte specifies the low-order address and the third byte specifies the high-order address.

Example: SHLD 2470

The contents of register H are exchanged with the contents of register D, and the contents of register L are exchanged with the contents of register E.

Example: XCHG

**2.7 ARITHMETIC INSTRUCTIONS**

|  |  |  |  |
| --- | --- | --- | --- |
| Opcode | Operand | Description |  |
| Add register or memory to accumulator | | |  |
| ADD | R | The contents of the operand (register or memory) are added to |  |
|  | M | the contents of the accumulator and the result is stored in the |  |
|  |  |  |
|  |  | accumulator. If the operand is a memory location, its location |  |
|  |  | is specified by the contents of the HL registers. All flags are |  |
|  |  | modified to reflect the result of the addition. |  |
|  |  | Example: ADD B or ADD M |  |
| Add register to accumulator with carry | | |  |
| ADC | R | The contents of the operand (register or memory) and the |  |
|  | M | Carry flag are added to the contents of the accumulator and the |  |
|  |  | result is stored in the accumulator. If the operand is a memory |  |
|  |  | location, its location is specified by the contents of the HL |  |
|  |  | registers. All flags are modified to reflect the result of the |  |
|  |  | addition. |  |
|  |  | Example: ADC B or ADC M |  |
| Add immediate to accumulator | |  |  |
| ADI | 8-bit data | The 8-bit data (operand) is added to the contents of the |  |
|  |  | accumulator and the result is stored in the accumulator. All flags |  |
|  |  | are modified to reflect the result of the addition. |  |
|  |  | Example: ADI 45 |  |

Add immediate to accumulator with carry

ACI 8-bit data

Add register pair to H and L registers DAD Reg. pair

The 8-bit data (operand) and the Carry flag are added to the contents of the accumulator and the result is stored in the accumulator. All flags are modified to reflect the result of the addition.

Example: ACI 45

The 16-bit contents of the specified register pair are added to the contents of the HL register and the sum is stored in the HL register. The contents of the source register pair are not altered. If the result is larger than 16 bits, the CY flag is set. No other flags are affected.

Example: DAD H

SUB R The contents of the operand (register or memory ) are

1. subtracted from the contents of the accumulator, and the result is stored in the accumulator. If the operand is a memory location, its location is specified by the contents of the HL registers. All flags are modified to reflect the result of the subtraction.

|  |  |  |
| --- | --- | --- |
|  |  | Example: SUB B or SUB M |
| Subtract source and borrow from accumulator | | |
| SBB | R | The contents of the operand (register or memory ) and the |
|  | M | Borrow flag are subtracted from the contents of the |

accumulator and the result is placed in the accumulator. If the operand is a memory location, its location is specified by the contents of the HL registers. All flags are modified to reflect the result in accumulator.

Example: SBB B or SBB M

Subtract immediate from accumulator

SUI 8-bit data The 8-bit data (operand) is subtracted from the contents of the accumulator and the result is stored in the accumulator. All flags are modified to reflect the result of the subtraction.

Example: SUI 45

Subtract immediate from accumulator with borrow

SBI 8-bit data The 8-bit data (operand) and the Borrow flag are subtracted from the contents of the accumulator and the result is stored in the accumulator. All flags are modified to reflect the result of the subtracion.

Example: SBI 45

Increment register or memory by 1

INR R

M

The contents of the designated register or memory) are incremented by 1 and the result is stored in the same place. If the operand is a memory location, its location is specified by the contents of the HL registers.

Example: INR B or INR M

Increment register pair by 1 INX R

The contents of the designated register pair are incremented by 1 and the result is stored in the same place.

Example: INX H

Decrement register or memory by 1 DCR R

M

The contents of the designated register or memory are decremented by 1 and the result is stored in the same place. If the operand is a memory location, its location is specified by the contents of the HL registers.

Example: DCR B or DCR M

Decrement register pair by 1 DCX R

Decimal adjust accumulator DAA none

**2.8 LOGICALINSTRUCTIONS**

The contents of the designated register pair are decremented by 1 and the result is stored in the same place. Example: DCX H

The contents of the accumulator are changed from a binary value to two 4-bit binary coded decimal (BCD) digits. This is the only instruction that uses the auxiliary flag to perform the binary to BCD conversion, and the conversion procedure is described below. S, Z, AC, P, CY flags are altered to reflect the results of the operation.

If the value of the low-order 4-bits in the accumulator is greater than 9 or if AC flag is set, the instruction adds 6 to the low-order four bits.

If the value of the high-order 4-bits in the accumulator is greater than 9 or if the Carry flag is set, the instruction adds 6 to the high-order four bits.

Example: DAA

|  |  |  |  |
| --- | --- | --- | --- |
| Opcode | Operand | Description | |
| Compare register or memory with accumulator | | |  |
| CMP | R | The contents of the operand (register or memory) are | |
|  | M | compared | with the contents of the accumulator. Both |
|  |  | contents are preserved . The result of the comparison is | |

shown by setting the flags of the PSW as follows: if (A) < (reg/mem): carry flag is set, s=1

if (A) = (reg/mem): zero flag is set, s=0

if (A) > (reg/mem): carry and zero flags are reset, s=0 Example: CMP B or CMP M

Compare immediate with accumulator

CPI 8-bit data The second byte (8- bit data) is compared with the contents of the accumulator. The values being compared remain unchanged. The result of the comparison is shown by setting the flags of the PSW as follows:

if (A) < data: carry flag is set, s=1 if (A) = data: zero flag is set, s=0

if (A) > data: carry and zero flags are reset, s=0 Example: CPI 89

Logical AND register or memory with accumulator

ANA R The contents of the accumulator are logically ANDed with

M the contents of the operand (register or memory), and the

result is placed in the accumulator. If the operand is a

memory location, its address is specified by the contents of

HL registers. S, Z, P are modified to reflect the result of

the operation. CY is reset. AC is set.

Example: ANA B or ANA M

Logical AND immediate with accumulator

ANI 8-bit data The contents of the accumulator are logically ANDed with the 8-bit data (operand) and the result is placed in the accumulator. S, Z, P are modified to reflect the result of the operation. CY is reset. AC is set.

Example: ANI 86

Exclusive OR register or memory with accumulator

|  |  |  |
| --- | --- | --- |
| XRA | R | The contents of the accumulator are Exclusive ORed with |
|  | M | the contents of the operand (register or memory), and the |
|  |  | result is placed in the accumulator. If the operand is a |
|  |  | memory location, its address is specified by the contents of |
|  |  | HL registers. S, Z, P are modified to reflect the result of |
|  |  | the operation. CY and AC are reset. |
|  |  | Example: XRA B or XRA M |
| Logical OR register or memory with accumulator | | |
| ORA | R | The contents of the accumulator are logically ORed with |
|  | M | the contents of the operand (register/memory), and the |
|  |  | Result is placed in the accumulator. If the operand is a |
|  |  | memory location, its address is specified by the contents of |
|  |  | HL registers. S, Z, P are modified to reflect the result of the |
|  |  | operation. CY and AC are reset. |
|  |  | Example: ORA B or ORA M |

Exclusive OR immediate with accumulator

XRI 8-bit data The contents of the accumulator are Exclusive ORed with the 8-bit data (operand) and the result is placed in the accumulator. S, Z, P are modified to reflect the result of the operation. CY and AC are reset.

Example: XRI 86

Logical OR immediate with accumulator

ORI 8-bit data The contents of the accumulator are logically ORed with the 8-bit data (operand) and the result is placed in the accumulator. S, Z, P are modified to reflect the result of the operation. CY and AC are reset.

Example: ORI 86

Complement accumulator

CMA none The contents of the accumulator are complemented. No flags are affected.Example: CMA

|  |  |  |  |
| --- | --- | --- | --- |
| Complement carry | |  |  |
| CMC none |  | The Carry flag is complemented. No other flags are affected. |  |
| Set Carry |  | Example: CMC |  |
|  |  |  |
| STC none |  | The Carry flag is set to 1. No other flags are affected. |  |
|  |  | Example: STC |  |
| **2.9 BRANCHING INSTRUCTIONS** | | |  |
| Opcode | Operand | Description |  |
| Jump unconditionally | |  |  |
| JMP | 16-bit address | The program sequence is transferred to the memory location |  |
|  |  | specified by the 16-bit address given in the operand. |  |
| Jump conditionally | | Example: JMP 2034 or JMP XYZ |  |
|  |  |
| Operand: 16-bit address | |  |  |
|  |  | The program sequence is transferred to the memory location |  |
|  |  | specified by the 16- bit address given in the operand based on |  |
|  |  | the specified flag of the PSW as described below. |  |
|  |  | Example: JZ 2034 or JZ XYZ |  |

|  |  |  |
| --- | --- | --- |
| Opcode | Description | Flag Status |
| JC | Jump on Carry | CY = 1 |
| JNC | Jump on no Carry | CY = 0 |
| JP | Jump on positive | S = 0 |
| JM | Jump on minus | S = 1 |
| JZ | Jump on zero | Z = 1 |
| JNZ | Jump on no zero | Z = 0 |
| JPE | Jump on parity even | P = 1 |
| JPO | Jump on parity odd | P = 0 |

Unconditional subroutine call

CALL 16-bit address The program sequence is transferred to the memory location

specified by the 16-bit address given in the operand. Before

the transfer, the address of the next instruction after CALL (the contents of the program counter) is pushed onto the stack. Example: CALL 2034 or CALL XYZ

Call conditionally Operand: 16-bit address

The program sequence is transferred to the memory location specified by the 16-bit address given in the operand based on the specified flag of the PSW as described below. Before the transfer, the address of the next instruction after the call (the contents of the program counter) is pushed onto the stack.

Example: CZ 2034 or CZ XYZ

|  |  |  |
| --- | --- | --- |
| Opcode | Description | Flag Status |
| CC | Call on Carry | CY = 1 |
| CNC | Call on no Carry | CY = 0 |
| CP | Call on positive | S = 0 |
| CM | Call on minus | S = 1 |
| CZ | Call on zero | Z = 1 |
| CNZ | Call on no zero | Z = 0 |
| CPE | Call on parity even | P = 1 |
| CPO | Call on parity odd | P = 0 |

Return from subroutine unconditionally

RET none The program sequence is transferred from the subroutine to the calling program. The two bytes from the top of the stack are copied into the program counter, and program execution begins at the new address.

Example: RET

Return from subroutine conditionally

Operand: none

The program sequence is transferred from the subroutine to the calling program based on the specified flag of the PSW as described below. The two bytes from the top of the stack are copied into the program counter, and program execution begins at the new address.

Example: RZ

|  |  |  |
| --- | --- | --- |
| Opcode | Description | Flag Status |
| RC | Return on Carry | CY = 1 |
| RNC | Return on no Carry | CY = 0 |
| RP | Return on positive | S = 0 |
| RM | Return on minus | S = 1 |
| RZ | Return on zero | Z = 1 |
| RNZ | Return on no zero | Z = 0 |
| RPE | Return on parity even | P = 1 |
| RPO | Return on parity odd | P = 0 |

Load program counter with HL contents

PCHL none The contents of registers H and L are copied into the program counter. The contents of H are placed as the high-order byte and the contents of L as the low-order byte.

Example: PCHL

Restart

RST 0-7 The RST instruction is equivalent to a 1-byte call instruction to one of eight memory locations depending upon the number. The instructions are generally used in conjunction with interrupts and inserted using external hardware. However these can be used as software instructions in a program to

Transfer program execution to one of the eight locations. The addresses are:

|  |  |
| --- | --- |
| Instruction | Restart Address |
| RST 0 | 0000H |
| RST 1 | 0008H |
| RST 2 | 0010H |
| RST 3 | 0018H |
| RST 4 | 0020H |
| RST 5 | 0028H |
| RST 6 | 0030H |
| RST 7 | 0038H |

The 8085 has four additional interrupts and these interrupts generate RST instructions internally and thus do not require any external hardware. These instructions and their Restart addresses are:

|  |  |
| --- | --- |
| Interrupt | Restart Address |
| TRAP | 0024H |
| RST 5.5 | 002CH |
| RST 6.5 | 0034H |
| RST 7.5 | 003CH |

**2.10 WRITING ASSEMBLY LANGUAGE PROGRAMMS**

1. Store the data byte 32H into memory location 4000H. MVI A, 52H : Store 32H in the accumulator

STA 4000H : Copy accumulator contents at address 4000H

HLT : Terminate program execution

Program 2:

LXI H : Load HL with 4000H

MVI M : Store 32H in memory location pointed by HL register pair

HLT : Terminate program execution

2. Exchange the contents of memory locations 2000H and 4000H.

Program 1:

LDA 2000H : Get the contents of memory location 2000H into accumulator

MOV B, A : Save the contents into B register

LDA 4000H : Get the contents of memory location 4000Hinto accumulator

STA 2000H : Store the contents of accumulator at address 2000H

MOV A, B : Get the saved contents back into A register

STA 4000H : Store the contents of accumulator at address 4000H

Program 2:

LXI H 2000H : Initialize HL register pair as a pointer to memory location 2000H.

LXI D 4000H : Initialize DE register pair as a pointer to memory location 4000H.

MOV B, M : Get the contents of memory location 2000H into B register.

LDAX D : Get the contents of memory location 4000H into A register.

MOV M, A : Store the contents of A register into memory location 2000H.

MOV A, B : Copy the contents of B register into accumulator.

STAX D : Store the contents of A register into memory location 4000H.

HLT : Terminate program execution.

3.Find the 2's complement of the number stored at memory location 4200H and store the complemented number at memory location 4300H.

*Source program:*

*LDA 4200H* : Get the number

CMA : Complement the number

ADI, 01 H : Add one in the number

STA 4300H : Store the result

HLT : Terminate program execution

Flow chart:

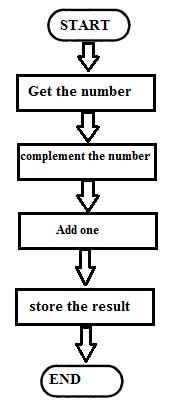


Fig 2.6 Flow Chart

4. Add the contents of memory locations 4000H and 4001H and place the result in memory location 4002H.

*Sample problem (4000H) = 14H (4001H) = 89H*

*Result* *= 14H + 89H = 9DH*

Source program

LXI H 4000H : HL points 4000H

MOV A, M : Get first operand

INX H : HL points 4001H

ADD M : Add second operand

INX H : HL points 4002H

MOV M, A : Store result at 4002H

HLT : Terminate program execution

Flowchart

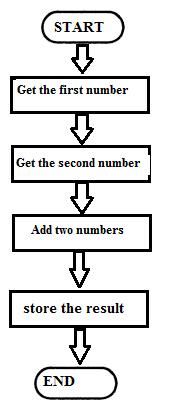


Fig 2.7 Flow Chart

5. Subtract the contents of memory location 4001H from the memory location 2000H and place the result in memory location 4002H.

Sample problem: (4000H) = 51H (4001H) = 19H

Result = 51H – 19H = 38H Flowchart

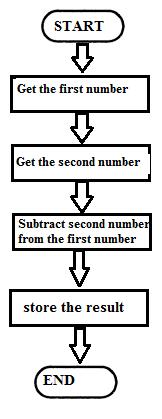


Fig 2.8 Flow Chart

Source program:

LXI H, 4000H : HL points 4000H

MOV A, M : Get first operand

INX H : HL points 4001H

SUB M : Subtract second operand

INX H : HL points 4002H

MOV M, A : Store result at 4002H.

HLT : Terminate program execution

6. Pack the two unpacked BCD numbers stored in memory locations 4200H and 4201H and store result in memory location 4300H. Assume the least significant digit is stored at 4200H.

Sample problem: (4200H) = 04 (4201H) = 09

Result = (4300H) = 94

Flow chart:

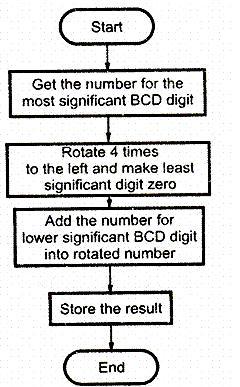


Fig 2.9 Flow Chart

Source program

LDA 4201H : Get the Most significant BCD digit

RLC

RLC

RLC

RLC : Adjust the position of the second digit (09 is changed to 90)

ANI FOH : Make least significant BCD digit zero

MOV C, A : store the partial result

LDA 4200H : Get the lower BCD digit

ADD C : Add lower BCD digit

STA 4300H : Store the result

HLT : Terminate program execution

NOTE:

BCD NO.: The numbers "0 to 9" are called BCD (Binary Coded Decimal) numbers. A decimal number 29 can be converted into BCD number by splitting it into two. ie. 02 and 09.

7. Two digit BCD number is stored in memory location 4200H. Unpack the BCD number and store the two digits in memory locations 4300H and 4301H such that memory location 4300H will have lower BCD digit.

Sample problem: (4200H) = 58

Result = (4300H) = 08 and (4301H) = 05

Flowchart

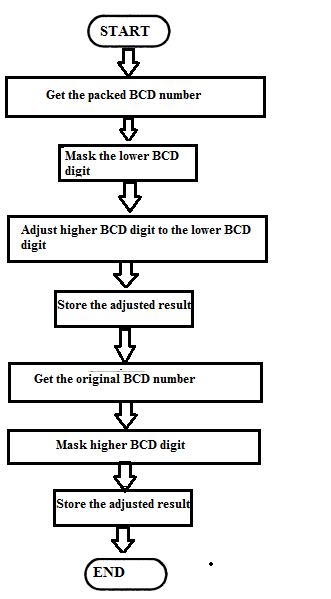


Fig 2.10 Flow Chart

Source program:

LDA 4200H : Get the packed BCD number

ANI FOH : Mask lower nibble

RRC

RRC

RRC

RRC : Adjust higher BCD digit as a lower digit

STA 4301H : Store the partial result

LDA 4200H : .Get the original BCD number

ANI OFH : Mask higher nibble

STA 4201H : Store the result

HLT : Terminate program execution

**ADDRESSING MODES**

Every instruction of a program has to operate on a data. The method of specifying the data to be operated by the instruction is called *Addressing.*

The 8085 has the following 5 different types of addressing.

1. Immediate Addressing
2. Direct Addressing
3. Register Addressing
4. Register Indirect Addressing
5. Implied Addressing

**Immediate Addressing**

In immediate addressing mode, the data is specified in the instruction itself. The data will be apart of the program instruction. All instructions that have ‘I’ in their mnemonics are of Immediate addressing type.

*Eg.*MVI B, 3EH- Move the data 3EH given in the instruction to B register.

**Direct Addressing**

In direct addressing mode, the address of the data is specified in the instruction. The data will be in memory. In this addressing mode, the program instructions and data can be stored in differentmemory blocks. This type of addressing can be identified by 16-

bit address present in theinstruction.

*Eg.*LDA 1050H- Load the data available in memory location 1050H in accumulator.

**Register Addressing**

In register addressing mode, the instruction specifies the name of the register in which the data is available. This type of addressing can be identified by register names (such as ‘A’, ‘B’....) in the instruction.

*Eg.* MOV A, B -Move the content of B register to A register.

**Register Indirect Addressing**

In register indirect addressing mode, 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 the register pair. This type of addressing can be identified by letter ‘M’ present in the instruction.

*Eg.* MOV A, M - The memory data addressed by HL pair is moved to A register.

**Implied Addressing**

In implied addressing mode, the instruction itself specifies the type of operation and location of data to be operated. This type of instruction does not have any address, register name, immediate data specified along with it. *Eg.* CMA - Complement the content of accumulator

**2.11 PROGRAMMING TECHNIQUES**

**Looping**-In this technique, the program is instructed to execute certain set of instructions repeatedly toexecute a particular task number of times.

**Counting**-This technique allows programmer to count how many times the instruction/set of instructionsare executed.

**Indexing**-This technique allows programmer to point or refer the data stored in sequential memorylocation one by one.