1. Introduction

Block diagram of microcomputer System :-

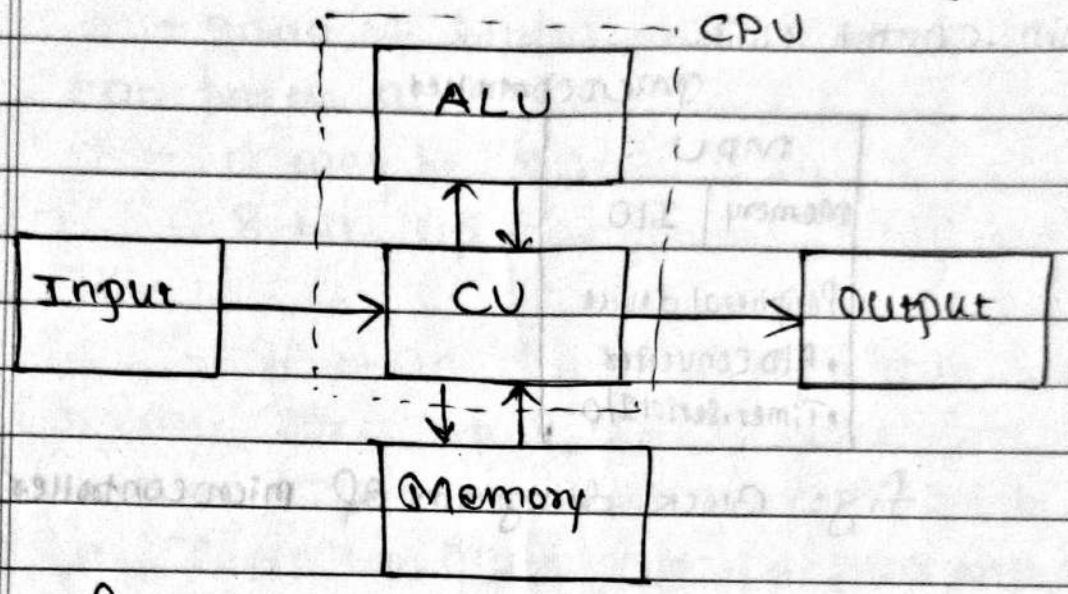


fig: Traditional block diagram of a computer

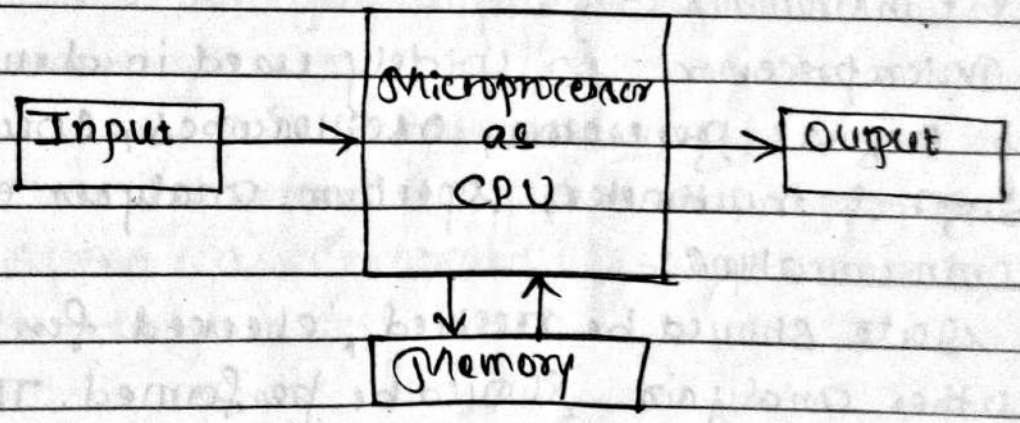


fig: Block diagram of computer with microprocessor as CPU

Microcomputer System consist of microprocessor, main memory, input & output devices & Secondary Storage devices. A microprocessor is a CPU which is constructed on a single Silicon chip. CPU consist of ALU & control unit. ALU is capable of processing arithmetic & logical operations on binary data. Control unit interprets the instruction in memory & causes

Signature

them to be executed. I/O unit gets operated from control unit.

Microcontroller

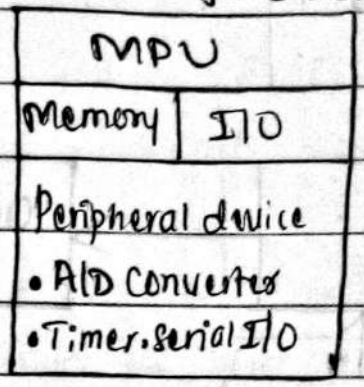


fig: Block diagram of microcontroller

Application of microprocessor :-

1) Test instruments :-

- Microprocessor is widely used in devices such as signal generator, oscilloscopes, counter, digital multimeter, spectrum analyzer etc

2) Communication :-

Data should be received, checked for errors & further analysis should be performed. The speed at which the microprocessor can take decision & compute errors is truly substantial.

3) Computer :-

- Microprocessor is CPU of microcomputer. It can perform arithmetic & logic functions as well as control functions. CU of microprocessor sends signals to input, output units, memory ALU & arrange the sequence of their controlling operation.

4) Industries: It is widely used in data monitoring, weighing system, smart cameras for quality control, automatic

Wordlength of microprocessor:-

- group of bits of data that a microprocessor can process at a time.
- n may be 8, 16, 32 or 64.
- 8-bit µp can process 8-bit data at a time.
- If data consist of more than 8 bits, the processor takes up 8 bits of data first & makes it processing, then the next group of 8 bits of data are taken up one by one for processing.
- In ALU is designed to process 8-bit data. In general purpose registers hold data for processing are 8-bit registers.
- A processor of longer wordlength is more powerful & can process data at faster speed as compared to a processor of shorter wordlength.

Evolution of microprocessor:-

The first microprocessor, Intel 4004, 4-bit PMOS microprocessor was introduced in the year 1971 by Intel Corporation, USA. After this 4-bit microprocessor Intel 4040, an enhanced version of Intel 4004 was developed.

In 1972, Intel introduced the first 8-bit microprocessor, Intel 8008 which also uses PMOS technology. The microprocessor using PMOS technology

were slow & not compatible with TTL logic.

In 1973, Intel introduced more powerful & faster 8-bit NMOS microprocessor, Intel 8080. The microprocessors using NMOS process technology are faster & compatible with TTL logic. The drawback of 8080 was that it needed three power supplies.

In 1975, Intel developed an improved 8-bit NMOS microprocessor, Intel 8085 which uses only one +5V supply. It is an improved version of Intel 8080. It is still used in laboratory for students training. In the first course of microprocessor, the theory based on Intel 8085 is taught as it gives the basic concept of microprocessor, which are needed for further study of other courses on microprocessors, microcontrollers & computers.

Generally the memory addressing capacity of 8-bit microprocessors are 64 KB. The clock freq for the most of 8-bit microprocessors lies in the range of 1 MHz to 6 MHz.

Some other examples of 8-bit microprocessors are Motorola 6500, Texas Instrument-9900, Toshiba TLCS-12 etc.

In the year 1978, Intel introduced 16-bit microprocessor, Intel 8086. Other 16-bit

microprocessors are Intel 80186, Intel 8088, Intel 80188, Intel 80286 etc.

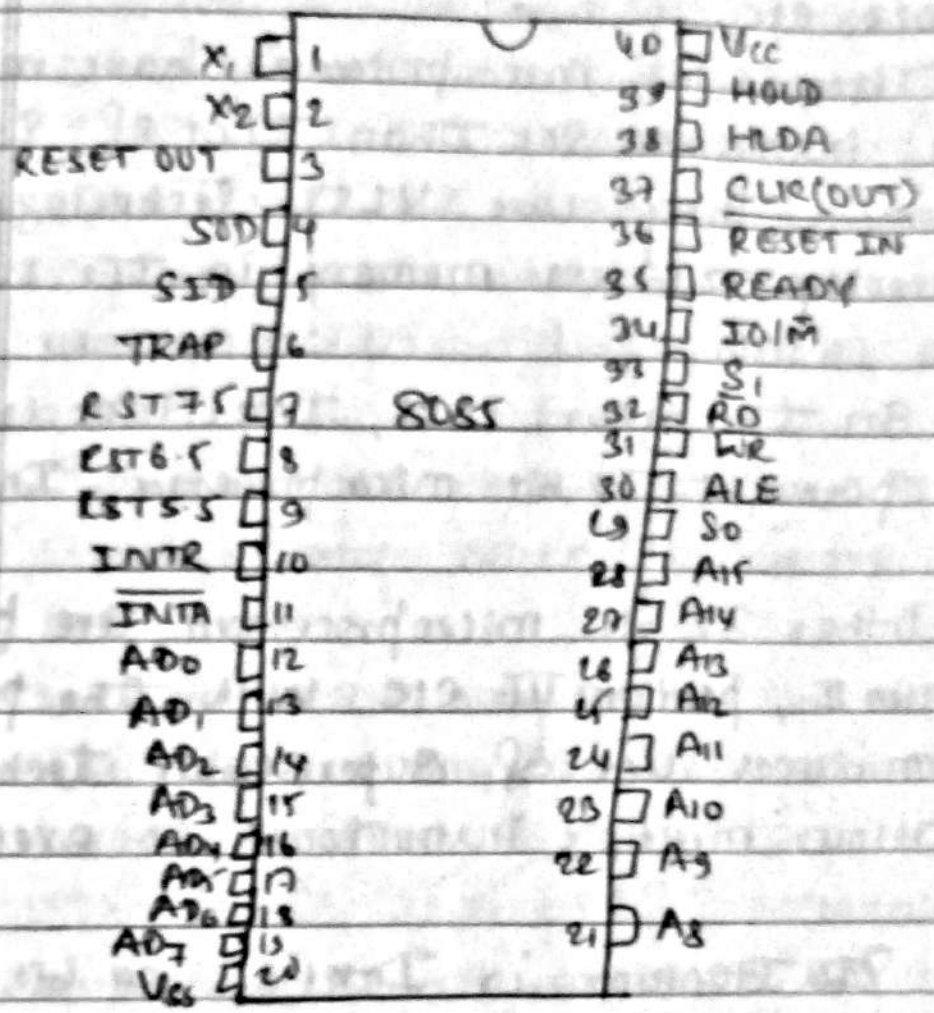
The 16-bit microprocessors have more powerful instruction set than that of 8-bit microprocessors. They use VLSI technology. They can directly address memory in the range of 1MB to 16MB.

In the year 1985, Intel introduced a more powerful 32-bit microprocessor, Intel 80386.

Other 32-bit microprocessors are pentium pro, pentium II, pentium III etc. With the pentium, Intel introduces use of superscalar techniques which allow multiple instructions to execute in parallel.

Itanium is Intel's 64-bit processor.

Pin diagram of 8085 microprocessor:-



8085 pinout

- We define µp as a device that can communicate with peripherals, provide timing signals, direct data flow & perform computing tasks as specified by the instructions in memory.
- 8085 is 8-bit general purpose microprocessor capable of addressing 64 KB of memory.
- The device has 40 pins requires +5V single power supply & can operate with 3 MHz single-phase clock.
- The description of various pins as follows:-

$A_8 - A_{15}$ (Output):-

These are address bus & are used for the most significant bit of memory address.

$AD_0 - AD_7$ (Input/Output):-

These are ~~address~~ time multiplexed address data bus i.e. They serve dual purpose. They are used for least significant 8-bits of memory address or I/O address during the first clock of a machine cycle. Again they are used for data during second & third clock cycles.

ALE (Output):-

It is an address latch enable signal. It goes high during first clock of a machine cycle.

$IO/\overline{M}$ (Output):-

It is a status signal which distinguishes whether the address is for memory or I/O. When it goes high, the address on the address bus is for I/O device. When it goes low, the address on the address bus is for memory location.

S_0, S_1 (Output):-

These are ~~the~~ status signals sent by the microprocessor to distinguish the various types of operation given below:-

Status code for Intel 8085

S_1	S_0	Operation
0	0	HALT
0	1	WRITE
1	0	READ
1	1	FETCH

$\overline{RD}$ (Output):-

It is a signal to control READ operation. When it goes low, the selected memory or I/O device is read.

$\overline{WR}$ (Output):-

It is a signal to control WRITE operation. When it goes low, the data on the data bus is written into the selected memory or I/O location.

READY (input):-

It is used by the microprocessor to sense whether a peripheral is ready to transfer data or not. A slow peripheral may be connected to the microprocessor through READY line. If READY is high, the peripheral is ready. If it is low, the microprocessor waits till it goes high.

HOLD (input):-

It indicates that another device is requesting for the use of address & data bus. Having received a HOLD request, the microprocessor relinquishes the use of the buses as soon as the current machine cycle is completed.

HLDA (Output):-

It is a signal for HOLD acknowledgement. It indicates that HOLD request has been received. After removal of HOLD request, HLDA goes low.

INTR (input):-

It is an interrupt request signal. Among interrupts, it has the lowest priority. When it goes high, the program counter doesn't increment its content. The microprocessor suspends its normal sequence of instructions. An interrupt is used by I/O devices to transfer data to the microprocessor without wasting its time.

$\overline{\text{INTA}}$ (Output):-

It is an interrupt acknowledgement sent by the microprocessor after INTR is received.

RST 7.5, 6.5, 5.5 & TRAP (inputs):-

These are interrupts. When an interrupt is recognised, the next instruction is executed from a fixed location in the memory as given below:-

Line	Location from which next instruction is picked up
TRAP	0024
RST 5.5	002C
RST 6.5	0034
RST 7.5	003C

RST 7.5, 6.5, 5.5 are the restart interrupts. TRAP has the highest priority among interrupts. It is non-maskable interrupt. The order of priority

of interrupts is as follows:-

TRAP (highest priority)

RST7.5

RST6.5

RST5.5

INTR (lowest priority)

RESET IN (input):-

It resets the program counter to zero. The microprocessor is held in reset condition as long as RESET is applied.

RESET OUT (output):-

It indicates that microprocessor is being reset. The signal can be used to reset other devices.

X_1, X_2 (input):-

These are terminals to be connected to an external crystal oscillator which drives an internal circuitry of the microprocessor to produce a suitable clock for the operation of microprocessor.

CLK (output):-

It is a clock output for user which can be used for other digital ICs. Its frequency is same at which processor operates.

SID (input):-

It is a data line for serial input.

SOD (output):-

It is a data line for serial output.

Vcc: +5V supply

Vss: ground reference

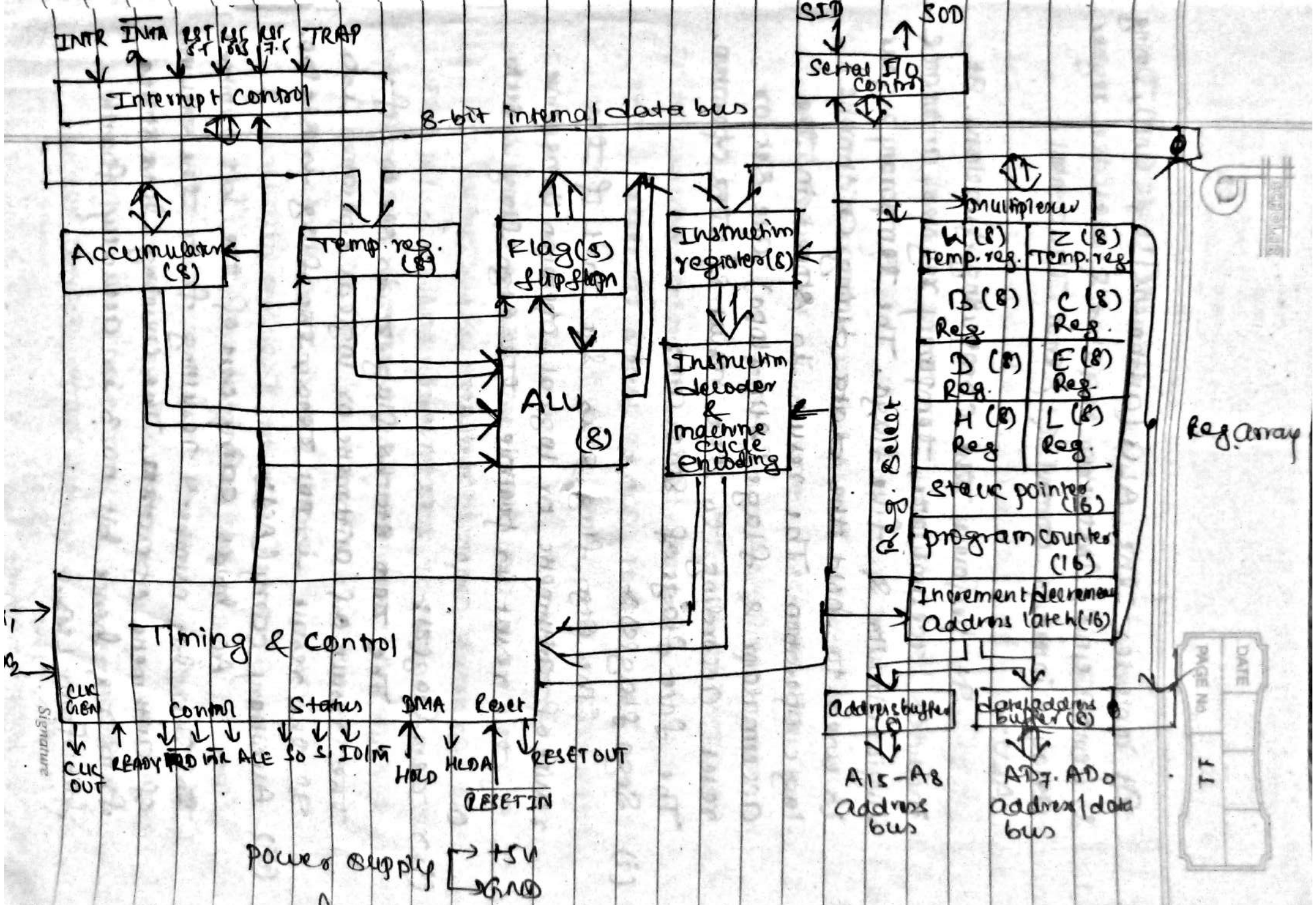


fig: Internal architecture of 8085 microprocessor

It includes the ALU (arithmetic/logic unit), timing & control unit, instruction registers & decoder, register array, interrupt control & serial I/O control.

ALU:-

It performs the computing functions. It includes accumulator, temporary registers, arithmetic logic circuits & five flags. The temporary register is used to hold the data during arithmetic/logic operation. The result is stored in the accumulator & flags (flip-flops) are set or reset according to the result of the operation. The five flags of 8085 are:-

(i) Sign flag(S):-

The sign flag S is set to 1 if the result of arithmetic or logical operation is negative. If the result is positive, the sign flag is set to 0.

(ii) Zero flag(Z):-

The zero status flag Z is set to 1 if the result of arithmetic or logical operation is 0. If the result is not zero, the flag is set to 0.

(iii) Auxilliary carry (AC):-

AC holds carry out of the bit number 3 to the bit number 4 resulting from the execution of arithmetic operations. The counting bits starts from 0 & hence bit no. 3 is actually fourth bit from LSB.

(iv) Parity flag(P) :-

The parity status flag is set to 1, if the result of arithmetic or logical operation contains even number of 1s. It is reset if the result contains odd number of 1s.

(v) Carry flag(CY) :-

After the execution of arithmetic instructions if a carry is produced, the carry flag CY is set to 1, otherwise it is 0. The carry flag is set or reset in case of addition as well as subtraction. After the addition of two 8-bit numbers, if the sum is larger than 8-bits, a carry is produced & carry flag is set to 1. In case of subtraction, if borrow occurs, the carry flag is set to 1. The carry flag holds carry out of the most significant bit resulting from the execution of an arithmetic operation.

The bit position reserved for these flags in the flag register are as follows:-

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
S	Z		AC		P		CY

PSW(program status word) ⇒ contains 5 flags & 3 undefined bits D_1, D_3, D_5.

Timing & control unit :-

It generates timing & control signals which are necessary for the execution of instructions. It controls data flow between microprocessor &

peripherals (including memory). It provides status, control & timing signals which are required for the operation of memory & I/O devices.

Instruction register & decoder:-

The instruction register & decoder are part of ALU. When an instruction is fetched from memory, it is loaded in the instruction register. The decoder decodes the instruction & establishes the sequence of events to follow. The instruction register is not programmable & can't be accessed through any instruction.

Register array:-

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

These registers are programmable meaning that a programmer can use them to load or copy data from the registers by using instructions.

For example, instruction MOV B, C copies data from register C to register B.

Two additional registers called temporary registers W & Z are included in the register array. They are used to hold 8-bit data during execution of some instructions. They are not available to the programmer.

Program counter:-

It is 16-bit register used to hold the memory address of next instruction to be executed. The microprocessor increments the contents of the program counter during the execution of an instruction so that it points to the address of the next instruction in the program at the end of execution of an instruction.

Stack pointer:-

It is 16-bit special function register. The stack is a sequence of memory locations set aside by a programmer to store/retrieve the contents of accumulator, flags, program counter & general purpose registers during the execution of a program. Any portion of memory can be used as stack. During the execution of a program sometimes it becomes necessary to save the contents of some registers which are needed for some other operations in the subsequent steps of the program.

If 8085 adds 87H & 79H, specify the contents of accumulator & status of S, Z & CY flags.

87H
79H
100H

contents of acc = 00H

S = 0, Z = 1, CY = 1, AC = 1, P = 0

Instruction cycle:-

An instruction is a command given to the computer to perform a specified operation on given data. To perform a particular task, a programmer writes a sequence of instructions called a program. Program & data are stored in the memory. The microprocessor fetches one instruction from the memory at a time & executes it. It executes all the instructions of a program one by one to provide to the final result.

The necessary steps that a microprocessor carries out to fetch an instruction & necessary data from the memory & to execute it, constitute an instruction cycle. An instruction cycle consist of fetch cycle & execute cycle. In fetch cycle, microprocessor fetches opcode from the memory. The necessary steps which are carried out to fetch an opcode from the memory constitute a fetch cycle. The necessary steps which are carried out to get data if any, from the memory & to perform the specific operation specified in an instruction constitute execute cycle. The time required to fetch an opcode (FC) is fixed slot of time while the time required to execute an instruction (EC) is variable which depends on the type of instruction to be executed. The total time required to

execute an instruction is given by

$$IC = FC + EC$$

Fetch Operation:-

1st byte of an instruction is its opcode. An instruction may be more than one byte long. The other bytes are data or operand address. The program counter keeps the memory address of the next instruction to be executed. In the beginning of fetch cycle, the content of PC, which is the address of memory location where opcode is available is sent to the memory. The memory places the opcode on the data bus so as to transfer it to the microprocessor. The entire ~~process~~ operation of fetching an opcode takes three clock cycles.

Execute Operation:-

The opcode fetched from the memory goes to the data register & then to instruction register. From IR, it goes to the decoder circuitry which decodes the instruction. After instruction is decoded, execution begins. If the operand is in the general purpose registers, execution is immediately performed. The time taken in decoding & execution is one clock cycles. If an instruction contains data or operand address which are still in the memory, the

microprocessor has to perform some read operations to get the desired data. After receiving the data, it performs execute operation. In some cases, an instruction cycle may involve one or more read or write cycles or both. Fig shows an instruction & fetch cycle respectively.

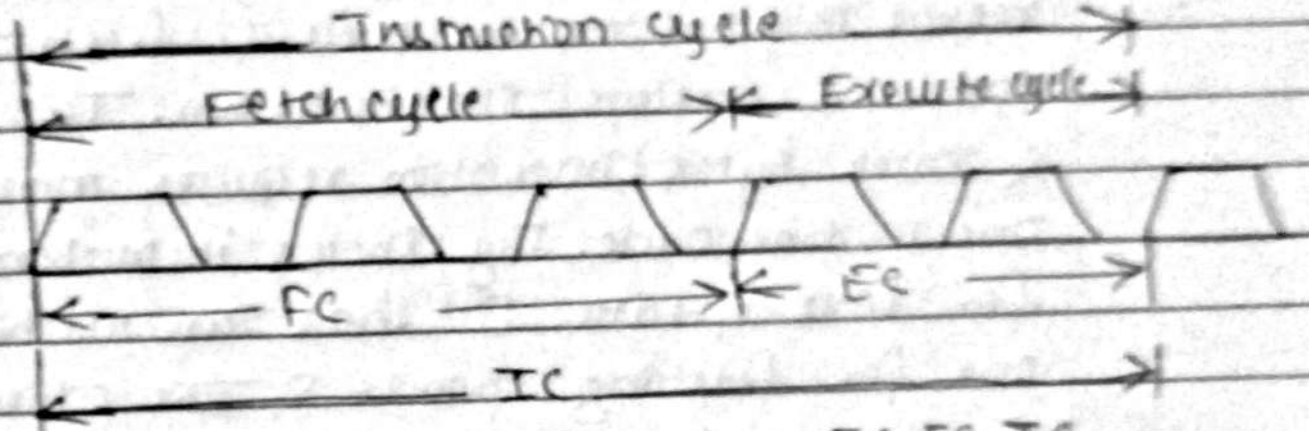


fig: Instruction cycle showing FC, EC, IC

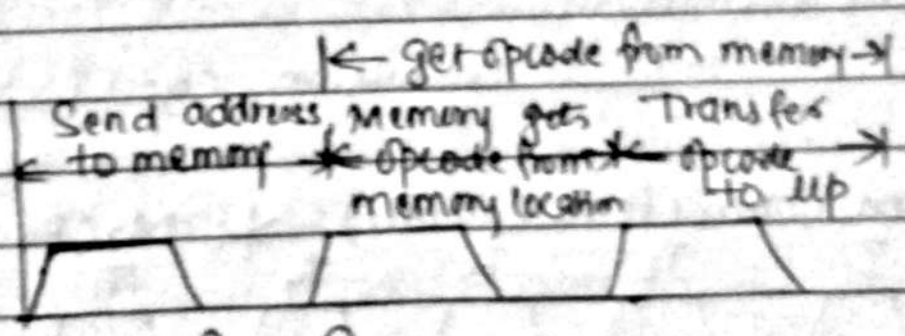


fig: fetch cycle

Machine cycle & State :-

The necessary steps carried out to perform the operation of accessing either memory or I/O device, constitute a machine cycle. In other words, necessary steps carried out to perform a fetch, a read, or a write operation constitute a machine cycle

In machine cycle, one basic operation such as opcode fetch, memory read, memory write, I/O read or I/O write is performed. An instruction cycle consist of several machine cycles. The opcode of an instruction is fetched in the first machine cycle of an instruction cycle. Most of the single byte instructions require only one machine cycle to fetch the opcode & execute the instruction. Two bytes & three bytes instruction requires more than one machine cycle. Fig shows instruction cycle for MVI r, data. It has two machine cycles: one for fetching opcode & the other for reading data from the memory & to execute the instruction.

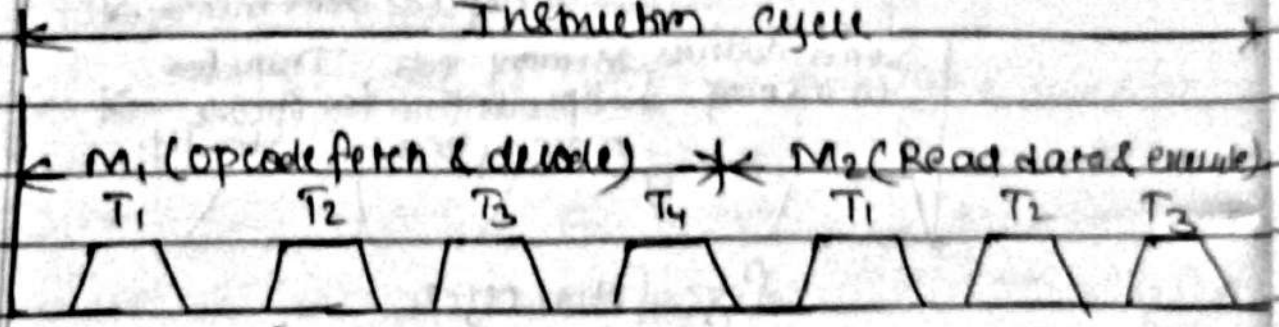


fig: Typical instruction cycle

One subdivision of an operation performed in one clock cycle is called state or T-state. The subdivisions are internal states synchronized with the system clock. So, one clock cycle of the system clock is referred to as a state.

Fetch-execution overlap:-

Instruction processing can be divided into two stages: fetch instruction & execute instruction. There are times during the execution of an instruction when main memory is not being accessed. This time could be used to fetch the next instruction in parallel with the execution of the current one. Fig depicts this approach. This approach is known as instruction pipeline.

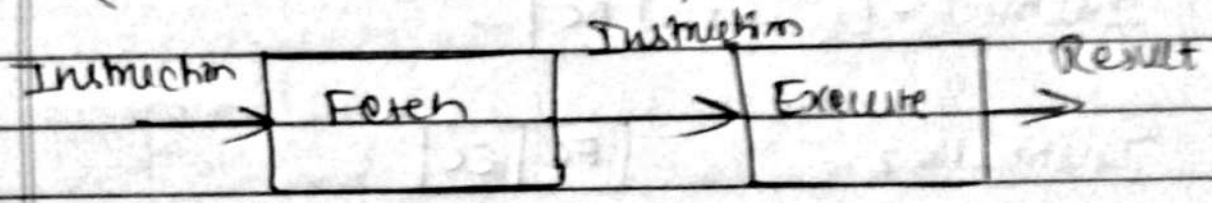


fig: Two stage instruction pipeline

The pipeline has two independent stages. The first stage fetches an instruction & buffers it. When the second stage is free, the first stage passes it the buffered instruction. While second stage is executing the instruction, the first stage takes advantage of any unused memory cycles to fetch & buffer the next instruction. This is called fetch overlap or fetch-execution overlap or instruction prefetch.

This process will speed up instruction execution. If the fetch & execute stages were of equal duration, the instruction cycle time would be halved.

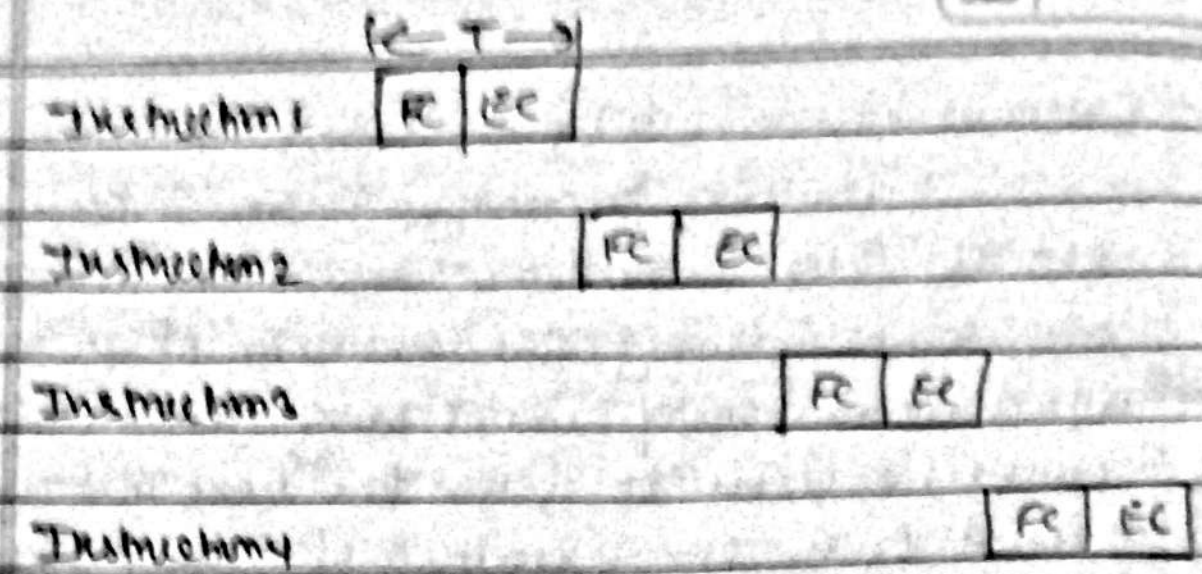


fig: Sequential execution

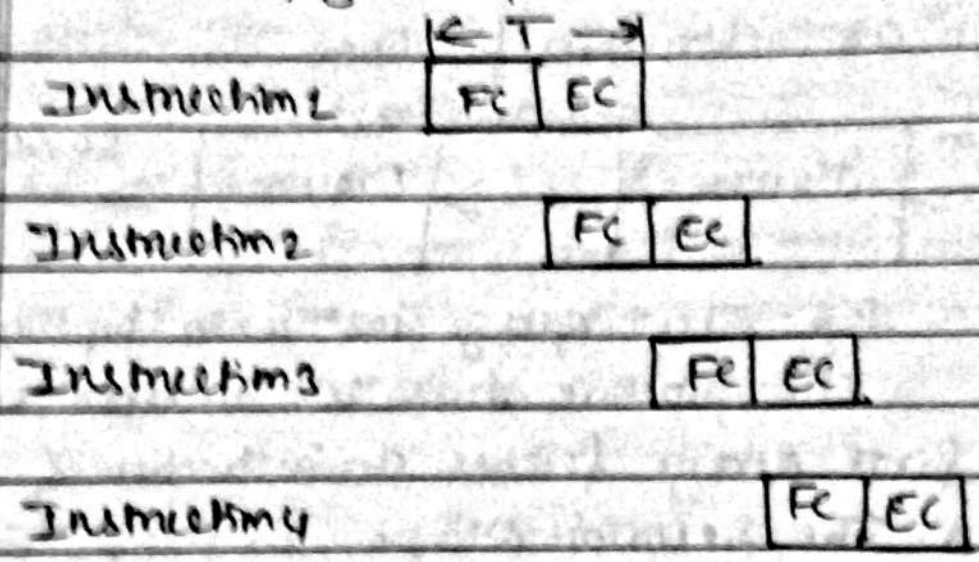


fig: Pipeline execution

Timing diagram:-

The necessary steps which are carried out in a machine cycle can be represented graphically. Such a graphical representation is called timing diagram.

The timing diagram of opcode fetch, memory read, memory write, I/O read & I/O write is given below:

Timing diagram for opcode fetch cycle:-

In a fetch cycle, the microprocessor fetches the opcode of an instruction from the memory.

The opcode of the instruction MOV C,A is 4FH, which is stored in the memory location 2005H. It is a 1-byte instruction & consist of only one machine cycle of opcode fetch only.

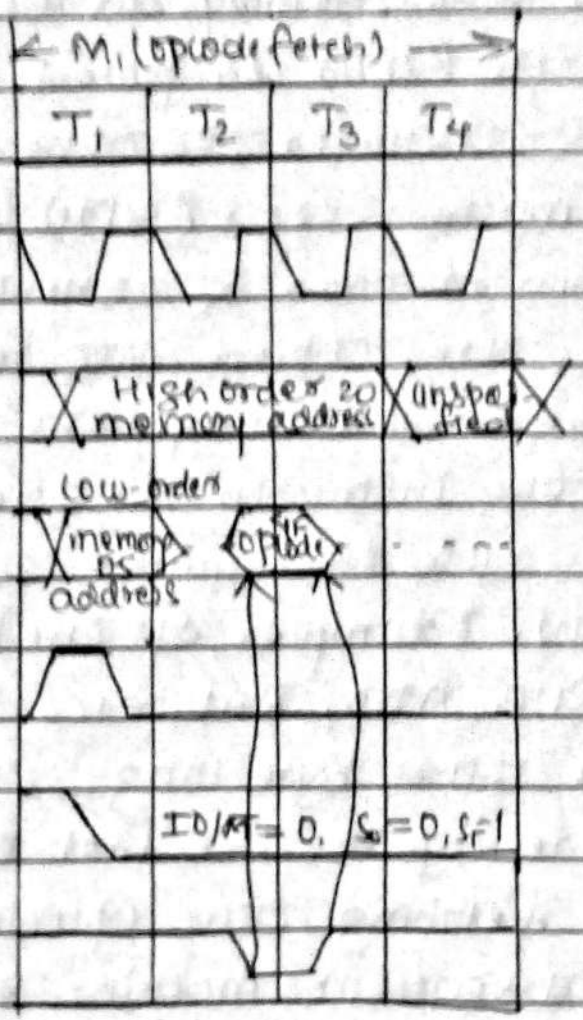


fig: Timing diagram of MOV C,A

Fig shows that at T_1, the higher-order memory address 20H is placed on the address lines A_{15}-A_8, the low-order memory address 05H

is placed on the data bus AD_7-AD_0 & ALE signal goes high. Similarly, the status signal $IO/\overline{M}$ goes low, indicating that this is a memory related operation.

The control signal $\overline{RD}$ is sent out during the clock period T_2, thus enabling the memory chip.

When the memory is enabled, the instruction byte (4FH) is placed on the bus AD_7-AD_0 & transfer to the microprocessor.

The machine code (4FH) is decoded by the instruction decoder & contents of the accumulator are copied into register C. This task is performed during the period T_4.

If the instruction is one byte long, only one machine cycle is required to fetch & execute the instruction. Examples of one byte long instructions are MOV, SUB, ADD, RAL etc. If an instruction is two or three byte long, it requires more machine cycles. The first machine cycle M_1 is for fetching the opcode from the memory. Subsequent machine cycles M_2, M_3 etc are required either to read data or address from the memory or I/O devices or to write data into the memory or I/O devices.

Timing diagram for memory read:-

In a memory read cycle, the microprocessor reads the content of memory location. The content is then placed either in the accumulator or any other register of microprocessor.

Now let us take an example of two byte long instruction,

MVI A,05H

3E,05H in the code form where 3E is the opcode for MVI A instruction & 05H is data

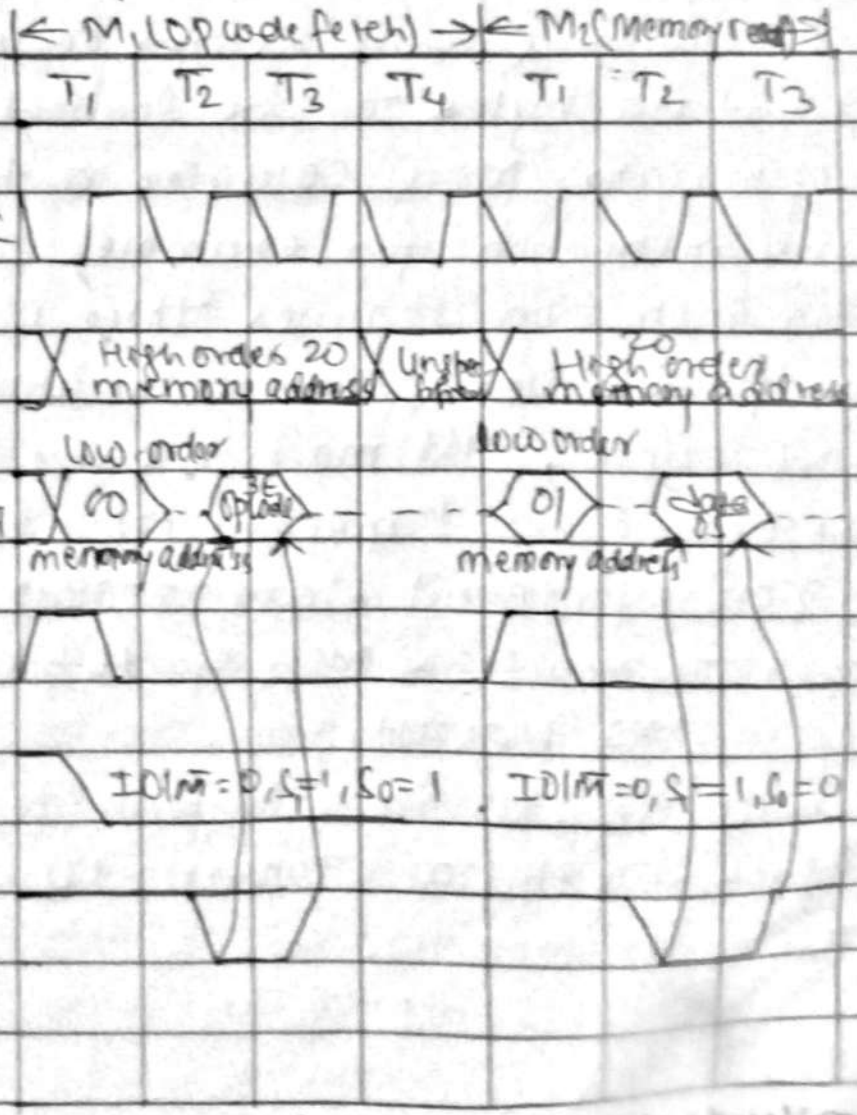


fig: Timing diagram for instruction MVI A,05H

This instruction requires two machine cycles M_1 & M_2. The first machine cycle M_1 is to fetch the opcode 3E from the memory & second machine cycle M_2 is for reading the data (05H) from the memory as shown in fig.

Two machine codes 3EH & 05H are stored in memory locations 2000H & 2001H respectively. The first machine code 3EH represents the opcode to load a data byte in the accumulator & the second code 05H represents the data bytes to be loaded in the accumulator. Now consider a three byte long instruction for example, LXI H, 2500H. This instruction requires three machine cycles one fetch cycle & two consecutive memory read cycles, 1st machine cycle is to fetch the opcode, 2nd machine cycle is to read 8 LSB of 16-bit data (2500H) & 3rd machine cycle to read 8 MSB of 16-bit data (2501H).

LXI H, 2500H

21, DD, 25 in code form

Machine cycle: 3 T-state: 10

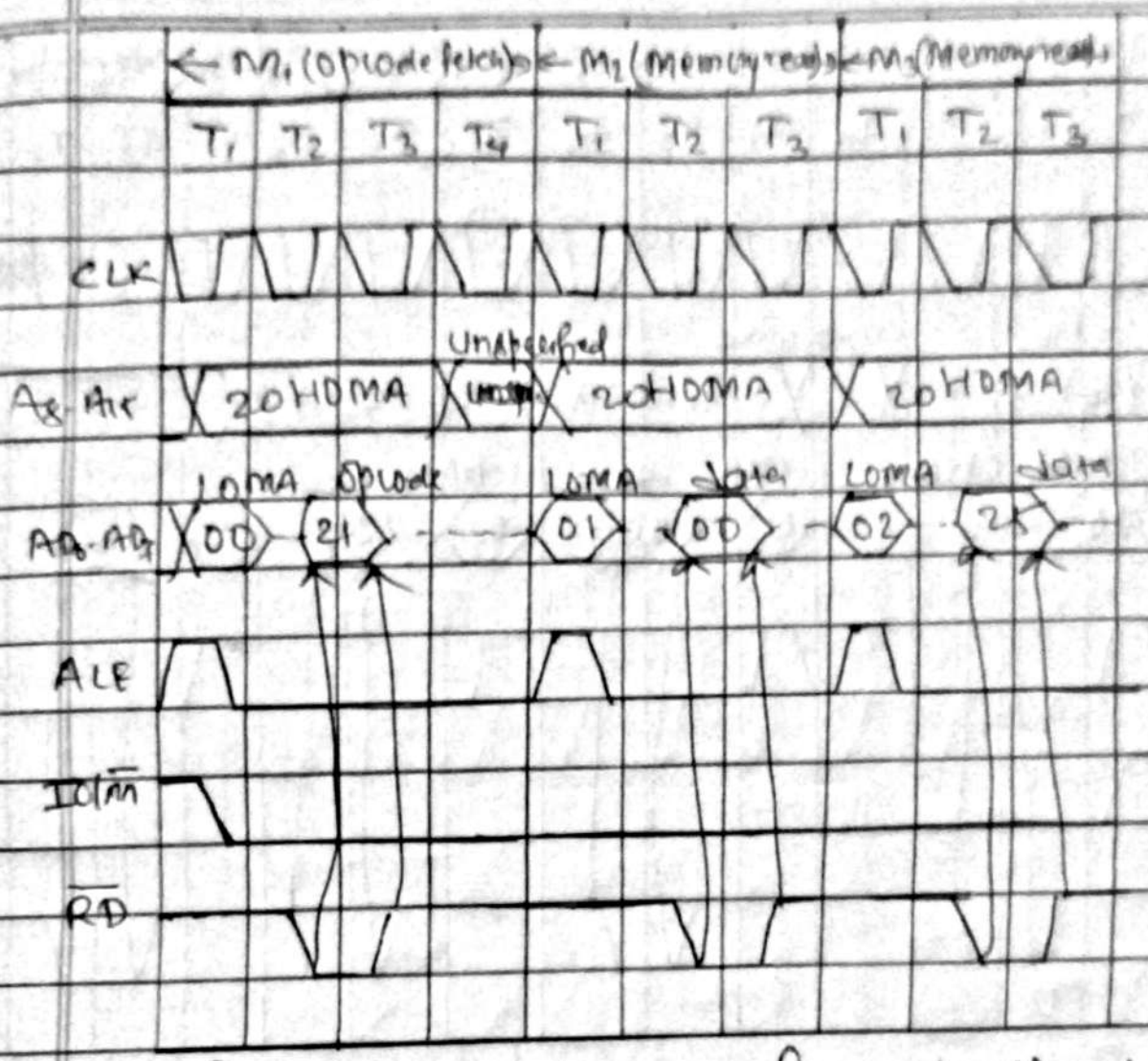


fig: Timing diagram for instruction LXI H,2500H

The instruction LDA 2400H requires four machine cycles. 1st machine cycle is to fetch the opcode, 2nd machine cycle for reading 8 LSB of memory address (2400H), 3rd machine cycle for reading 8 MSB of memory address. In the 4th machine cycle, microprocessor sends memory address 2400H to get its content into the accumulator.

LDA 2400H 3A,00,24 in the code form

Machine cycle : 4 T-State : 13

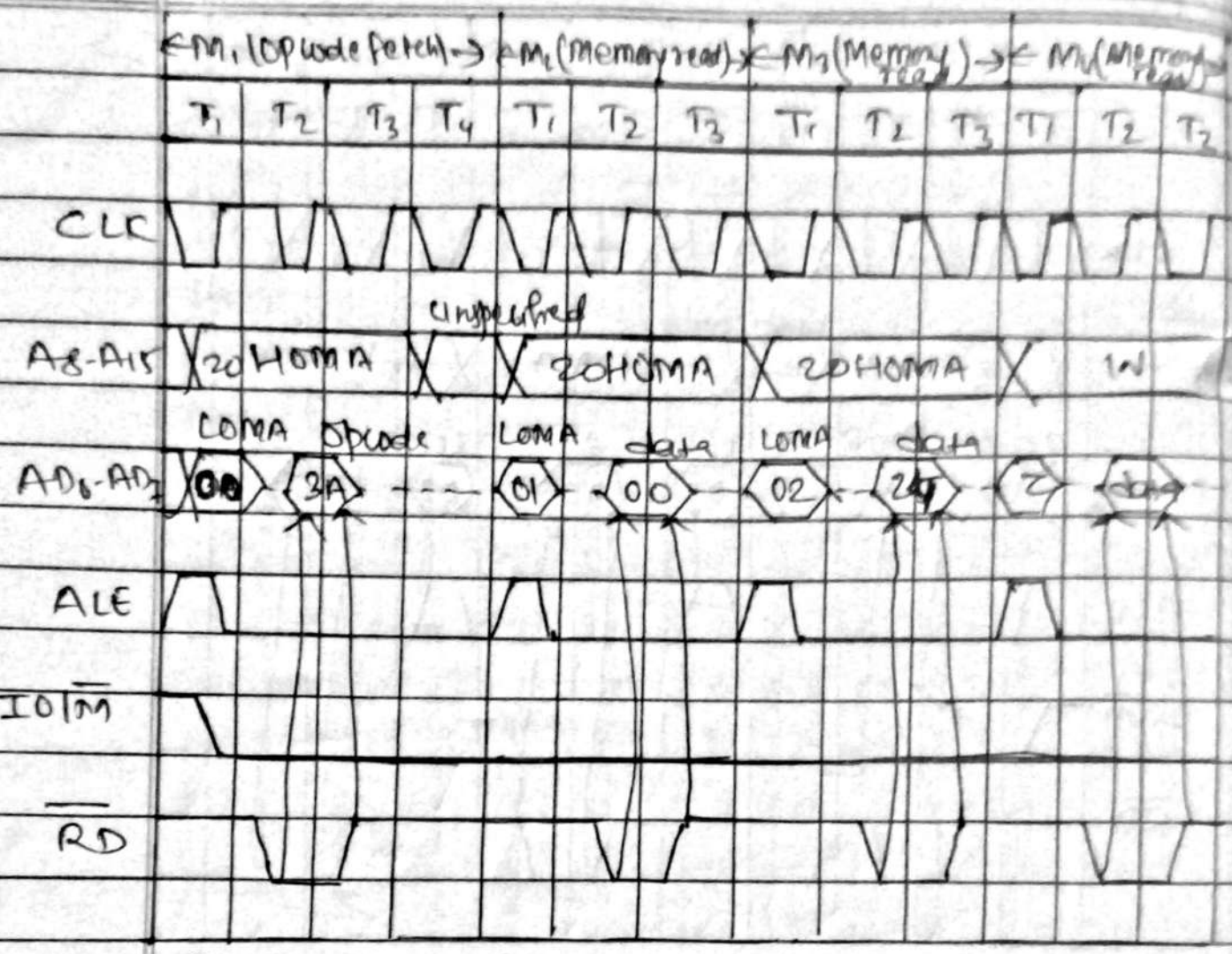


fig: Timing diagram of LDA 2400H

Timing diagram of memory write:-

In memory write cycle, the microprocessor sends data from the accumulator or any other register to the memory. In this cycle, status signals S_0 & S_1 are 1 & 0 respectively. The instruction MOV M,A , STA 2500H etc use memory write cycle.

STA 2400H 32,00,24 in the code form

Machine cycle: 4 T-State: 13

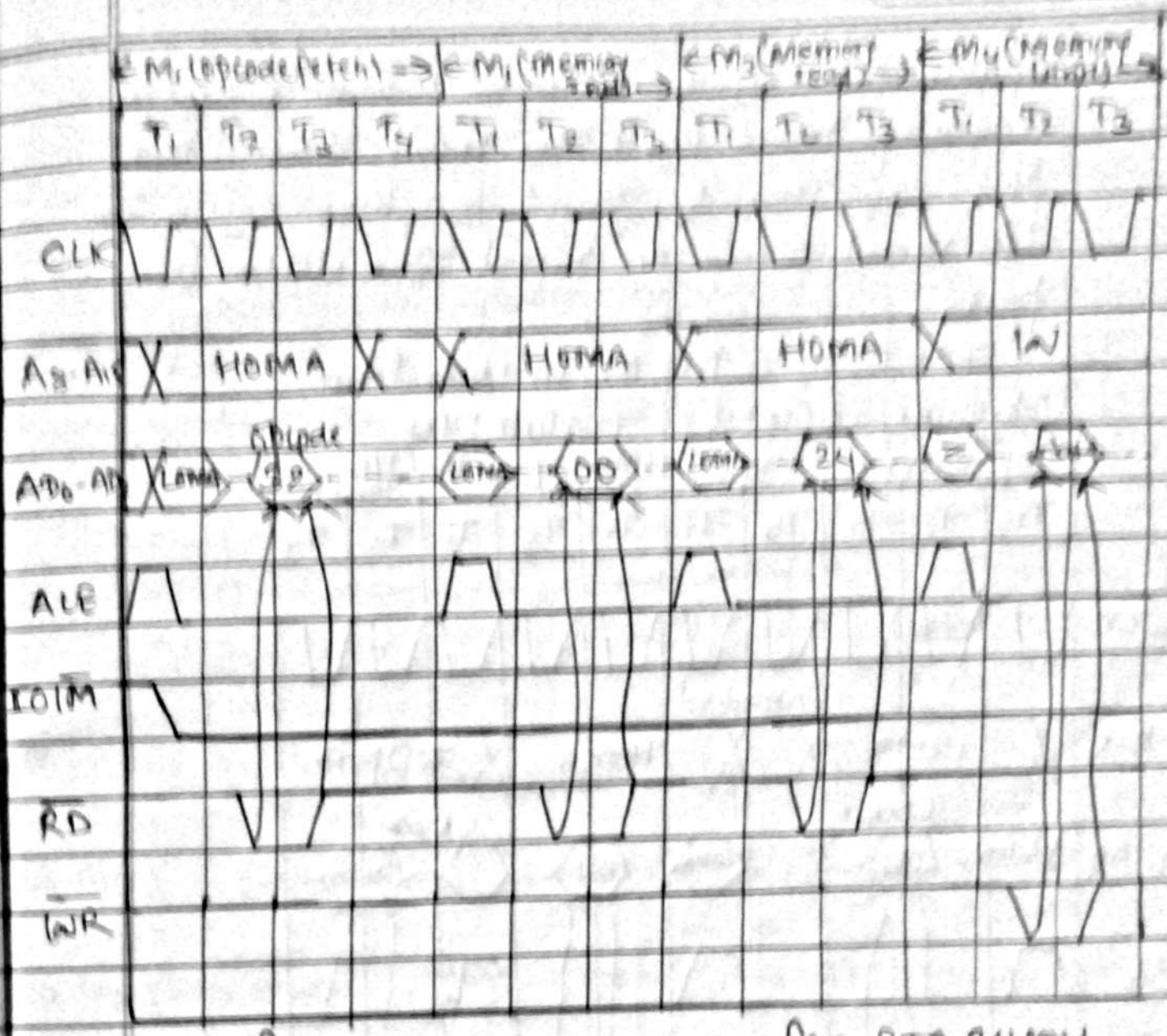


fig: Timing diagram for STA 2400H

Timing diagram for I/O Read:

In an I/O read cycle, the microprocessor reads the data available at an input port. The data is placed in the accumulator.

IN instruction is used for I/O read. It is two byte long. It requires three machine cycles for execution. The four machine cycle is

opcode fetch cycle, Second machine cycle is memory read cycle to read the input port address & third machine cycle is I/O read cycle to read the data from port.

IN 01H DB, 01 in code form

Machine cycle: 3 T-state: 10

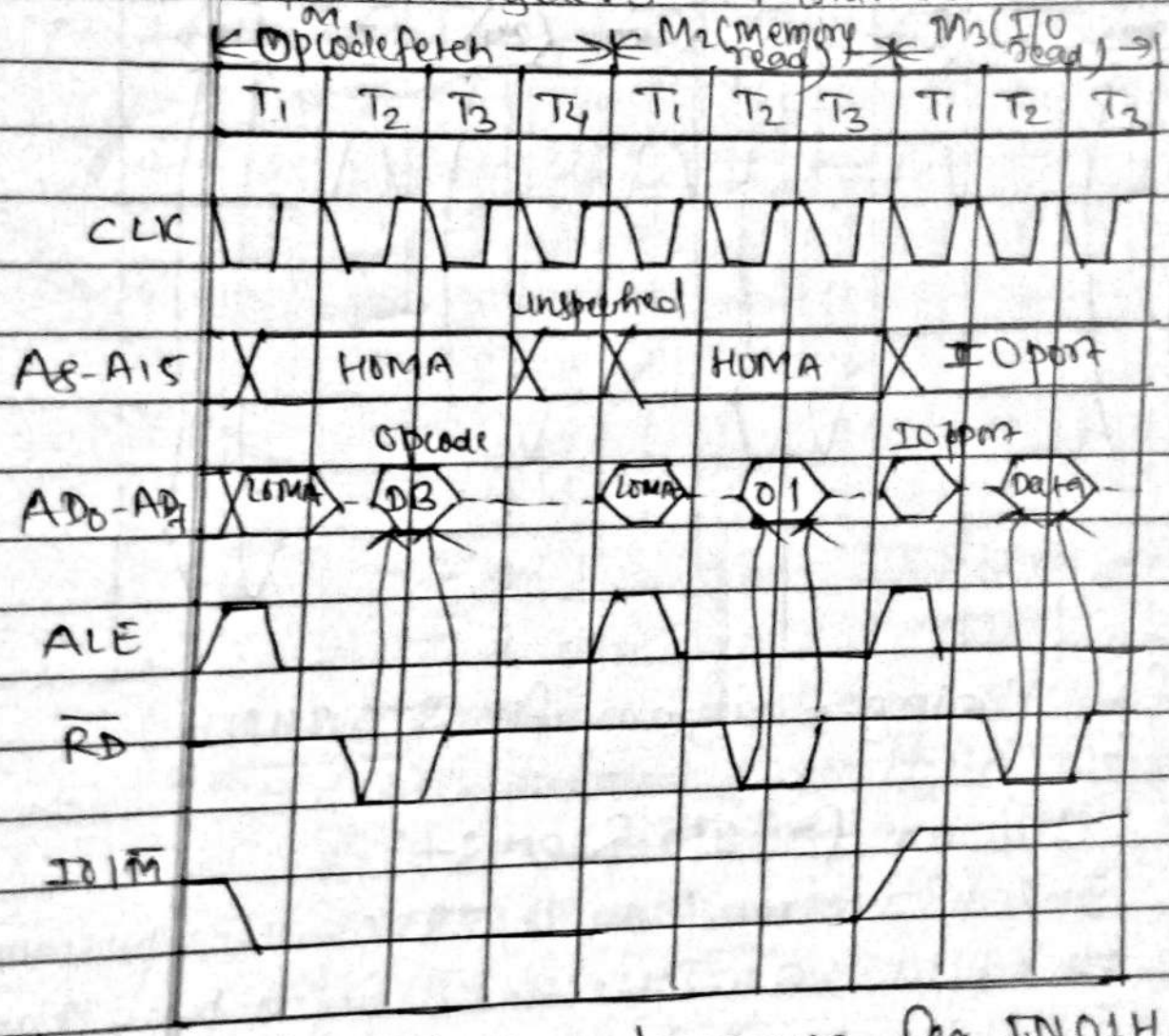


fig: Timing diagram for IN 01H

Timing diagram for I/O write:-

In I/O write cycle, the microprocessor sends data to I/O port from the accumulator

OUT instruction is used for I/O write. It is two byte long instruction & requires three machine cycles. 1st machine cycle is for opcode fetch operation, 2nd machine cycle is memory read cycle for reading I/O address from memory & 3rd machine is I/O write cycle for sending data to the I/O device.

OUT 02H D3, 02H in the code form

Machine cycle : 3 T-State : 10

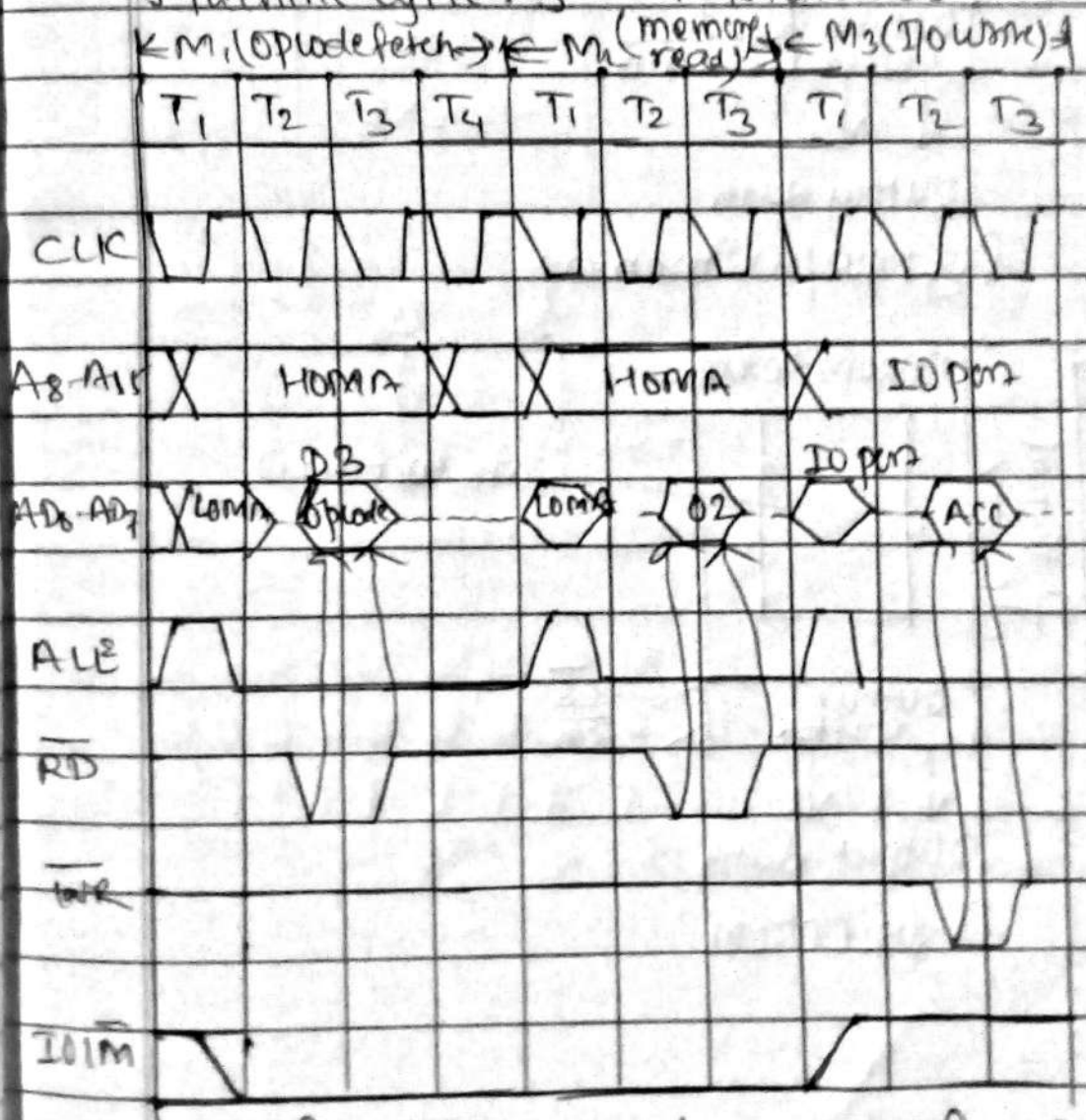


fig: Timing diagram for OUT 02H

Memory Interfacing & Generation of chip select Signal :-

Memory structure :-

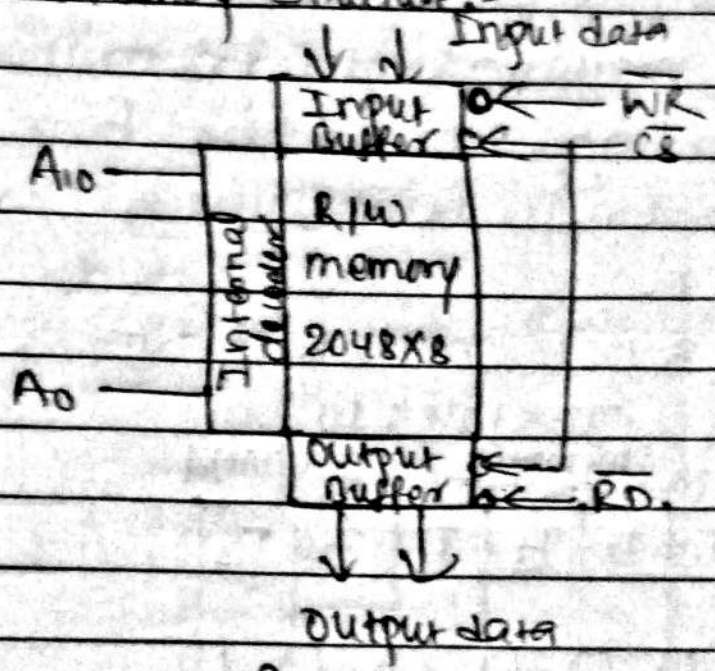


fig (a): R/W memory

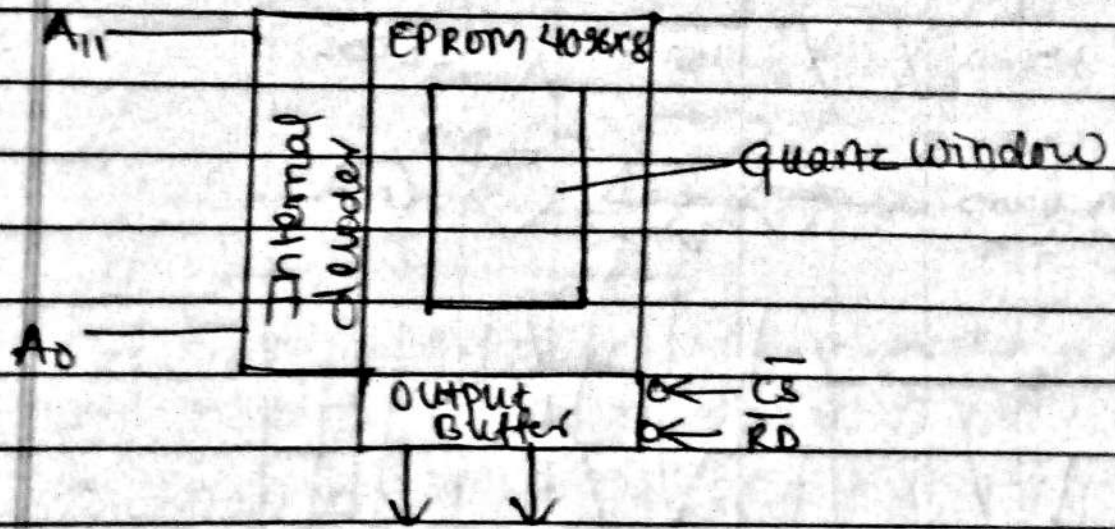


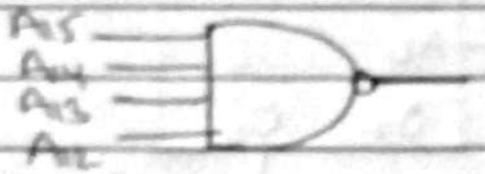
fig (b): EPROM

Address decoding:

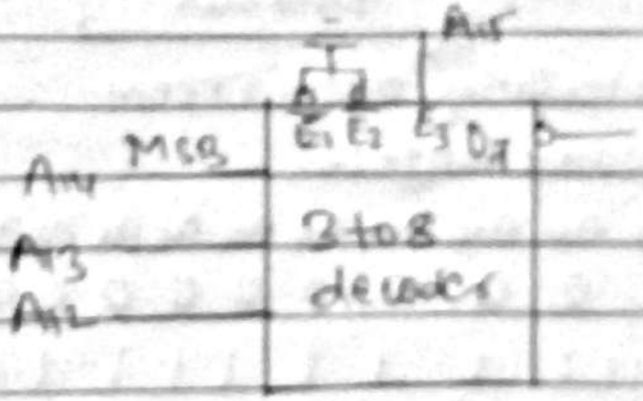
The process of address decoding should result in identifying a register for given address. It should be able to generate a unique pulse for a given address. Eg in fig(b), 12 address lines (A_{12}–A_0) are connected to memory chip & remaining 4 address lines (A_{15}–A_{12}) of 8085 microprocessor must be decoded.

Two methods of decoding these lines:

One by using NAND gate & other by using 3-to 8 decoder.



(a) NAND gate



(b) 3-to 8 decoder

Output of NAND gate ~~high~~ active & selects chip only when all address lines A_{15}–A_{12} are at logic 1.

We can obtain same result by using O_7

of 3 to 8 decoder, which is capable of decoding eight different input address.

Memory interfacing:

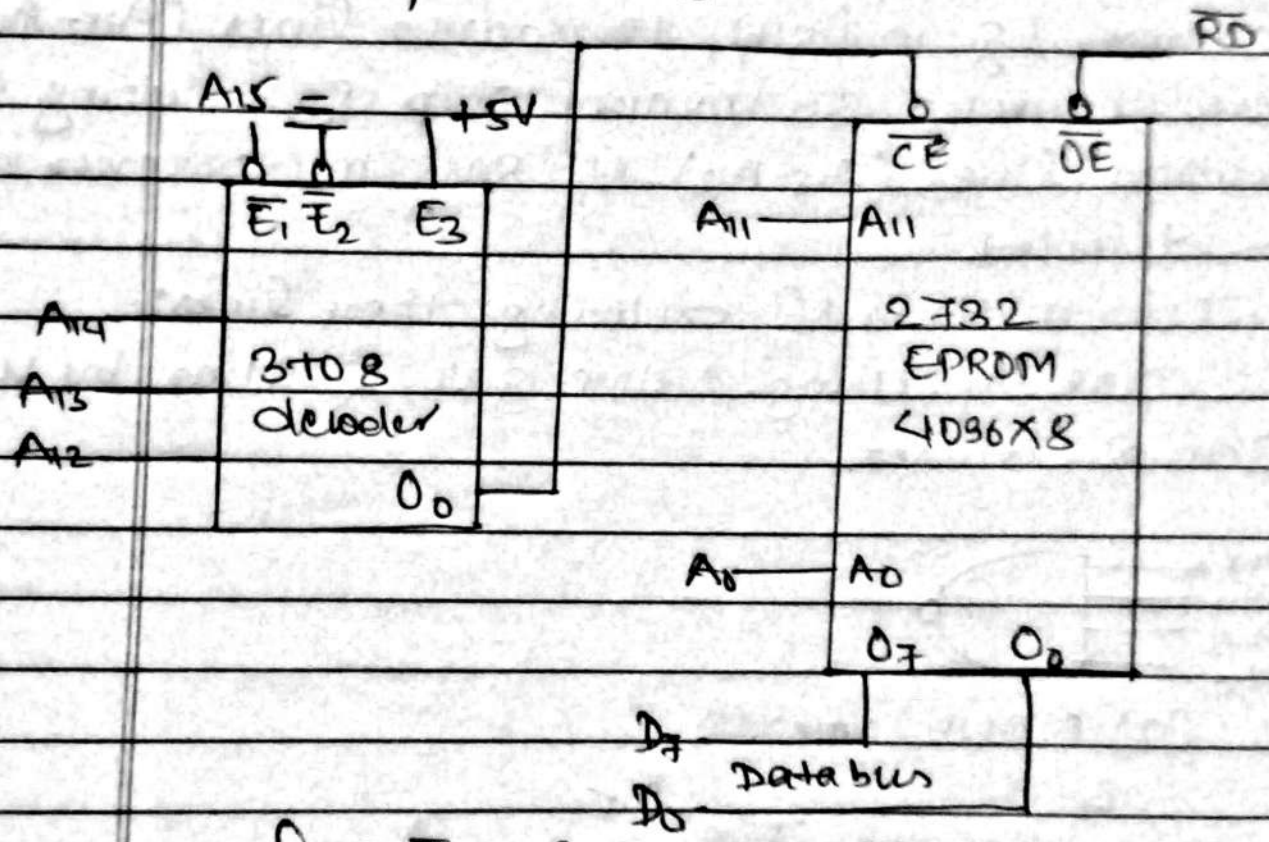


fig: Interfacing 2732 EPROM

Memory address range:-

A15 A14 A13 A12 A11 A10 A9 A8 A7 A6 A5 A4 A3 A2 A1 A0

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 = 0000

0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 = 0FFF

chip enable — Register enable

Page
Date

Assembly language programming:-

The writing of program in machine language is very difficult, tiresome, boring & error prone. Hence to facilitate programmers easily understanding languages have been developed. Assembly language is one of them. A programmer can easily write a program in alphanumeric symbols instead of zeros & ones. Meaningful & ~~rember~~ rememberable symbols are chosen for the purpose. Examples are : ADD for addition, SUB for subtraction etc. Such symbols are called mnemonics. A program written in mnemonics is known as assembly language program. The writing of program in assembly language is much easier & faster as compared to the writing of a program in machine language. When a program is written in a language other than machine language, a computer can't understand it. Consequently a program written in a language other than machine languages has to be translated into machine language before it is executed. The translation is done with the help of software. A program which translates an assembly language program into a machine language is called assembler.

Assembly language programs are called "source codes". Machine language programs are

called "object codes." A translator converts source codes to object codes & then into executable formats. The process of converting source code into object code is called compilation & assembler does it. The process of converting object codes into executable formats is called linking & linker does it.

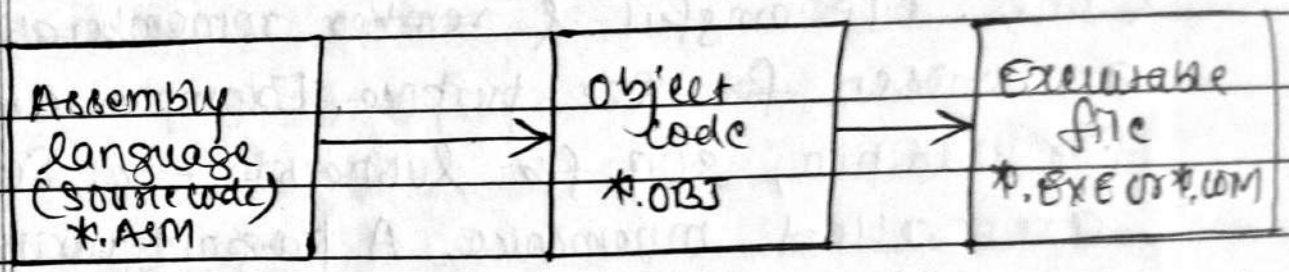


fig: process of compilation & linking assembly language

Each instruction in an assembly language is composed of three or four fields as follows

- Label field
- Instruction, mnemonics or op-code field
- Operand field
- Comment field

As an example, a typical program for adding two 8-bits numbers written in 8085 assembly language is given below:-

Label	Mnemonics	Operand	comments
	MVI	A, 10H	; move 10H into acc.
	ADI	20H	; add 20H to acc
	LXI	H, 2000H	; load H-L pair ; with 2000H.

MOV M,A ; Move content of acc into memory
JMP START ; jump to beginning of program

There are two ways of converting an assembly language program into machine language, namely, via manual assembly or by using an assembler.

With the manual assembly, the programmer is the assembler i.e. the programmer ~~translates~~ translates each mnemonics into its numerical machine language representation by looking up a table of the microprocessor's instruction set, which contains both assembly & machine language instruction. Manual assembly is acceptable for short programs but becomes very inconvenient for large programs.

When an assembler is used, the assembler program reads each assembly instruction of a program as ASCII characters & translates them into respective binary opcodes. For example, consider HLT ~~instruction~~ instruction for 8085, its binary opcode is 01110110_2. An assembler would convert HLT into 01110110_2.

An advantage of the assembler is address computation. Most programs use addresses within the program as data or as labels for jumps or calls. When programming in machine language, these addresses must be calculated by manually. Every time the program changes size, all the addresses must be recomputed. This is a time consuming

task. The assembler solves this problems by allowing the programmer to assign a symbol to an address called a label. The programmer may then reference that address elsewhere by using the symbol. The assembler computes the actual address for the programmer & fills it in automatically.

Disassembler:-

It is a software aid to convert a machine language program to an assembly language program.

(1) Write ALP to add two hex numbers stored at memory locations 2050H & 2051H & store the result at 2052H.

Program:-

Label	Mnemonics	Operand	Comments
	LXI	H, 2050H	
	MOV	A, M	
	INX	H	
	ADD	M	
	INX	H	
	MOV	M, A	
	HLT		

Page
Date

2) Write ALP to multiply two numbers stored in location 2050H & 2051H & store the result at 2052H.

Program:-

Label	Mnemonics	Operand	Comments
	LXI	H,2050H	
	MOV	C,M	
	XRA	A	
	INX	H	
l1:	ADD	M	
	DCR	C	
	JNZ	l1	
	INX	H	
	MOV	M,A	
	HLT		

3) Write ALP to divide two numbers stored in memory locations 2050H & 2051H & store the remainder at 2070H & quotient at 2071H.

Program:-

Label	Mnemonics	Operand	Comments
	LXI	H,2050H	
	MOV	A,M	
	MVI	D,00H	
	INX	H	
label1:	SUB	M	
	INR	D	
	CMP	M	

Page
Date

```
JNC  label1
STA  2070H
MOV  A,D
STA  2071H
HLT
```

4) Write ALP to disassemble the byte stored at memory location 2040H & store the result at memory location 2041H & 2042H.

Program:-

Label	Mnemonic	Operand	Comments
	LXI	H,2040H	
	MOV	A,M	
	MOV	B,A	
	ANI	0FH	
	INX	H	
	MOV	M,A	
	MOV	A,B	
	RRC		
	RRC		
	RRC		
	RRC		
	ANI	0FH	
	INX	H	
	MOV	M,A	
	HLT		

Page
Date

5) ALP to find 1's & 2's complement of data stored at memory location 2050H & store the result at 2051H & 2052H.

Program :-

Label	Mnemonics	Operand	Comments
	LXI	H, 2050H	
	MOV	A, M	
	CMA		
	INX	H	
	MOV	M, A	
	ADI	01H	
	INX	H	
	MOV	M, A	
	HLT		

6) Program to EX-OR two data without using EX-OR instructions.

Program :-

Label	Mnemonics	Operand	Comments
	LXI	H, 2050H	
	MOV	A, M	
	MOV	B, A	
	CMA		
	INX	H	
	ANA	M	
	MOV	C, A	
	MOV	A, M	

```
CMA
ANA  B
ORA  C
INX  H
MOV  M,A
HLT
```

7) Program to find largest number in a given series of data. The length is given in memory location 203FH & the series itself starts from 2040H. Store the result at 2060H.

Program:-

```
Label   Mnemonics  Operand  Comments
        LXI        H,203FH
        MOV        C,M
        DCR        C
        INX        H
        MOV        A,M
l2:     INX        H
        CMP        M
        JNC        l1
        MOV        A,M
l1:     DCR        C
        JNZ        l2
        STA        2060H
        HLT
```

Page
Date

8) Program to distinguish even number from a given set of data stored from memory location 2050H & to store the even number from 2070H.

Program:-

Label	Mnemonics	Operand	Comments
	LXI	H, 2050H	
	LXI	D, 2070H	
	MOV	B, M	
	INX	H	
l2:	MOV	A, M	
	RRC		
	JC	l1	
	RLC		
	STAX	D	
	INX	D	
l1:	INX	H	
	DCR	B	
	JNZ	l2	
	HLT		

9) Program to convert BCD number stored at 2050H into binary code & store it at 2051H.

Label	Mnemonics	Operand	Comments
	LXI	H, 2050H	
	MOV	A, M	
	ANI	0FH	
	MOV	C, A	

Page
Date

```
        MOV A,M
        RRC
        RRC
        RRC
        RRC
        ANI 0FH
        MOV D,A
        XRA A
L1:     ADI 0AH
        DCR D
        JNZ L1
        ADD C
        INX H
        MOV M,A
        HLT
```

Input:- Output:-

Address	Data	Address	Data
205DH	$(86)_{10}$	2051H	(56)H

10) Write a program to arrange six bytes of data stored from the location D000H in descending order.

Data(H): 02, 0A, 01, 11, 05, 06

Program:-

Label	Mnemonics	Operands	Comments
	LXI	H, D000H	
	MVI	B, 06H	

```
        ~~DCR~~ B
        MOV C,B
        DCR C
        MOV E,C
label2: MOV A,M
        INX H
        CMP M
        JNC label1
        MOV D,M
        MOV M,A
        DCX H
        MOV M,D
        INX H
label1: DCR C
        JNZ label2
        LXI H,D000H
        MOV C,E
        DCR B
        JNZ label2
        HLT
```

11) Exchange the contents of memory location 2000H & 4004H.

Program:-

Label	Mnemonics	Operand	Comments
	LDA	2000H	
	MOV	B,A	

Page
Date

```
LDA  4000H
STA  2000H
MOV  A,B
STA  4000H
HLT
```

12) Addition of two 16-bit numbers

Label	Mnemonics	Operand	Comments
	LHLD	2040H	
	XCHG		
	LHLD	2042H	
	DAD	D	
	SHLD	2044H	
	HLT		

Page
Date

Stack:-

The stack in an 8085 microcomputer system can be described as a set of memory locations in the R/W memory, specified by a programmer in a main program. The memory locations are used to store binary information temporarily during the execution of a program.

The beginning of the stack is defined in the program by using the instruction LXI SP which loads 16-bit memory address in the stack pointer register of the microprocessor. Once the stack is defined, storing of data bytes begins at the memory address that is one less than the address in the stack pointer register. For example, if the stack pointer is loaded with the memory address 2099H (LXI SP,2099H), the storing of data bytes begins at 2098H & continues in reversed numerical order (decreasing memory address such as 2098H, 2097H etc).

Data bytes in the register pair of the microprocessor can be stored on the stack (two at a time) in reverse order (decreasing memory address) by using the instruction PUSH. Data bytes can be transferred from the stack to respective registers by using the instruction POP. Because two data bytes are being stored at a time, 16-bit memory

address in the stack pointer register is decremented by two, when data bytes are retrieved, the address is incremented by two.

The stack is shared by the programmer & the microprocessor. The programmer can store & retrieve the contents of a register pair by using PUSH & POP instructions. Similarly the microprocessor automatically stores the contents of the program counter when a subroutine is called.

```
LXI SP,2099H
LXI H,42F2H
PUSH H
POP H
```

The instruction PUSH H stores the content of registers H-L on the stack i.e. 42H at 2098H & F2H at 2097H. Similarly the instruction POP H loads H-L register with contents of two top locations of stack i.e. Reg L with F2H & H with 42H.

Subroutine :-

A subroutine is a group of instructions written separately from the main program to perform a function that occurs repeatedly in the main program. For example, if a time delay is

required between three successive events, three delays can be written in the main program. To avoid repetition of the same delay instructions, the subroutine technique is used. Delay instructions are written once, separately from the main program & are called by the main program when needed.

8085 microprocessor has two instructions to implement subroutines: CALL (call subroutine) & RET (return to main program from a subroutine). When a subroutine is called, the contents of program counter which is the address of instruction following the CALL instruction, is stored on the stack & the program execution is transferred to the subroutine address. When RET instruction is executed at the end of subroutine, the memory address stored on the stack is retrieved & the sequence of execution is resumed in the main program.

Example

Program to make output port high & low for the same interval.

```
START: MVI A, 01H
       OUT PORT#
       CALL DELAY
       XRA A
       OUT PORT#
```

```
        CALL DELAY
        JMP  START
DELAY:  MVI B,FFH
L1:     DCR B
        JNZ L1
        RET
```

Types of assembler

- One pass assembler
- Two pass assembler

One pass assembler :-

An assembler which goes through assembly language program only once is known as One pass assembler. Such an assembler must have some technique to take forward reference into account. Assembly language programs use labels that may appear later on in the program. Such labels point to forward references.

Two pass assembler :-

An assembler which goes through an assembly language program twice is called two pass assembler. Such an assembler doesn't face difficulty with forward references. During the first pass, it collects all labels & it generates table of symbol. During second pass, it produce the machine code for each instruction & assigns addresses to each of them. It assigns

Page
Date

addresses to labels by counting their position from the starting address.

Pass I

Address (Hex)	Machine code (Hex)	Label	Mnemonics	Operand	Table Symbol
2000H		START:	IN	PORT0	PORT0 00
2002H			OUT	PORT1	PORT1 01
2004H			JMP	START	START 2000

Pass II

2000H DB00

2002H D301

2004H C30020

Assembler directives:-

The assembler directives are the instructions to the assembler concerning the program being assembled, they are also called pseudo instructions or pseudo opcodes. These instructions are neither translated into machine code nor assigns any memory locations in the object file. Some of the important assembler directives are as follows:-

Assembler Directives	Example	Description
1. ORG (Origin)	ORG 2000H	The next block of instructions should be stored in memory locations starting at 2000H.
2. END	END	End of assembly. HLT

Page
Date

suggests the end of program but that doesn't necessarily mean it is end of assembly.

3. EQU (Equate) PORT1 EQU 01H — The value of term PORT1 is equal to 01H. Generally, this means PORT1 has port address 01H.

INBUF EQU 2099H — The value of term INBUF is 2099H. This may be the memory location used as input buffer.

STACK EQU INBUF+1 — The equate can be expressed by using the label of another equate. This example defines the stack as next location of INBUF

4. DB (Define byte) DATA: DB A2H, 9FH — Initializes an area byte by byte. Assembler bytes of data are stored in successive memory location until all values are stored.

5. DW (Define word) DATA: DW 2050H — Initializes an area two bytes at a time.

6. DS (Define Storage) OUTBUF: DS4 Reserves a specified no. of memory locations. In this example, four memory locations are reserved for OUTBUF.

Macro assembler:-

An assembler having facility for macro is known as macro assembler.

Macro:-

A sequence of instructions to which a name is assigned is called a macro. The name of macro is used in assembly language programming. Each macro is clearly defined & unique name is assigned to it.

Example 1

```
COMPLE  MACRO  ADDRESS
        LXI    H,ADDRESS
        MOV    A,M
        CMA
        ENDM
```

In above example, COMPLE is the name of the macro & ADDRESS is a parameter. ENDM is used to end the macro. If we write COMPLE 2500H in an assembly language program, the assembler will replace this macro by the following sequence of instructions in the program.

Page
Date

LXI H,2500H
MOV A,M
CMA

This sequence of instructions will take the complement of the content of memory location 2500H.

Example 2

SHIFT MACRO
ADD A
ENDM

SHIFT is the name of the macro. If SHIFT is written in the program, the assembler will replace it by ADD A.

Once a sequence of instructions is written & the macro name is assigned to it, the macro name can be used frequently in the program. It makes program easier to read & understand. Macro & subroutines are similar. A subroutine requires CALL & RETURN instructions where as macros do not. The macros execute faster than subroutine. Macros are used for short sequence of instructions where as subroutines for longer ones, preferably for 10 instructions & more.

Specify the register contents & flag status as the following instructions are executed. Specify also the output at PORTO.

A	B	S	Z	CY	
00	FF	0	1	0	Initial contents

```
MVI A, F2H
MVI B, 7AH
ADD B
OUT PORTO
HLT
```

Soln:-

	A	B	S	Z	CY
	00	FF	0	1	0
MVI A, F2H	F2	FF	0	1	0
MVI B, 7AH	F2	7A	0	1	0
ADD B	6C	7A	0	0	1
OUT PORTO	6C	7A	0	0	1
HLT	6C	7A	0	0	1

PORTO = 6CH

8085 microprocesnor instructions :-

A computer receives data from the user, processes data & sends the result back to the user. The computer simply performs a given task on a specified data in response to certain instructions. An instruction is a command given to the computer to perform a specified operation on given data.

Each instruction has two parts: one is the task to be performed called the operation code (Opcode) & the second is the data to be operated on called the operand. The operand can be data (8 or 16 bit), address or register or it can be implicit.

Instruction word size:-

The 8085 instruction set is classified into the following three groups according to word size:

1) One word or 1-byte instruction
2) Two word or 2-byte instruction
3) Three word or 3-byte instruction

One byte instruction:-

1-byte instruction includes the ~~Operand~~ Opcode & ~~Opcode~~ operand in the same byte. E.g. MOV A,B moves the content of register B to register A. 78H is the opcode for MOV A,B. In this instruction, both operand registers are specified.
01111000
A B

E.g. ADD B (80H) (10000000$_B$) add the content of register B to the content of accumulator. In this instruction, the operand B is specified & the accumulator is assumed.

CMA, complement each bit in the accumulator. In this instruction, the accumulator is assumed to be the implicit operand.

These instructions are stored in 8-bit binary format in memory, each requires one memory location.

Two byte instruction :-

In 2-byte instruction, the first byte specifies the operation code & the second byte specifies the operand. Example are:

(i) MVI B,05H , move 05H to reg B.

06,05 , MVI B,05H in the code form

The first byte 06H is the opcode for MVI B & second byte 05H is the data which is to be moved to reg B.

(ii) IN 01H , Read data at port B

DB,01 in code form

A two byte instruction is stored in two consecutive memory location.

Three byte instruction :-

In 3-byte instruction, the first byte specifies

the opcode, the following two bytes specify 16-bit address. Note that the second byte is the low-order address & third byte is high order address.

Examples are:

(i) LXI H, 2400H , load H-L pair with 2400H

21, 00, 24 in code form

1st byte 21H is the opcode for instruction LXI H. 2nd byte 00H is low-order address & 3rd byte 24H is high-order address.

(ii) LDA 2500H

3A, 00, 25

A 3-byte instruction is stored in three consecutive memory locations.

8085 Instruction Set:-

Data transfer group:-

1) MOV r1, r2 (move content of reg r2 to reg r1)
E.g. MOV A, B

2) MOV r, M (move content of memory to reg)
E.g. MOV A, M

3) MOV M, r (move content of register to memory)
E.g. MOV M, C moves the content of register C to the memory location whose address is in H-L pair.

4) MVI r, data (move immediate data to register)
E.g. MVI A, 05H

5) MVI M, data (move immediate data to memory)

E.g. MVI M,08H will move 08H to memory location whose address is in H-L pair.

6) LXI rp, data16 (load reg pair immediate)
E.g. LXI H,2500H loads 2500H into H-L pair.

7) LDA addr (load accumulator direct)
E.g. LDA 2400H will load the content of memory location 2400H into the accumulator.

8) STA addr (store accumulator direct)
E.g. STA 2400H will store the content of accumulator in the memory location 2400H.

9) LHLD addr (load H-L pair direct)
LHLD 2500H will load the content of memory location 2500H into register L. The content of the memory location 2501H is loaded into the register H.

10) SHLD addr (store H-L pair direct)
SHLD 2500H will store the content of reg L in the memory location 2500H. The content of register H is stored in the memory location 2501H.

11) LDAX rp (load accumulator indirect)
LDAX B will load the content of memory location whose address is in B-C pair, into the accumulator.

12) STAX rp (store accumulator indirect)
STAX D will store the content of accumulator in the memory location whose address is in D-E pair.

13) XCHG (Exchange the content of H-L with D-E pair)

The contents of H-L pair are exchanged with contents of D-E pair.

Arithmetic group:-

1) ADD r (add register to accumulator)
ADD B will add the content of register B to the content of accumulator & the sum is placed in the accumulator.

2) ADD M (add memory to accumulator)
The content of memory location addressed by H-L pair is added to the content of accumulator & sum is placed in the accumulator)

3) ADC r (add reg with carry to accumulator)
ADC B will add the content of reg B & carry status to the content of accumulator & sum is placed in the accumulator.

4) ADI data (add immediate data to accumulator)
ADI 08H will add 08H to the content of accumulator & place the result in the accumulator.

5) ACI data (add with carry immediate data to acc)
ACI 08H will add 08H & carry status to the content of accumulator.

6) DAD rp (add reg pair to H-L pair)
DAD B will add the content of reg pair BC to the content of HL pair.

7) SUB r (Subtract reg from accumulator)
SUB B will subtract the content of reg B from

The content of accumulator & the result is placed in accumulator.

8) SUB M (Subtract memory from accumulator)
The content of memory location addressed by H-L pair is subtracted from the content of the accumulator. The result is placed in the accumulator.

9) SBB r (Subtract reg from accumulator with borrow)
SBB B, the content of reg B & carry status are subtracted from the content of accumulator. The result is placed in the accumulator.

10) SUI data (Subtract immediate data from acc)
SUI 05H will subtract 05H from the content of accumulator & place the result in the accumulator.

11) SBI data (Subtract immediate data from acc with borrow)
The data & carry status are subtracted from the content of acc & result is placed in the accumulator.

12) INR r (Increment reg content)
INR B, the content of reg B is incremented by 1

13) INR M (Increment memory location content)
The content of memory location addressed by H-L pair is incremented by one.

14) DCR r (Decrement reg content)
DCR B, the content of reg B is decremented by 1.

15) DCR M (Decrement memory contents)

The content of memory location addressed by H-L pair is decremented by 1.

16) INX rp (Increment reg pair)
INX B, the content of reg pair BC is incremented by 1.

17) DCX rp (Decrement reg pair)
DCX B, the content of reg pair BC is decremented by 1.

18) DAA (Decimal adjust accumulator)
After execution of ADD, ADC etc instructions, the result is in hexadecimal & it is placed in the accumulator. DAA instruction operates on this result & gives the final result in decimal system.

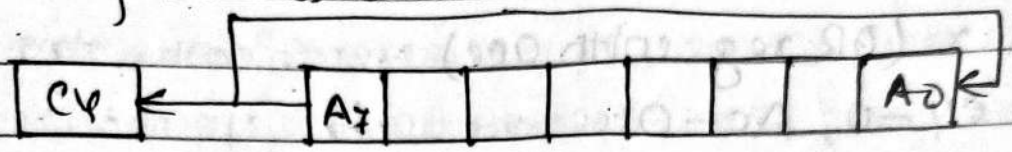
Logical group:-
The instructions of this group perform AND, OR, EX-OR operations, compare, rotate or take complement of data in reg or memory.

1) ANA r (AND reg with accumulator)
ANA B, content of reg B is ANDed with content of accumulator & the result is placed in the accumulator. (CY=0, AC=1)

2) ANA M (AND memory with accumulator)
The content of memory location addressed by H-L pair is ANDed with accumulator & result is placed in accumulator. (CY=0, AC=1)

3) ANI data (AND immediate data with accumulator)
- CY=0, AC=1
4) ORA r (OR reg with acc)
- CY=0, AC=0
5) ORA M (OR memory with acc)
- CY=0, AC=0
6) ORI data (OR immediate data with acc)
- CY=0, AC=0
7) XRA r (EX-OR reg with acc)
- CY & AC=0
8) XRA M (EX-OR memory with acc)
- CY & AC=0
9) XRI data (EX-OR immediate data with acc)
- CY & AC=0
10) CMA (complement the accumulator)
11) CMC (complement the carry status)
12) STC (set carry status), CY=1
13) CMP r (compare register with acc)
The content of register r is subtracted from the content of accumulator & status flags are set according to the result of the subtraction. But the result is discarded. The content of accumulator remains unchanged.
14) CMP M (compare memory with acc)
15) CPI data (compare immediate data with acc)
16) RLC (Rotate acc left)
The content of accumulator is rotated

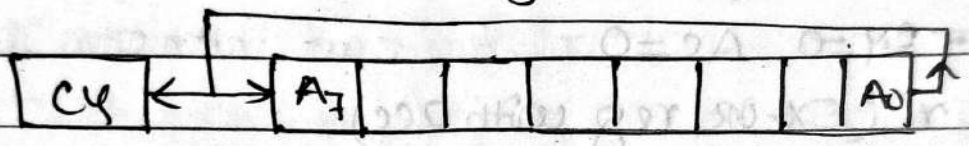
one bit. The seventh bit of the acc is moved to carry bit as well as to the zero-bit of the acc.



Carry status ACC

fig: Schematic diagram for RLC

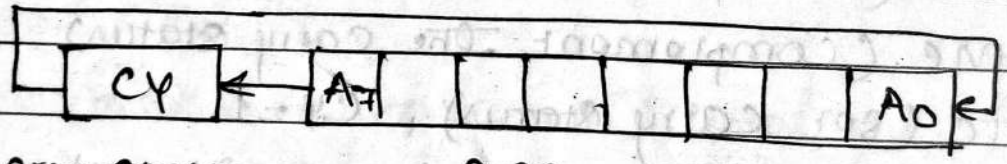
17) RRC (Rotate acc right)



Carry status ACC

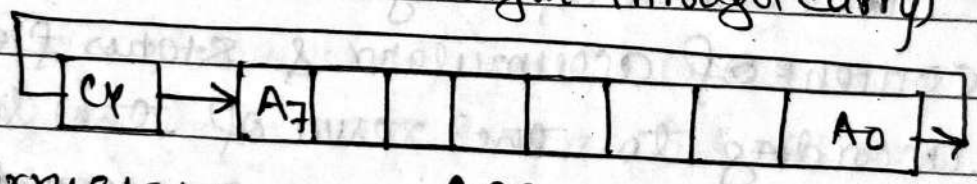
18) RAL (Rotate acc left through carry)

The content of acc is rotated left one bit through carry. 7th bit of acc is moved to carry & carry-bit is moved to zero bit of the acc.



Carry status ACC

19) RAR (Rotate acc right through carry)



Carry status ACC

Branch group:-

The instructions of this group change the normal sequence of the program. There are two types of branch instructions: conditional & unconditional. The conditional

branch instruction transfer the program to the specified label when certain condition is satisfied. The unconditional branch instructions transfer the program to the specified label unconditionally.

1) JMP addr(label) (unconditional jump: jump to the instruction specified by the address)

The address of the label is the address of the memory location for next instruction to be executed.

2) Conditional jump:-

After the execution of conditional jump instructions the program jumps to the instruction specified by the address (label) if the specified condition is fulfilled. The program proceeds further in the normal sequence if the specified condition is not fulfilled.

i) JZ addr(label) (jump if result is zero)

The program jumps to the instruction specified by the address (label) if the result is zero. (i.e. Z=1)

ii) JNZ addr(label) (jump if result is not zero) (i.e. Z=0)

iii) JC addr(label) (jump if there is carry) (i.e. CY=1)

iv) JNC addr(label) (jump if there is no carry) (i.e. CY=0)

v) JP addr(label) (jump if the result is plus) (i.e. S=0)

vi) JM addr(label) (jump if the result is minus) (i.e. S=1)

vii) JPE addr(label) (jump if even parity) (i.e. P=1)

viii) JPO addr(label) (jump if odd parity) (i.e. P=0)

3) CALL addr(label) (unconditional CALL: call the subroutine identified by the operand)

4) Conditional CALL:-

If the condition is true, program calls the specified subroutine.

i) CC addr(label) - call subroutine if CY = 1
ii) CNC addr(label) - " " " CY = 0
iii) CZ addr(label) - " " " Z = 1
iv) CNZ addr(label) - " " " Z = 0
v) CP addr(label) - " " " S = 0
vi) CM addr(label) - " " " S = 1
vii) CPE addr(label) - " " " P = 1
viii) CPO addr(label) - " " " P = 0

5) RET (Return from subroutine)

6) Conditional return:-

If the condition is true, the program returns from the subroutine.

i) RC returns from subroutine if CY = 1.
ii) RNC " " " " CY = 0.
iii) RZ " " " " Z = 1.
iv) RNZ " " " " Z = 0.
v) RP " " " " S = 0.
vi) RM " " " " S = 1.
vii) RPE " " " " P = 1
viii) RPO " " " " P = 0

7) RST n (Restart):-

The program jumps to the instruction starting at

restart location. The address of restart location is 8 times n.

Instruction	Restart location
RST0	0000
RST1	0008
RST2	0010
RST3	0018
RST4	0020
RST5	0028
RST6	0030
RST7	0038

8) PCHL (jumps to address specified by H-L pair)

Miscellaneous group instruction:

It includes stack operation, input/output operation & interrupt operation instructions.

1) PUSH rp (Push the content of register pair to stack)
2) POP rp (Pop the content of register pair which was saved from the stack)
3) HLT (Halt):
The execution of the instruction HLT stops the microprocessor.
4) IN port address (Input to accumulator from I/O port)
5) OUT port address (Output from accumulator to I/O port)
6) EI (Enable interrupt)
7) DI (Disable interrupt)
8) NOP (No operation)

Addressing modes:-

Each instruction requires certain data on which it has to operate. There are various techniques to specify data for instructions. These techniques are called addressing modes. Intel 8085 uses the following addressing modes:

1) Direct addressing
2) Register addressing
3) Register indirect addressing
4) Immediate addressing
5) Implicit addressing

1) Direct addressing:-

In this mode of addressing, the address of an operand (data) is given in the instruction itself.

Examples are:

i) STA 2400H ii) IN 02H

2) Register addressing:-

In register addressing mode, the operands are in the general purpose registers.

Examples:-

i) MOV A,B ii) ADD B

3) Register indirect addressing:-

In this mode of addressing, the address of the operand is specified by a register pair.

Examples:

i) MOV A,M ii) ADD M

4) Immediate addressing:-

In immediate addressing mode, the operand is specified within the instruction itself.

Examples:

i) MVI A,05H ii) LXI H,2500H iii) ADI 02H

5) Implicit addressing:-

There are certain instructions which operate on the content of accumulator. Such instruction don't require the address of the operand. Examples are: CMA, RAL, RAR etc.

Input / Output Interface

Page
Date

I/O interface :-

Input & Output interfaces provide a method for transfering information between internal storage & external I/O devices. Peripherals connected to a computer need special communication links for interfacing them with microcomputer. The purpose of communication link is to resolve the difference that exist between processor & each peripherals.

The major differences are :-

1) Peripherals are electromechanical & electromagnetic & their manner of operation is different from the operation of microprocessor & memory, which are electronic devices. Therefore, a conversion of signal value may be required.
2) The data transfer rate of peripheral is usually slower than the transfer rate of microprocessor & consequently a synchronization mechanism may be needed.
3) Data code & format in peripheral differ from word format in microprocessor & memory.
4) The operating modes of peripherals are different from each other & each must be controlled so as not to disturb the operation of other peripherals, connected to the microprocessor.

To resolve these differences, computer system

include special hardware components between microprocessor & peripherals to supervise all input & output transfers. These components are called interfacing device or I/O interface.

Microprocessor communicates with peripherals in either of two formats: asynchronous & synchronous. Synchronous means at the same time, transmitter & receiver are synchronized with same clock. Asynchronous means at irregular intervals.

Similarly it transfers data in either of two modes: parallel & serial. In parallel mode, entire word (4-bit, 8-bit or 16-bit) is transferred at once. In serial mode, data are transferred one bit at a time over a single line between microprocessor & peripheral.

There are two types of I/O:

i) I/O mapped I/O (peripheral I/O)

ii) Memory mapped I/O

In I/O mapped I/O, a peripheral is identified with 8-bit address. 8085 has two instructions: IN & OUT to implement data transfer between microprocessor & peripheral.

In memory mapped I/O, a peripheral is connected as if it were a memory location & is identified with 16-bit address

Data transfer is implemented by using memory related instructions such as STA, LDA, MOV M,R & MOV R,M.

Serial communication :-

One bit at a time is transferred over a single line. In serial transmission (from microprocessor to a peripheral), 8-bit parallel word should be converted into a stream of eight serial bits, this is known as parallel to serial conversion. After conversion, one bit a time is transmitted over a single line at a given rate called baud (bits per second). In serial reception, microprocessor receives a stream of 8-bits & they should be converted into 8-bit parallel word, this is known as serial to parallel conversion. Serial communication occurs in either synchronous or asynchronous format.

1) Synchronous Serial communication :-

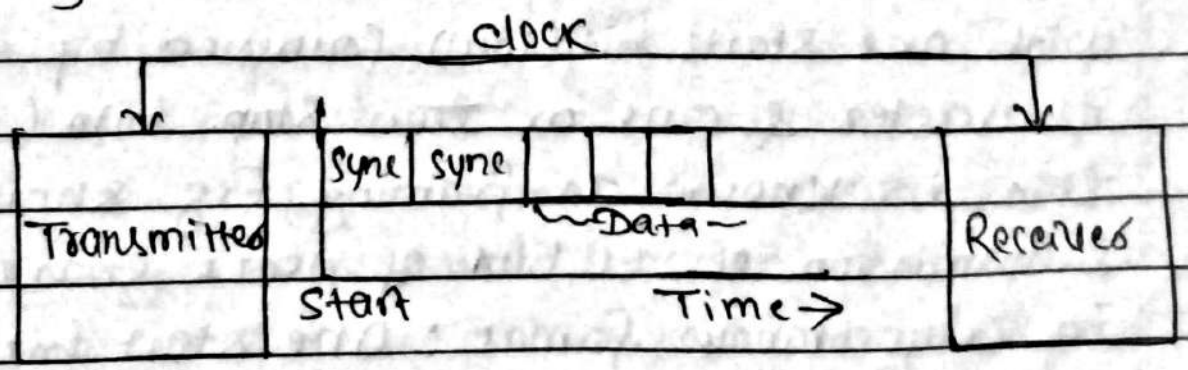


fig: Synchronous serial communication

In synchronous serial communication format, receiver & transmitter are synchronized, a block

of characters are transmitted along with synchronization information as shown in fig. This format is generally used for high speed transmission (more than 20 Kbits/sec).

2) Asynchronous Serial communication :-

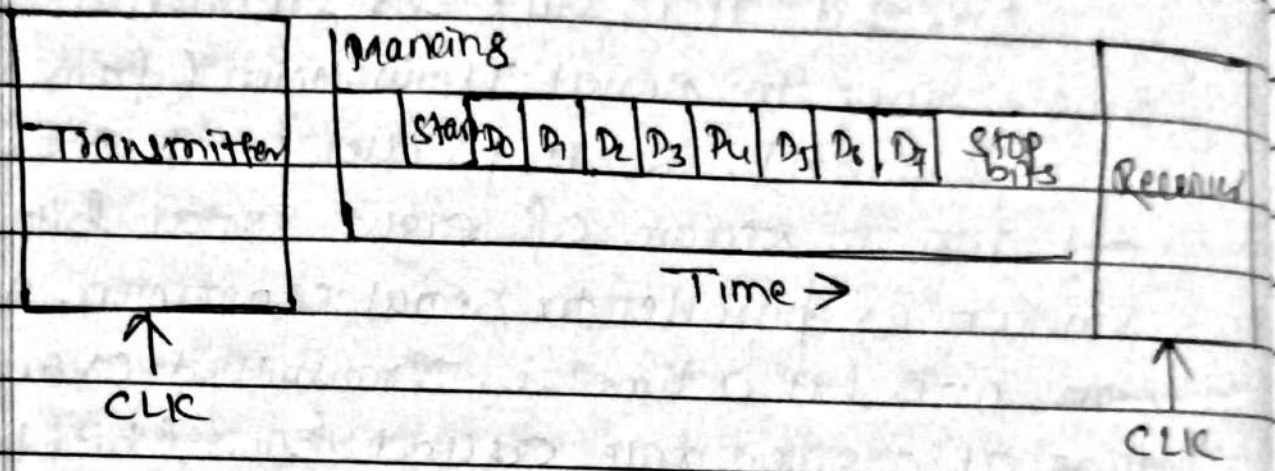


fig: Asynchronous Serial communication

Asynchronous format is character oriented. Each character carries the information of start & stop bits. When no data are being transmitted, receiver stays high at logic 1 called Mark. logic 0 is called space. Transmission begins with one start bit (low) followed by a character & one or two stop bits (high). This is known as framing. Fig shows transmission of 11 bits of ASCII character in asynchronous format : One start bit, eight character bits & two stop bits. This format is generally used in low speed transmission (less than 20 Kbps).

Simplex & Duplex transmission :-

Serial communication also can be classified according to the direction & simultaneity of data flow.

In simplex transmission, data are transmitted in only one direction. E.g. transmission from microcomputer to printer.

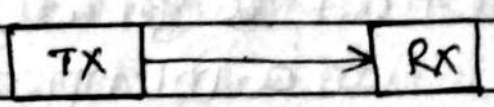


fig: Simplex communication

In duplex transmission, data flow in both direction. However if transmission goes one-way at a time, it is called half duplex communication

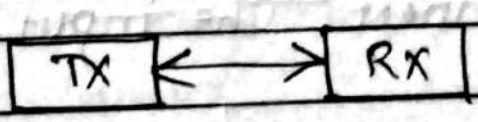


fig: Full duplex communication

If it goes both ways simultaneously, it is called full duplex communication.

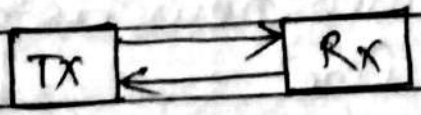


fig: Full duplex communication

E.g. Transmission between two computers

Parallel communication :-

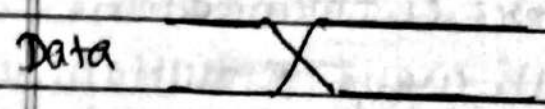
8085 microprocessor transfers eight bits of data simultaneously over eight data lines, this is called parallel mode of data transfer.

Methods of parallel data transfer :-

1) Simple I/O :-

When we need to get digital data from a simple switch into a microprocessor, all we have to do is connect the switch to the input port line & read the port.

Likewise, when we need to output data to a simple display like LED, all we have to do is connect the input of LED on output port.



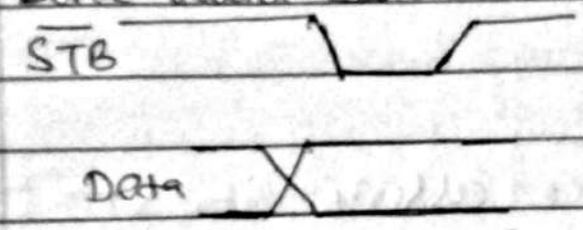
~~Cross~~ Crossed line on waveform represents the time at which new data becomes valid on output line.

2) Simple Strobe I/O :-

In many applications, valid data is present on external device only at a certain time, so it must be read in that time.

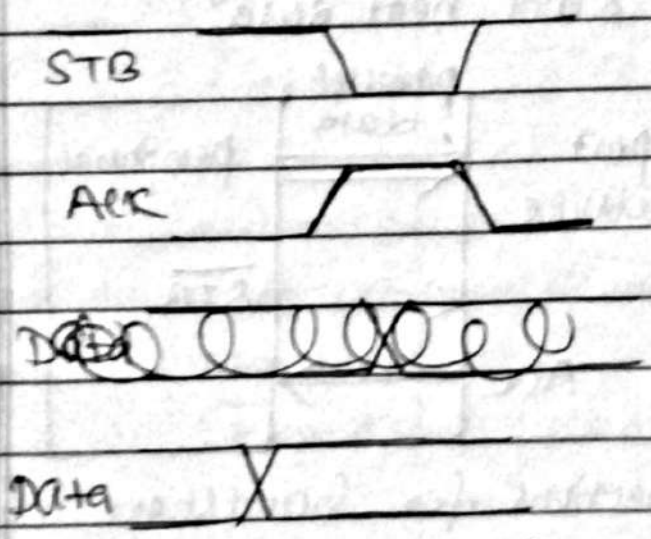
When a key is pressed, circuitry on the keyboard sends out ASCII code for pressed

key on 8 parallel data lines. ~~to~~ and then send out a strobe signal on another line to indicate the valid data is present on 8 data lines.



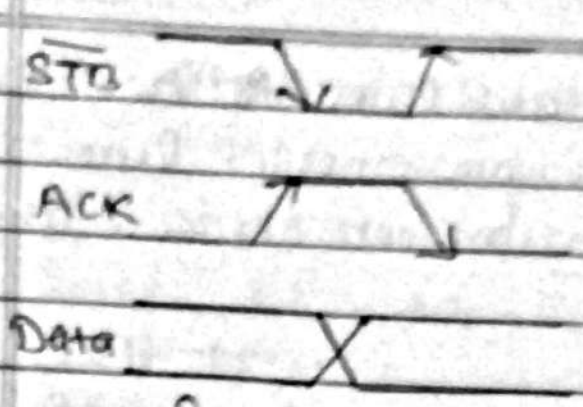
3) Single Handshake I/O:-

Peripheral outputs some parallel data & sends $\overline{STB}$ signal from microprocessor. Microprocessor detects $\overline{STB}$ signal & read data. Then microprocessor sends acknowledge signal (ACK) to the peripheral to indicate that data has been read & peripheral can send next byte of data.



4) Double Handshake I/O:-

For data transfers when even more co-ordination is required between sending & receiving system, double handshake is used.



Sending device asserts its STB line low to ask "Are you ready?", receiving device raises its Ack line high to say "I am Ready". Peripheral then sends data & raises its STB line high to say "Here is some valid data for you". After it has read data, receiving system drops its Ack line low to say "I have & I await your request to send next data".

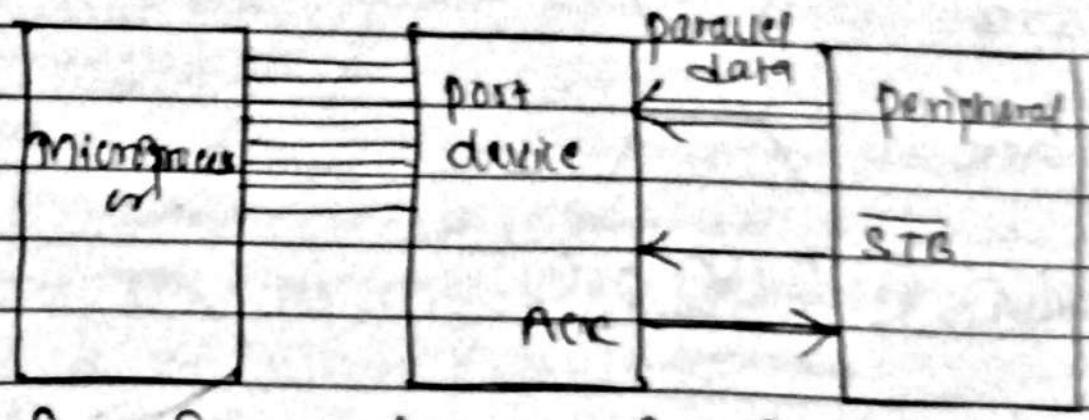


fig: Signal directions for handshake input data transfer

8255 programmable peripheral interface:-

8255 is widely used, programmable, parallel I/O device. It can be programmed to transfer data under various conditions, from simple I/O to interrupt I/O. It is important, general purpose I/O device that can be used with almost any microprocessor.

8255 has 24 I/O pins that can be grouped primarily in two 8-bit parallel ports: A & B, with the remaining eight bits as port C. The eight bits of port C can be used as individual bits or be grouped in two 4-bit ports: C_{upper} (C_U) & C_{lower} (C_L).

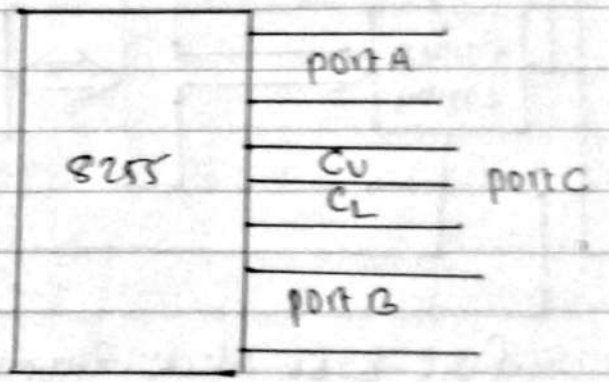


fig: 8255 I/O ports

The function of these ports are defined by writing a control word in the control register.

The block diagram shown two 8 bit ports (A & B), two 4-bit ports (C_U & C_L), data bus buffer & control logic.

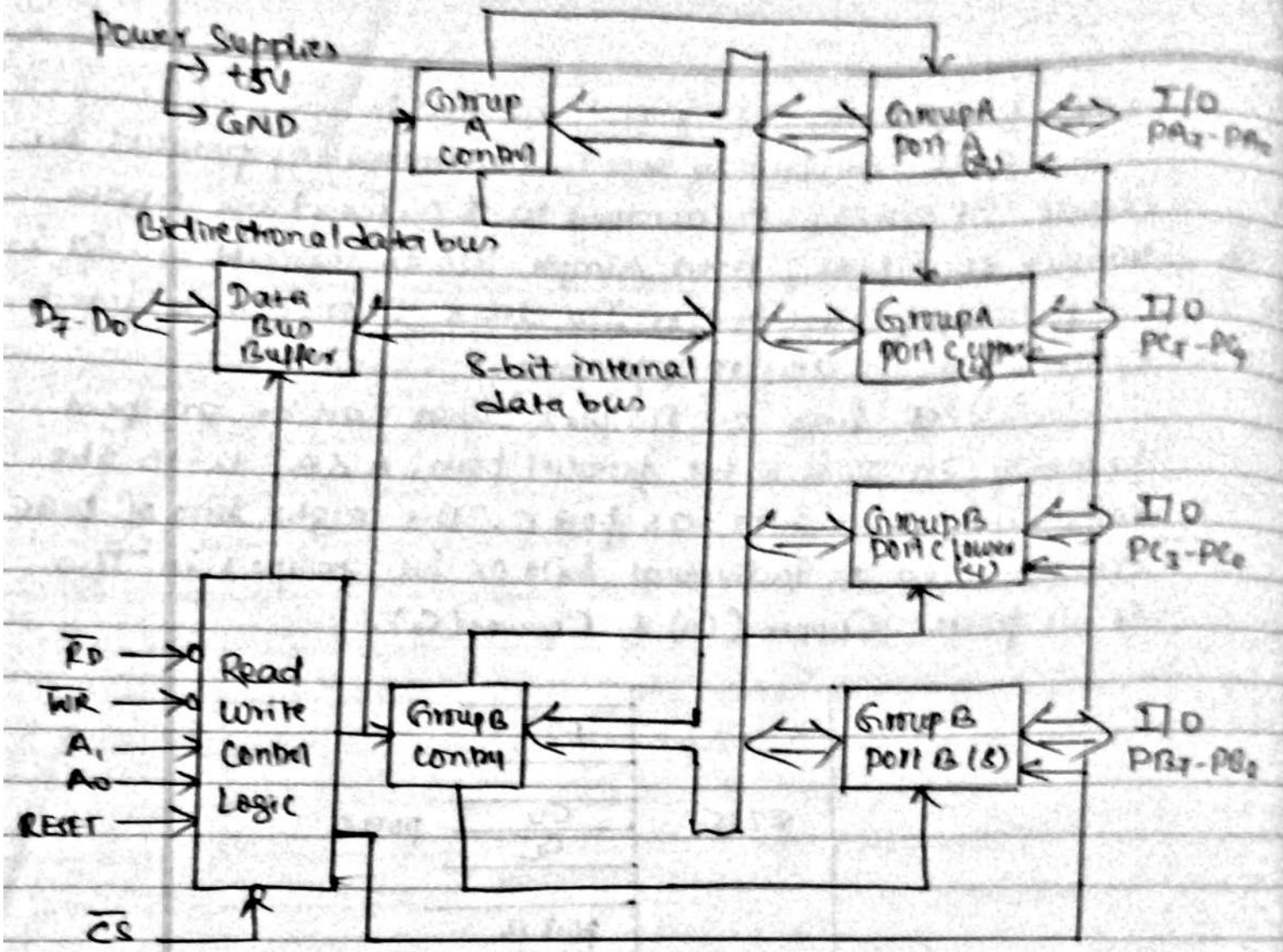


fig: 8255 block diagram

Control logic:-

It has six lines.

$\overline{RD}$ (Read):-

This control signal enables Read operation. When the signal is low, microprocessor reads data from selected I/O port of 8255.

$\overline{WR}$ (write):-

This control signal enables write operation. When the signal goes low, microprocessor writes into selected I/O port or control register.

RESET:-

It clears the control registers & sets all ports in

the input mode.

~~$\overline{CS}$, A_0, A_1:~~

$\overline{CS}$:-

It is chip select signal & connected to decoded address.

A_0 & A_1:-

A_0 & A_1 specify one of I/O ports or control register as given below:-

A_1	A_0	Selected
0	0	PORT A
0	1	PORT B
1	0	PORT C
1	1	control register.

Operation mode of 8255:-

Mode 0:-

In this mode, ports A & B are used as two simple 8-bit I/O ports & port C as two 4-bit ports. Each port can be programmed to function as simply an input port or an output port.

Mode 1: Input or output with handshake

Two ports (A & B) function as 8-bit I/O ports. They can be configured either as input or output ports. Each port uses three lines from port C as handshake signals.

Mode 2: Bidirectional data transfer

In this mode, port A can be configured as bidirectional port & port B either in mode 0 or mode 1. PORT A uses five signals from port C as handshake signals for data transfer.

BSR (Bit set / reset) mode:-

It is concerned with eight bits of port C, which can be set or reset.

Control word for I/O mode:-

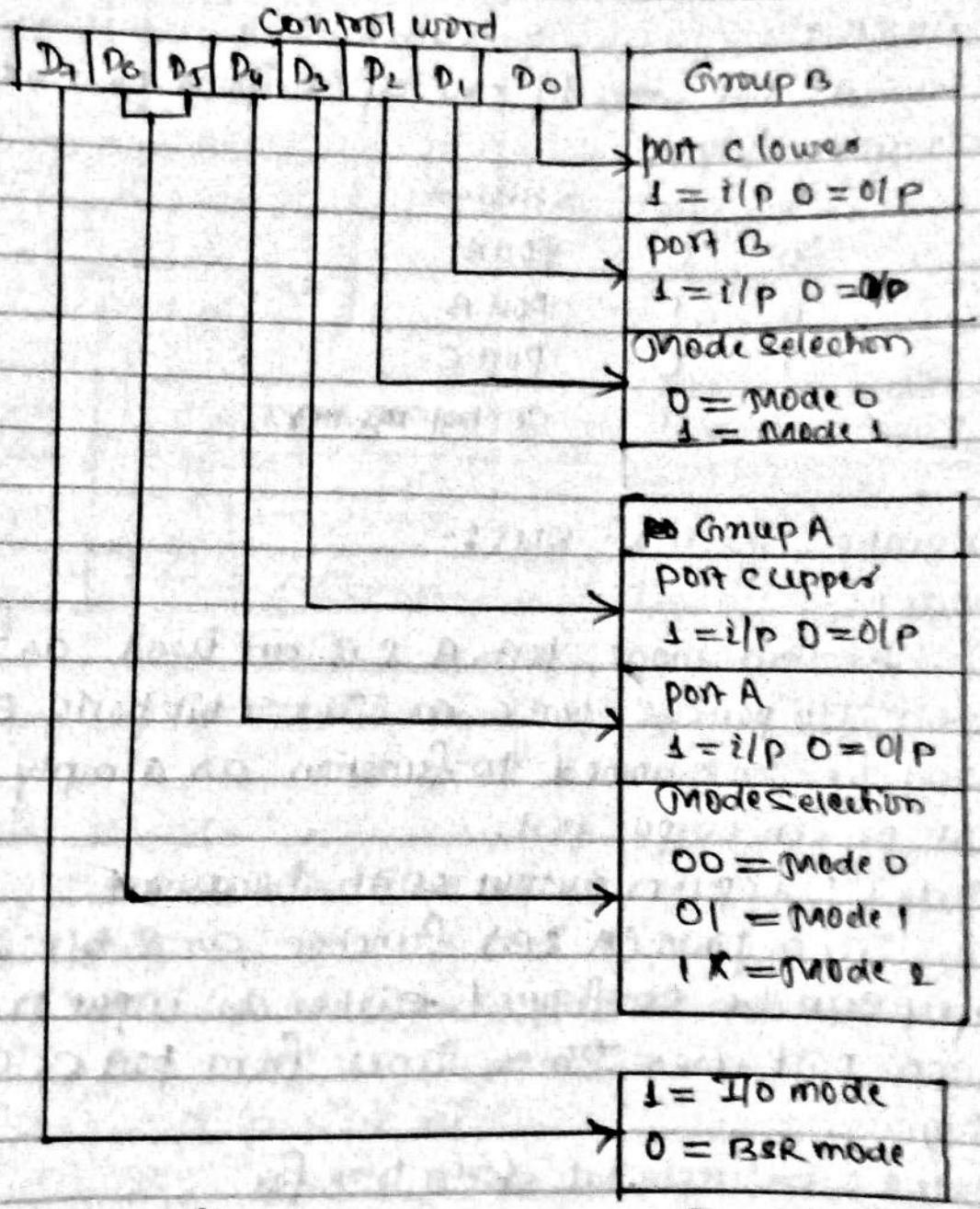
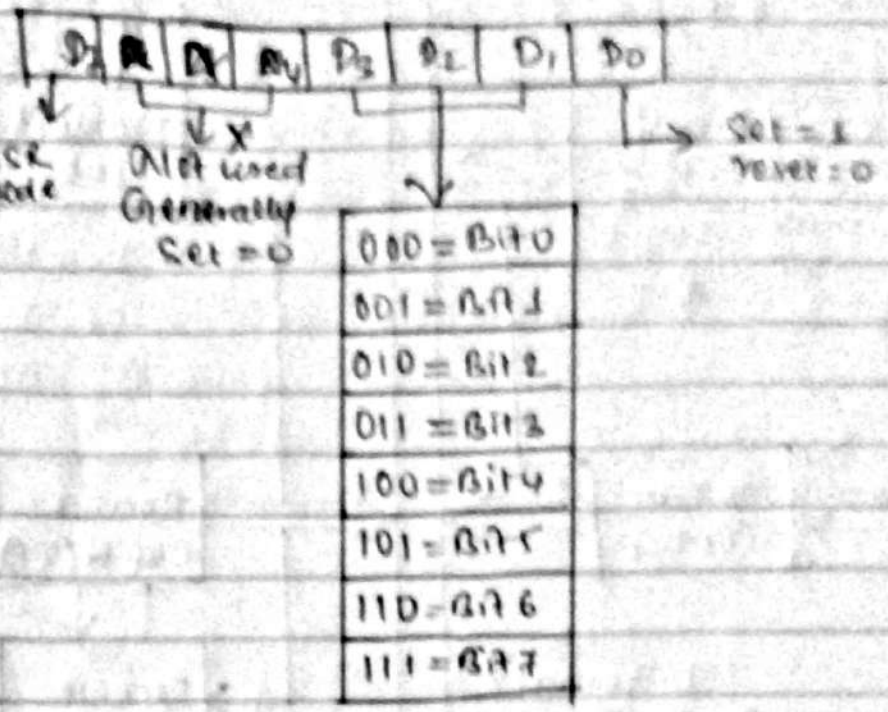


fig: 8255 control word for I/O mode

Control word for BSR mode:-



8251 programmable communication interface:

It is programmable chip designed for synchronous & asynchronous serial data communication, packaged in 28 pin DIP. It includes three sections: Read/write control logic, transmitter, receiver, data bus buffer & modem control.

R/W control logic & registers:-

This section includes R/W control logic, six input signals, control logic & three buffer registers: data register, control register & status register.

Input signals to control logic are:

$\overline{CS}$ - Chip select:-

when this signal goes low, 8251 is selected by microprocessor for communication.

$C/\overline{D}$ - control/Data:

when this signal is high, control register or status register is addressed, when it is low, data buffer is addressed.

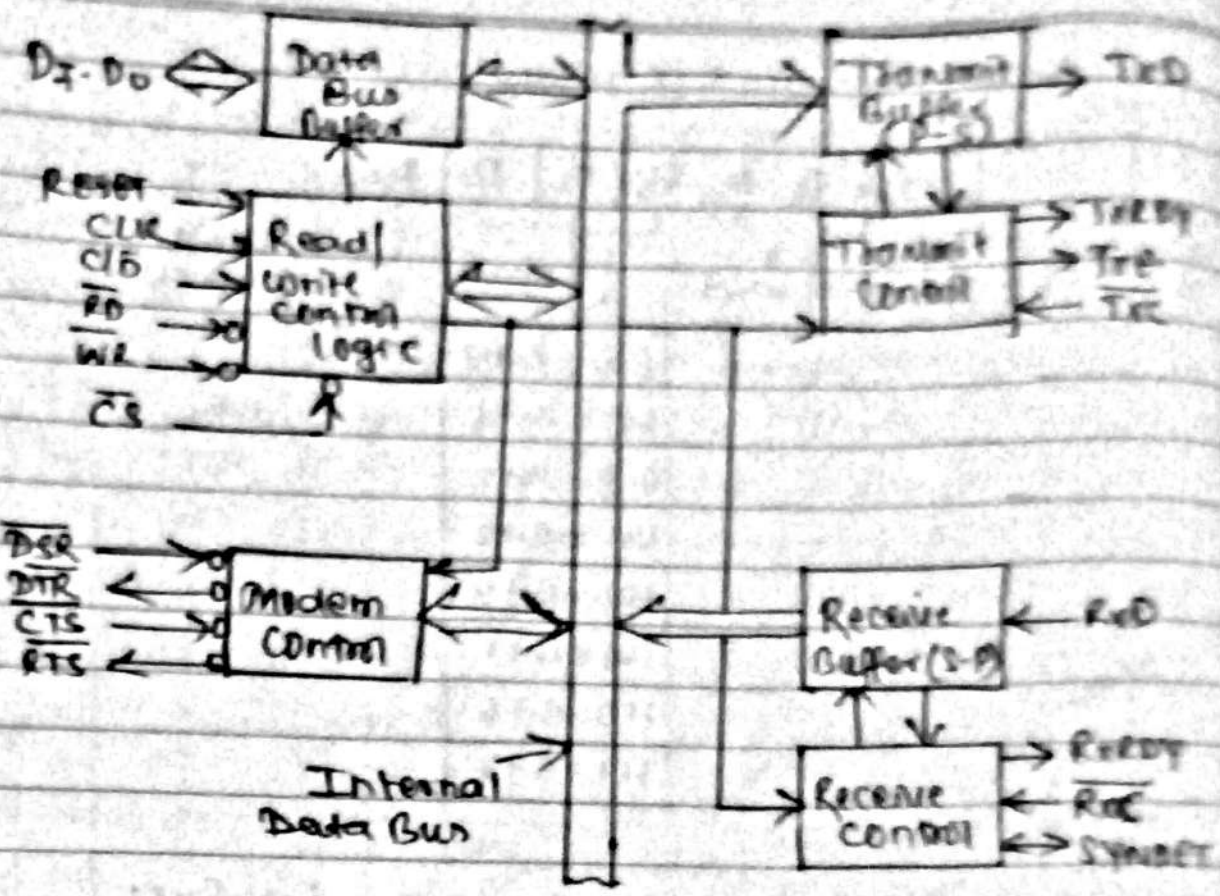


fig: Block diagram of 8251

$\overline{WR}$ – write :-

When this signal goes low, microprocessor either writes in control register or sends output to data buffer.

$\overline{RD}$ - Read :-

When this signal goes low, microprocessor either reads status from status register or accepts data from data buffer.

RESET :-

A high on this input resets 8251 & forces it into idle mode.

CLK - Clock :

This is clock input, usually connected to system clock.

Transmitter Section:-

Transmitter accepts parallel data from microprocessor & converts them into serial data. It has two registers: buffer register to hold eight bits & output register to convert eight bits into stream of serial bits.

TxD – Transmit data:

Serial bits are transmitted on this line.

$\overline{TXC}$ – Transmitter clock:

This input signal controls the rate at which bits are transmitted by USART. The clock frequency can be 1, 16 or 64 times the baud.

TxRDY – Transmitter Ready:-

This is output signal. When it is high, it indicates that buffer register is empty & USART is ready to accept a byte.

TXE – Transmitter Empty:

This is output signal. Logic 1 on this line indicates that the output register is empty.

Receiver Section:-

The receiver accepts serial data on RxD line from peripheral & converts them into parallel data. The section has two registers: receiver input register & buffer register.

RxD – Receive data:

Bits are received serially on this line.

$\overline{RXC}$ – Receiver clock:

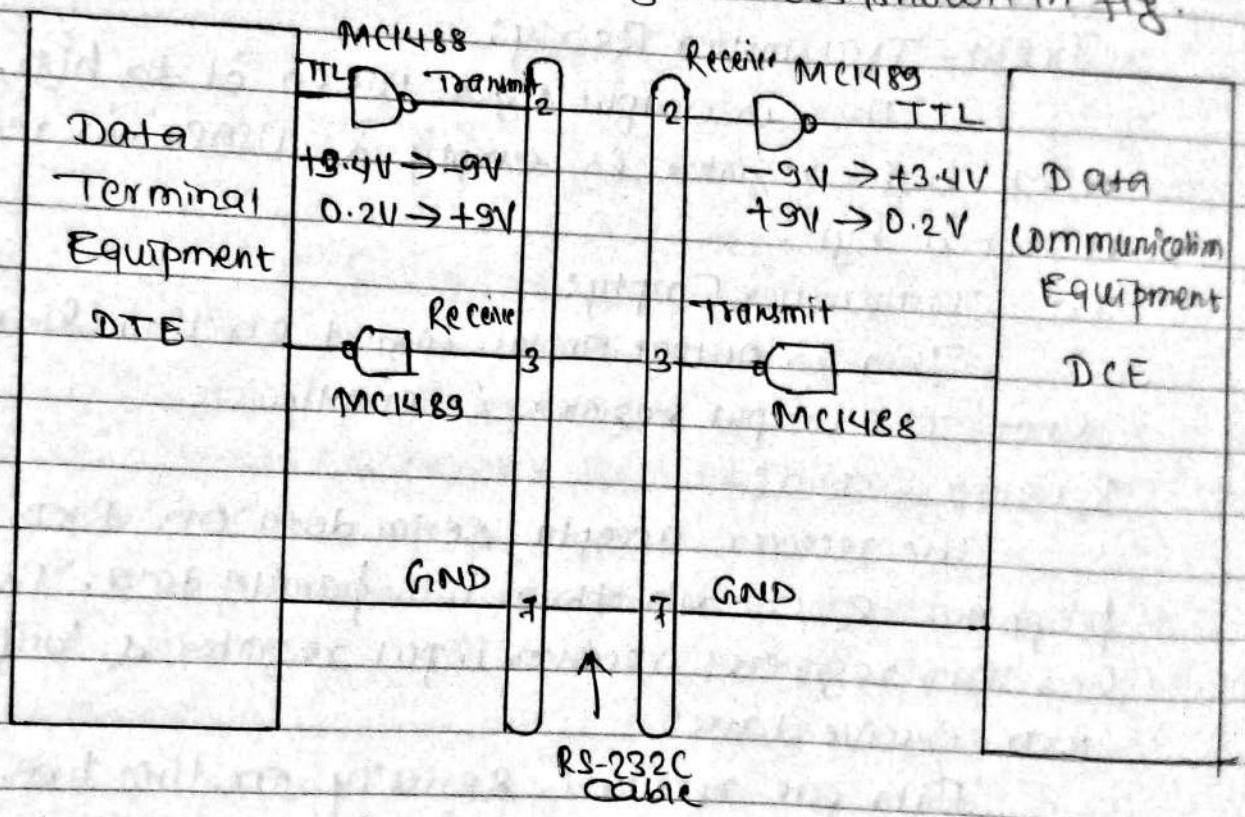
This is clock signal that controls the rate at which bits are received by USART.

RXRDY – Receiver Ready:

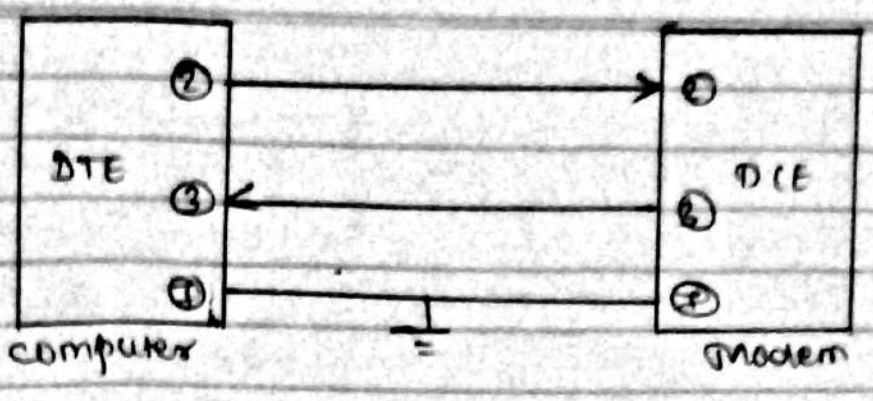
This is output signal. It goes high when USART has character in buffer register & is ready to transfer it to the microprocessor.

RS232C :-

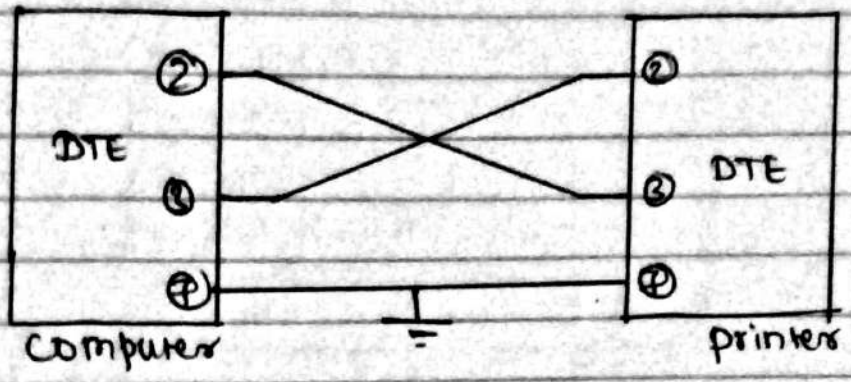
It is 25-pin connector & its signals are divided into four groups: data signals, control signals, timing signals & grounds. For data lines, voltage level +3V to +15V is defined as logic 0, from -3V to -15V is defined as logic 1. This is negative true logic. However, other signals (control & timing) are compatible with TTL level. Because of incompatibility of data lines with TTL logic, voltage translators called line drivers & line receivers are required to interface TTL logic with RS232 signals as shown in fig.



This interface standard is most widely used standard for serial communication between microprocessor & peripheral devices. This interface relates essentially two types of equipments. First is called DTE which is capable of sending or receiving data via serial interface while the second is called DCE which can facilitate serial data communication.



fig(a): DTE to DCE connection



fig(b): DTE to DTE connection (Null modem connection)

fig: RS232 connections

Interrupt:-

It is a signal that a peripheral sends to the microprocessor in order to request attention. In response to an interrupt, microprocessor stops what is currently doing & executes a service routine.

Interrupt I/O is a process of data transfer where by an external device can inform the processor that it is ready for communication & it requests attention.

Page
Date

Basic interrupt processing:-

Step1:

The interrupt process should be enabled by writing the instruction EI in the main program. The instruction EI sets interrupt enable flipflop. The instruction DI resets the flipflop & disables the interrupt process.

Step2:

When the microprocessor is executing a program, it checks INTR line during execution of each instruction.

Step3:

If the line INTR is high & interrupt is enabled, the microprocessor completes the current instruction, disables the interrupt enable flipflop & sends a signal called $\overline{INTA}$. The processor can not accept any interrupt request until the interrupt flipflop is enabled again.

Step4:

The signal $\overline{INTA}$ is used to insert restart (RST) instruction through external hardware.

Step5:

When the microprocessor receives RST instruction, it saves the memory address of next instruction on the stack. The program is transferred to the call location.

Step 6:

Assuming that the task to be performed is written as subroutine at specified location, the processor performs the task. This subroutine is known as service routine.

Step 7:

The service routine should include instruction EI to enable the interrupt agn.

Step 8:

At the end of subroutine, RET instruction retrives the memory address where the program was interrupted & continues the execution.

Types of interrupt:-

1) Hardware interrupt
   i) External interrupt
   ii) Internal interrupt
2) Software interrupt

External interrupt:-

External interrupts are initiated via microprocessor interrupt pins by the external devices.

It can be further divided into two types:

i) Maskable interrupt
ii) Non-maskable interrupt

Maskable interrupt :-

Microprocessor can ignore or delay maskable interrupt request if it is performing some critical task. E.g. RST 7.5, RST 6.5, RST 5.5, INTR

Non-maskable interrupt :-

Microprocessor can't ignore the non-maskable interrupt. E.g. TRAP

Internal interrupt :-

It arise from erroneous use of instruction. The cause of this interrupt may be register overflow, attempt to divide by zero.

Software interrupt :-

It is initiated by executing an instruction. In 8085, RST0, RST1, --- RST7 instruction causes software interrupt.

Interrupts of 8085 :-

8085 has five interrupts.

TRAP (highest priority)
RST 7.5
RST 6.5
RST 5.5
INTR (lowest priority)

Interrupt	Call location
TRAP	0024H
RST7.5	003CH
RST6.5	0034H
RST5.5	002CH

Page
Date

Interrupt priority:-

It is a system that established a priority over various sources to determine which is to be serviced first when two or more request arrive simultaneously. There are mainly two ways of servicing multiple interrupts. These are:

1) Polled interrupt 2) Chained (vectored) interrupt

Polled interrupt:-

These interrupts are handled by using software & are therefore slower compared to vectored (hardware) interrupts. In this method, there is one common branch address for all interrupts. The program that takes care of interrupts begins at the branch address & polls the interrupt source in sequence. The order in which they are tested determines the priority of each interrupt. The highest priority source is tested first & if its interrupt signal is on, control branches to a service routine for this source. Otherwise the next lower priority source is tested & so on. Thus, the initial service routine for all interrupts consist of a program that test the interrupt sources in sequence & branches to one of many possible service routines

Polled interrupts are very simple. But for larger number of devices, the time required to poll each device may exceed the time to service the device. In such case, faster mechanism called chained interrupt is used.

Chained interrupts-

This is hardware concept of handling the multiple interrupts. In this technique, the devices are connected in a chain fashion as shown in fig for setting up the priority system.

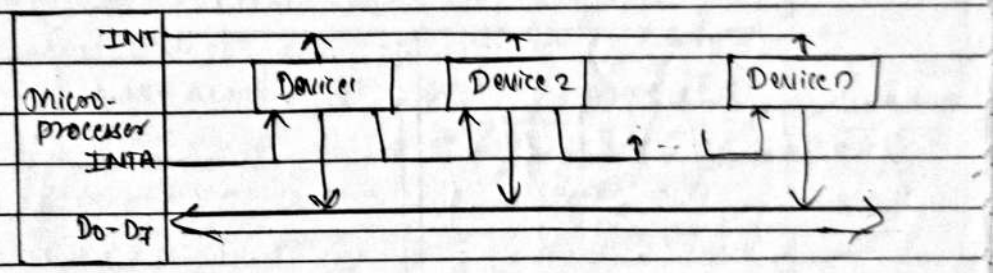


fig: Chained interrupt

The device with the highest priority is placed in the first position, followed by lower priority devices. Suppose that one or more devices interrupt the processor at a time. In response, the processor saves its current status & then generate interrupt acknowledge ($\overline{INTA}$) signal to the highest priority device, which is device 1 in this case. If this device has generated the interrupt, it will accept $\overline{INTA}$ signal from the processor otherwise it will

Page
Date

pass $\overline{INTA}$ on to next device until $\overline{INTA}$ is accepted by the interrupting device.

Once accepted, the device provides a means to the processor for finding the interrupt address vector using external hardware. Usually requesting device responds by placing a word on the data lines. With the help of hardware, it generates interrupt vector address.

Page
Date

Memory Interfacing & Generation of chip select Signal :-

Memory structure :-

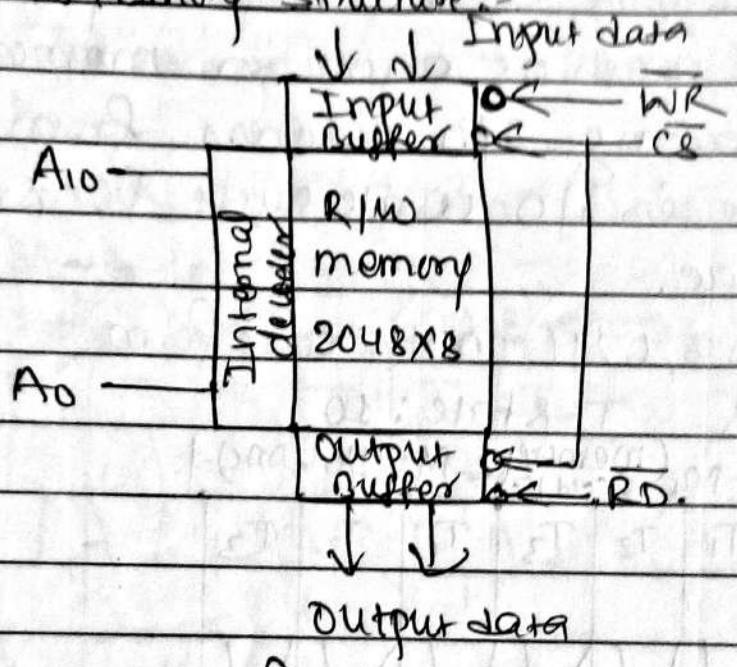


fig (a): R/W memory

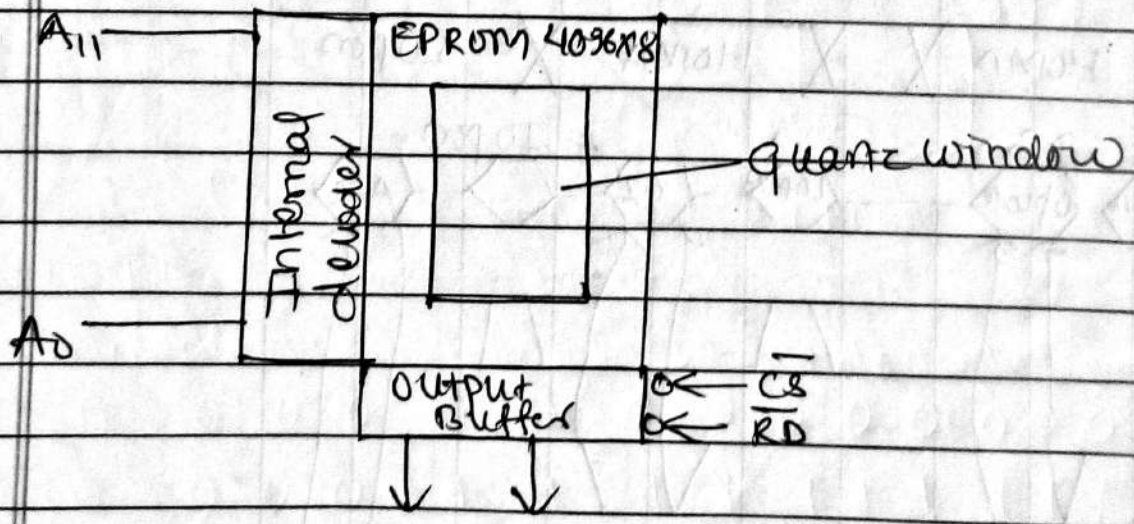


fig (b): EPROM

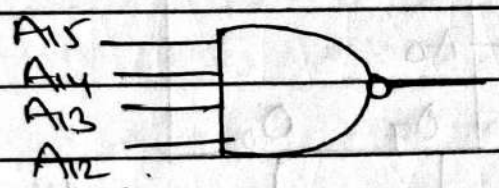
Page
Date

Address decoding:-

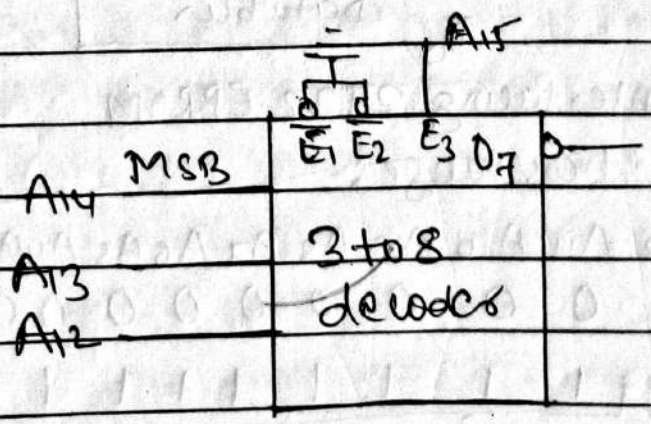
The process of address decoding should result in identifying a register for given address. We should be able to generate a unique pulse for a given address. E.g. in fig(b), 12 address lines (A_{11}-A_0) are connected to memory chip & remaining 4 address lines (A_{15}-A_{12}) of 8085 microprocessor must be decoded.

Two methods of decoding these lines:

One by using NAND gate & other by using 3 to 8 decoder.



(a) NAND gate



(b) 3 to 8 decoder

Output of NAND goes ~~high~~ active & selects chip only when all address lines A_{15}-A_{12} are at logic 1.

We can obtain same result by using D_7

of 3 to 8 decoder, which is capable of decoding eight different input address.

Memory interfacing:-

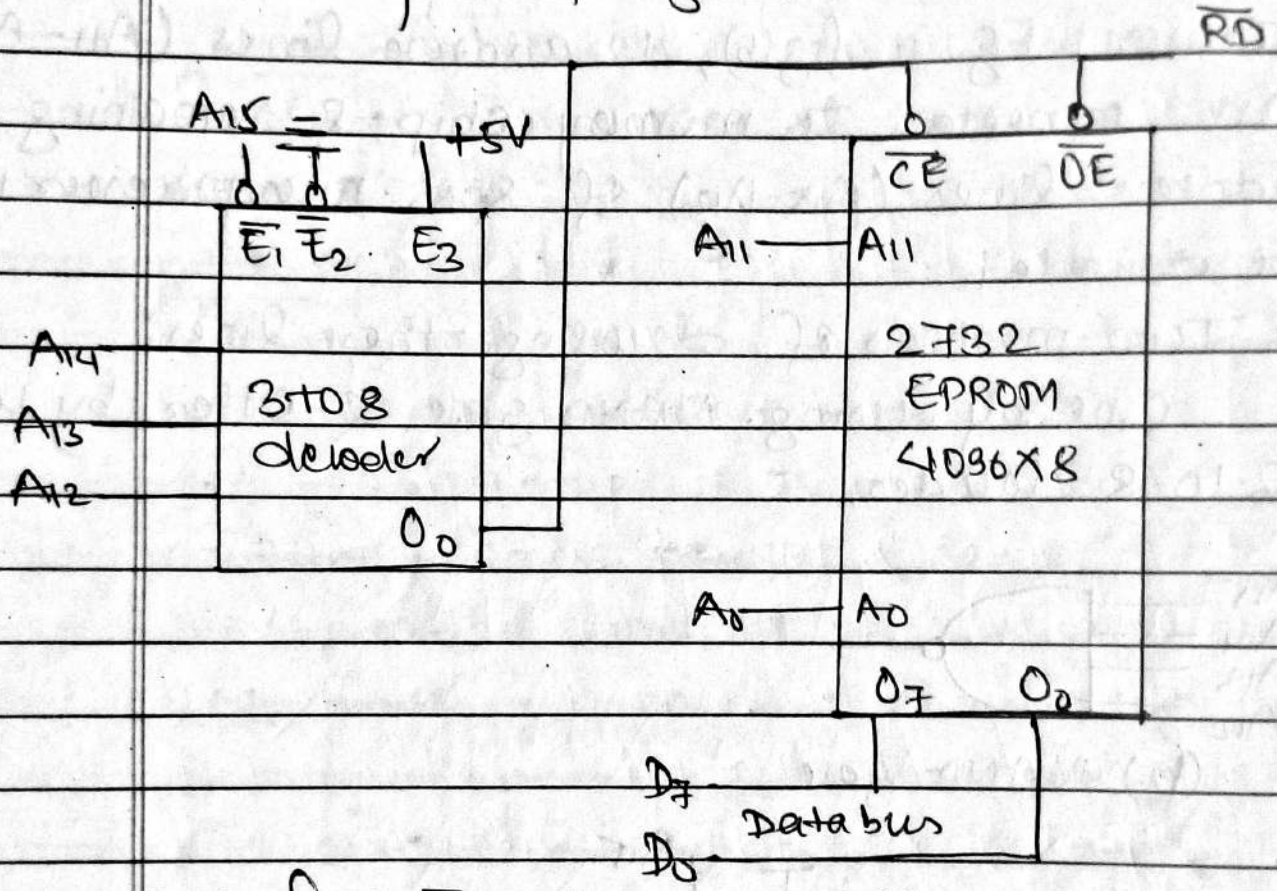


fig: Interfacing 2732 EPROM

Memory address range:-

A15	A14	A13	A12	A11	A10	A9	A8	A7	A6	A5	A4	A3	A2	A1	A0	
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	= 0000H
0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	= 0FFFH

Chip enable — Register enable