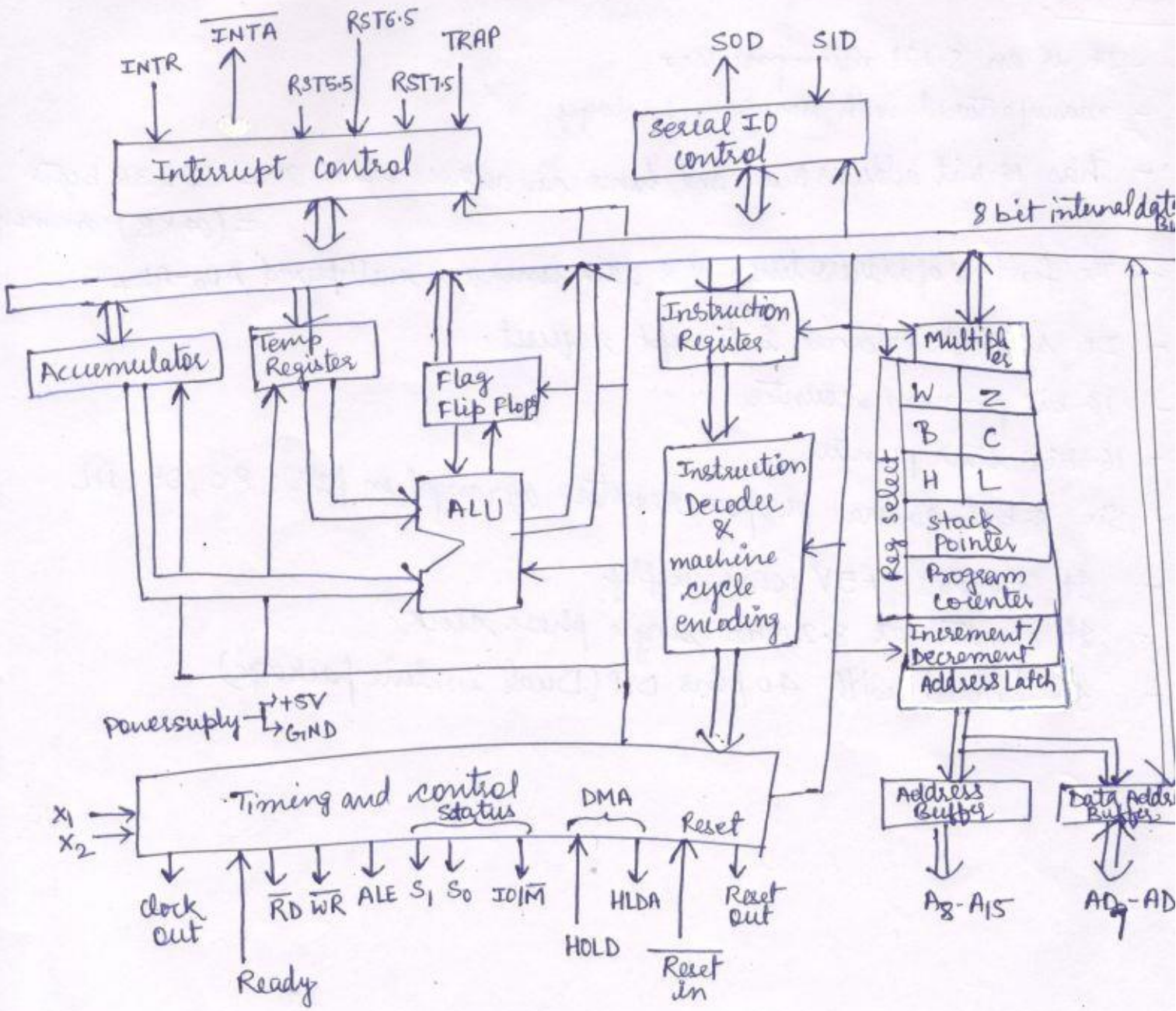
Intel 8085 Microprocessor:

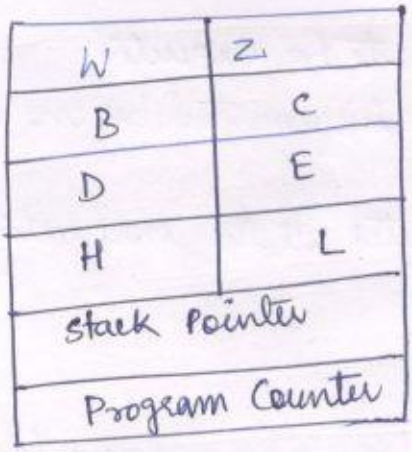
Features:

- It is an 8-bit microprocessor
- manufactured with NMOS technology
- has 16-bit address bus and hence can address upto 2^{16} = 65536 bytes = (64 KB) memo
- The first 8 of address lines & 8 data lines are multiplexed AD_0-AD_7
- It supports external interrupt request.
- 16-bit program counter
- 16-bit stack pointer
- Six 8-bit general purpose registers arranged in pair: BC, DE, HL
- It requires +5V power supply.
- It operates at 3.2 MHz single phase clock.
- It enclosed with 40 pins DIP (Dual in line package).

Block Diagram or Architecture of 8085 Microprocessor:



Register Organization:



There are two types of registers available in 8085 microprocessor.

- General Purpose Registers (GPR)
- Special Purpose Registers (or special Function Registers)

General Purpose Registers:

→ There are 6 general purpose registers

→ B, C, D, E, H and L.

→ size of all the general-purpose registers is 8-bit

→ they can be used in pairs for 16 bit information like BC, DE and HL

B, D, H are higher Order registers
C, E, L are lower Order registers

* from the above pairs only HL register is used for memory accessing.

Special Function Registers:

Accumulator

→ size is 8 bit

→ considered as the part of ALU.

→ One of the operands must be stored in it during Arithmetic and logic operation.

→ After operation, the result is stored in this register.

Program counter:

→ It is a 16-bit register ~~used to~~
→ used to store the address of next instruction to be executed
→ The content of this register is automatically incremented by one after each byte is fetched.
→ Because of this register the microprocessor executes all the instruction sequentially.

Stack:

- A stack is the group of memory location defined in the R/w memory (RAM) which is used to store the data temporarily during program execution.
- It is always defined by the user. Stack is always works on the principle of LIFO (Last In First Out)
- So in stack, data can be stored by decrementing the address and can be retrieved by incrementing the address.

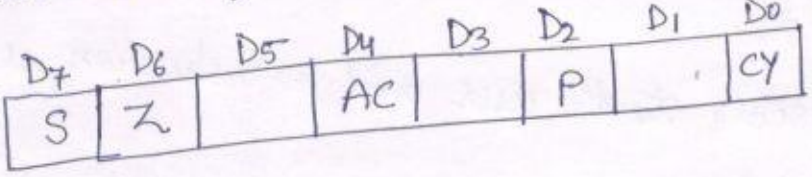
→ The stack is always accessed in 16 bit operation.

Stack Pointer:

- The stack pointer is the 16 bit register which store the starting address of stack
- So the data stored into the stack by decrementing the content of the SP and retrieve by incrementing the content of the SP.
- It is always advisable to the user to define the stack in the higher end of the memory to avoid the overlapping of between the main program and stack.
- The SP is initialized by the instruction LXI SP, 16 bit address.
- The stack can access be accessed by using the two instructions like PUSH and POP.

Flags:

- This is the 8-bit register and used to reflect the condition of the result of any arithmetic and logical operation.
- Among 8 flip flops only 5 flip flops are used.
- Due to this flag register MP take the decision to change the sequence of program execution.
- This flags depend on the content of accumulator because result is stored in accumulator after each operation.



Carry Flag:

- After each operation, if there is a carry result, then this flag is set otherwise it is reset

CY ⇒ 1 ⇒ carry generated

CY ⇒ 0 ⇒ no carry

Parity Flag:

If the number of 1's present in the result is even, then this flag is set otherwise reset.

P ⇒ 1 ⇒ number of 1's is even.

P = 0 ⇒ number of 1's is odd.

Zero Flag:

If accumulator contains 00H (or 00000000) then this flag is set. Otherwise it will be reset.

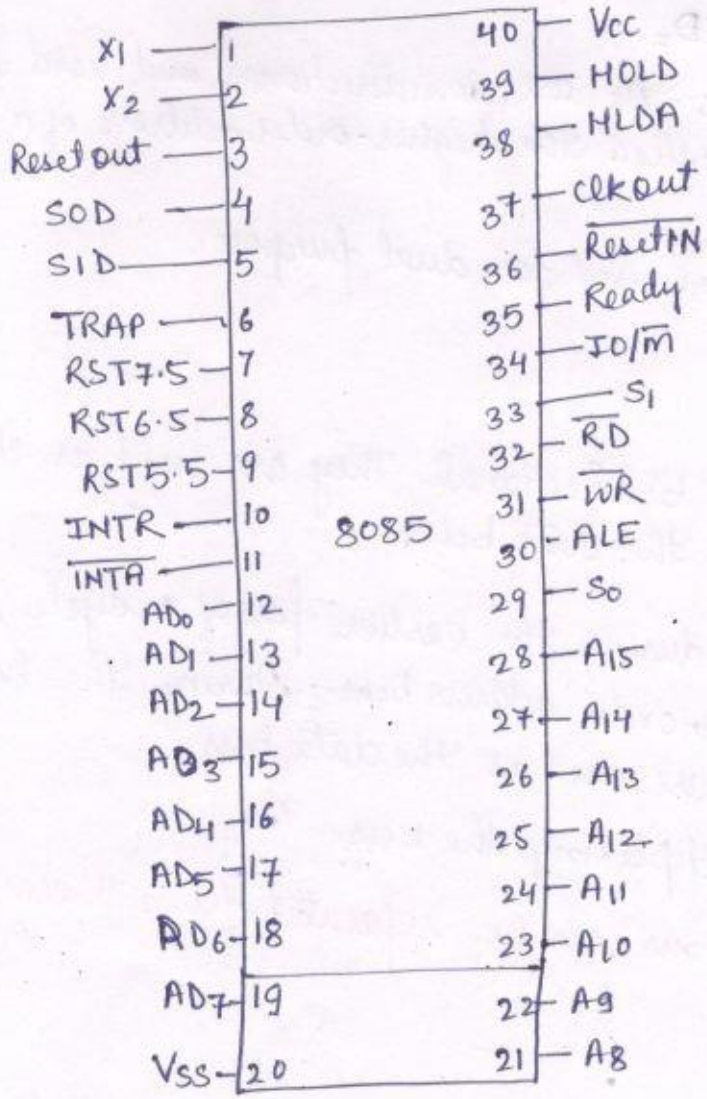
Sign Flag:

- if the MSB (D7 bit) in the accumulator is 1 after any A/L operation then this flag is set otherwise reset.
- MSB is used to represent the sign of a number i.e if the MSB of a numb is 1 then that number is a negative number ~~so~~ so this flag depends on the MSB of accumulator.
- In sign number manipulation, among 8 bits only 7 bits are used for the magnitude and last bit is used for sign.

Auxillary Flag:

- This flag is not accesible to the user and no instruction is available using this flag.
- This flag is used in BCD operation.
- In any arithmetic (BCD) operation when there will be a carry result from D_3 bit to D_4 bit of operand then this flag is set otherwise it is reset.

Pin diagram of 8085 Microprocessor



8085 Pinout

In the logic pinout of 8085 microprocessor, all signals can be classified into six groups.

1. Address bus
2. Data bus
3. control and status
4. power supply and frequency signals
5. externally initiated signals
6. serial I/O ports.

→ Address Bus:

- The 8085 has 16 signals lines (pins) that are used as address bus.
- These 16 lines are split into two segments
  - $A_8 - A_{15}$
  - $AD_0 - AD_7$
- The eight signal lines $A_{15} - A_8$ are unidirectional and used for the the most significant bits, called the higher-order address of a 16 bit address.
- The signal lines $AD_7 - AD_0$ are used for dual purpose.

Multiplexed Address / Data Bus:

- The signal lines $AD_7 - AD_0$ are bidirectional. They are used as the lower address bus as well as the data bus.
- In executing an instruction, during the earlier part of a cycle, these lines are used as the lower-order address bus. During the later part of cycle, these lines are used as the data bus.
- This is also known as multiplexing the bus.
- However, the lower address bus can be separated from these signals by using a latch

control and status signal:

This group of signals include two control signals $\overline{RD}$ and $\overline{WR}$, three status signals ($IO/\overline{M}$, S1 and S0) to identify the nature of operation, and one special signal ALE (Address Latch Enable) to indicate the beginning of the operation.

ALE: (Address Latch Enable): This is a positive going pulse generated everytime the 8085 begins an operation (machine cycle); it indicates that the bits on $AD_7 - AD_0$ are address bits.

This signal is used primarily to latch the lower order address from the ~~multiplexing~~ multiplexed bus and generate a separate set of eight address lines $A_7 - A_0$.

$\overline{RD}$ (Read): (Active Low) This is read control signal. It indicates that the selected I/O or memory device is to be read and data are available on the data bus.

$\overline{WR}$ (Write): (Active Low): This a write control signal. This signal indicates that the data on the data bus are to be written into a selected memory or I/O location.

IO/$\overline{M}$: This is a status signal used to differentiate between I/O and memory ~~locations~~ operations.
when it is high ⇒ it indicates an I/O operation
when it is low ⇒ it indicates a memory operation.

S_1 and S_0: These status signals can identify various operations.

machine cycle status and control signals

machine cycle	IO/$\overline{M}$	S_1	S_0	control signals
Opcode Fetch	0	1	1	$\overline{RD} = 0$
Memory Read	0	1	0	$\overline{RD} = 0$
Memory Write	0	0	1	$\overline{WR} = 0$
I/O Read	1	1	0	$\overline{RD} = 0$
I/O Write	1	0	1	$\overline{WR} = 0$
Interrupt Acknowledge	1	1	1	$\overline{INTA} = 0$
Halt	Z	0	0	$\overline{RD}, \overline{WR} = Z$ and $\overline{INTA} = 1$
Hold	Z	X	X	
Reset	Z	X	X	

Z = Tri state
X ⇒ unspecified or Don't ca

Power supply and clock Frequency:

V_{CC}: +5 V power supply.

V_{SS}: Ground Reference.

X_1 and X_2: A crystal clock is connected to these pins. The frequency is internally divided by two! Therefore, to operate a system at 3 MHz, the crystal clock should have a frequency at 6 MHz.

CLK out: Clock output: This signal can be used as the system clock for other devices.

Externally initiated signals, including interrupts:

→ The 8085 has 5 interrupts signals that can be used to interrupt a program execution. (INTR, RST7.5, RST6.5, RST5.5 and TRAP)

→ In addition to the interrupts, three pins Reset, HOLD and READY - accept the externally initiated signals.

8085 interrupts and Externally initiated signals:

→ INTR: Interrupt Request. This is used as a general purpose interrupt.

→ $\overline{\text{INTA}}$: Interrupt Acknowledge. This is used to acknowledge an interrupt.

→ RST7.5, RST6.5, RST5.5 ⇒ Restart Interrupts. These are vectored interrupts that transfer the program control to specific memory locations. They have higher priority than INTR. Among these three, priority order is 7.5, 6.5 and 5.5.

- TRAP: This is non-maskable interrupt and has the highest priority

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

- HLDA: Hold Acknowledge. This signal acknowledges the HOLD request

- READY: This signal is used to delay the microprocessor cycles until a slow responding peripheral is ready to send or accept data. When this signal goes low, the microprocessor waits for an integral number of ~~cycles~~ clock cycles until it goes high.

RESET IN: when the signal on this pin goes low, the program counter is set to zero, the buses are tristated and the MPU is reset.

Reset OUT: This signal indicates that MPU is being reset. This signal can be used to reset other devices.

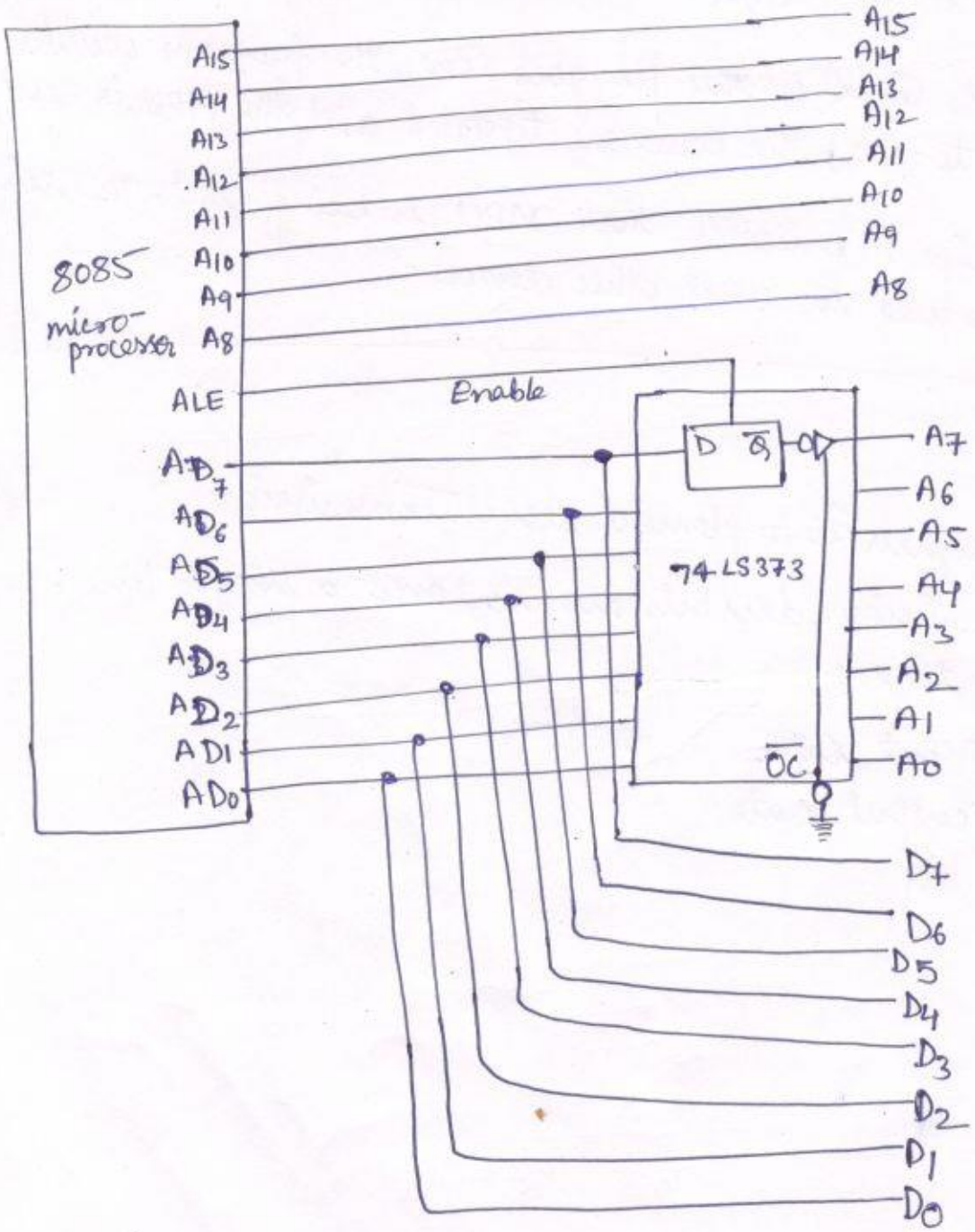
Serial I/O Ports:

- 8085 has two signals to implement serial transmission
- In serial transmission, data bits are sent over a single line. one bit at a time.
- SID: Serial Input Data
- SOD: Serial output Data.

Demultiplexing the bus $AD_7 - AD_0$

→ The need of demultiplexing: The address on the higher order bus remains on the bus for three clock periods. However, the lower address is lost after the first clock period. This address needs to be latched and used for identifying the memory address

→ Schematic of latching Low order address bus:



- The ALE signal is connected to the Enable (G_1) pin of the latch and output control($\overline{OC}$) signal of the latch is grounded..

- ALE goes high during T_1. when the ALE is high, the latch is transparent; this means that the output changes according to the input.

→ when the ALE goes low, the address is latched until the next ALE, and the output of latch represents the low-order address bus A_7-A_0 after latching operation.

Generating control signals:

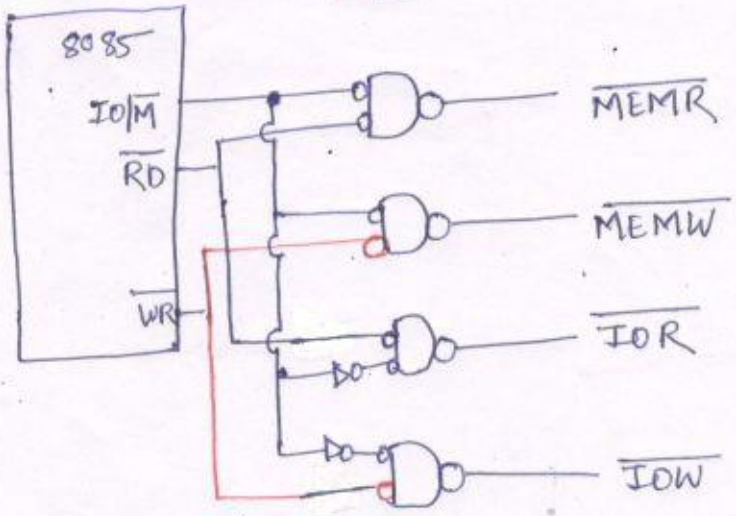
→ $\overline{RD}$ (Read) as a control signal. Because this signal is used both for reading memory and for reading input device.

→ It is necessary to generate two different Read signals: one for memory and other for input device.

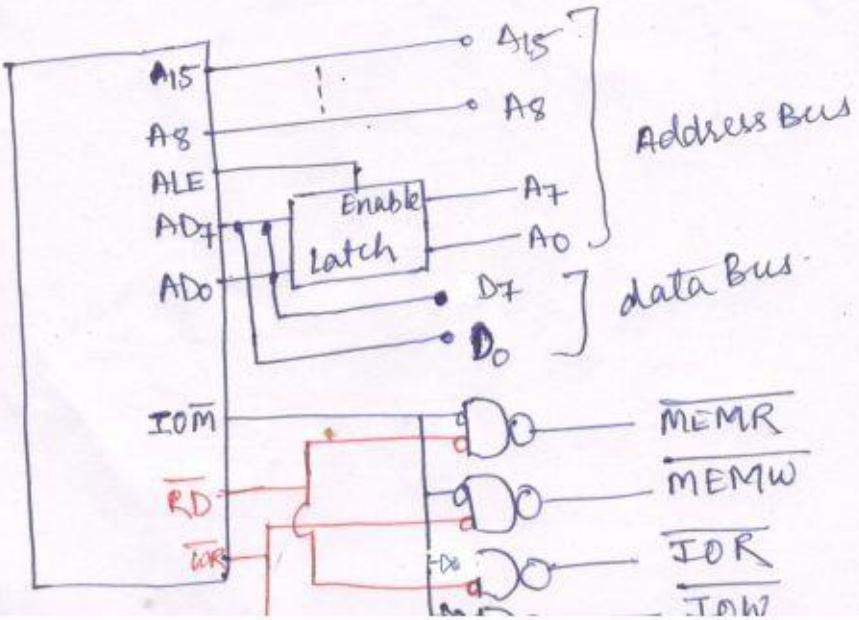
→ Similarly two separate write signals are generated

→ Four different control signals are generated by combining the signals $\overline{RD}$, $\overline{WR}$ and $IO/\overline{M}$

Schematic to generate Read/write control signal for memory and I/O



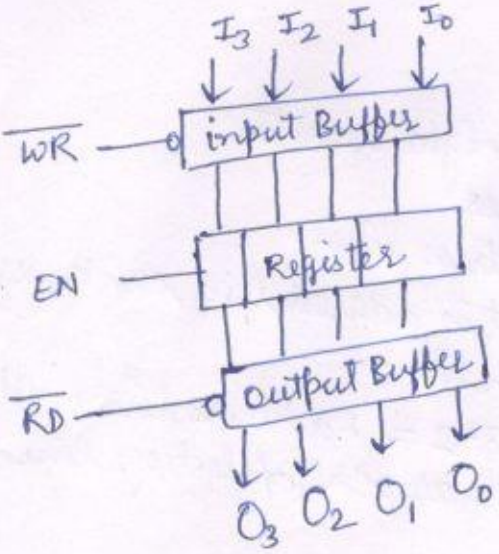
Demultiplexed Address and data bus with control signals

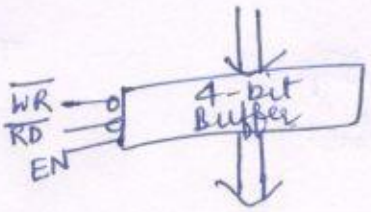


Memory Addressing and Interfacing

Addressing:

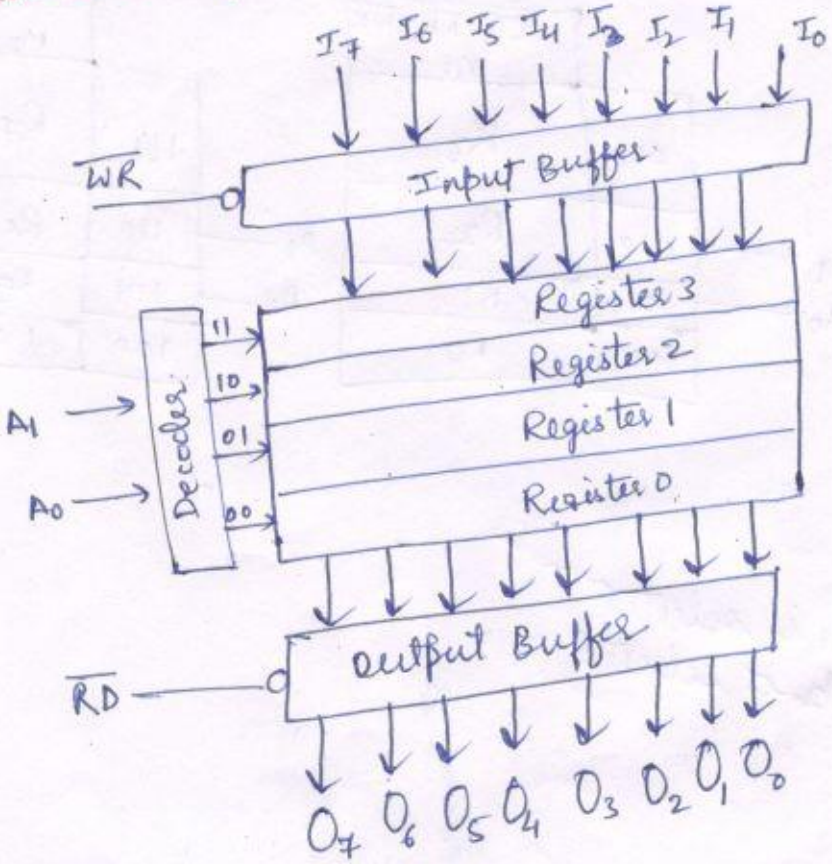
→ Block diagram of 4 bit register





Block diagram of 4-bit Register.

Ques: How to arrange 4 register with 8 ~~bit~~ cells.



Addresstable.

A_1	A_0	Register
0	0	Register 0
0	1	Register 1
1	0	Register 2
1	1	Register 3

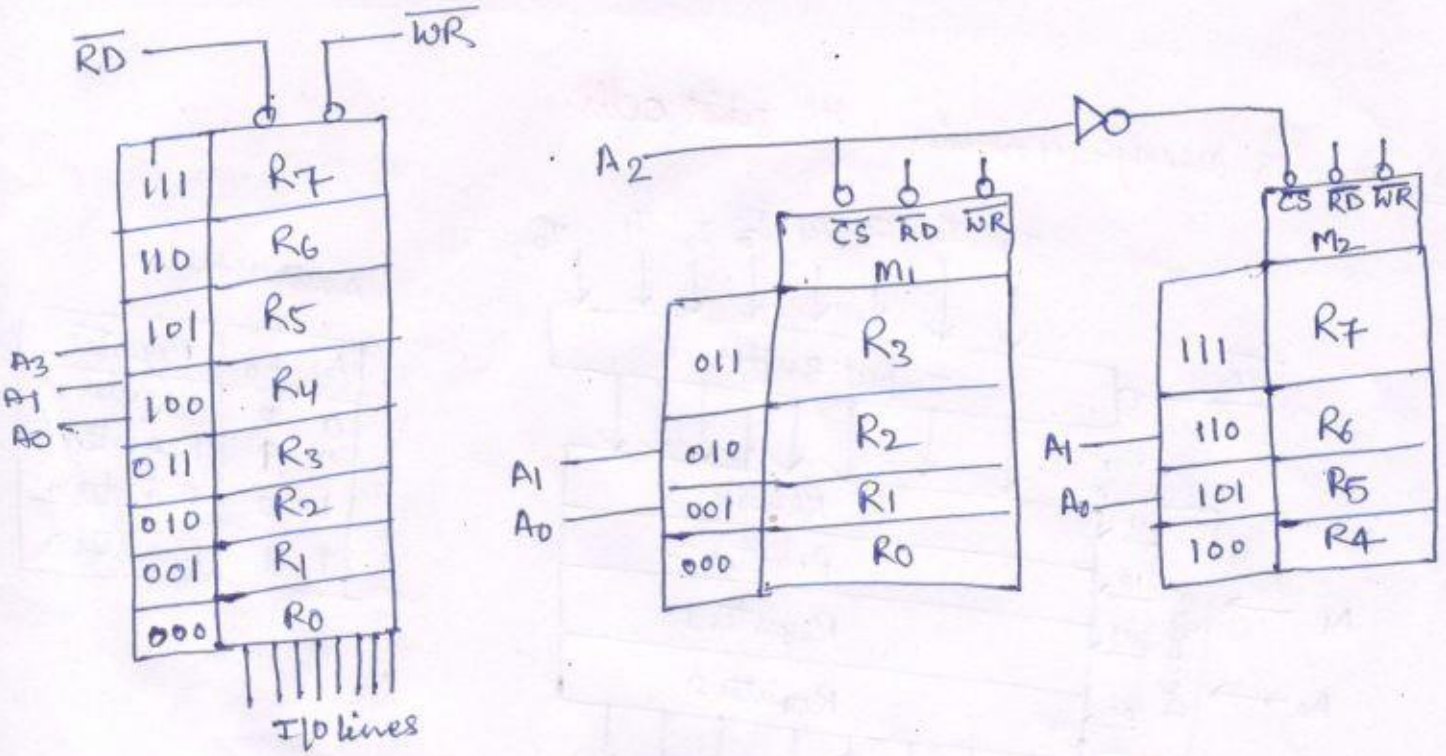
Aim: How to deal with more than one chip.

using chip select pin.

Ques: If we have two chips with four register each and to make a memory total of eight register.

chip has 4 registers
So 2 address lines
are there.

Total of eight register requires
3 address lines.
only two address line can be used with chip.
So 3-2 = 1 address line will be used as chip selection line.



when A_2 is low ⇒ chip M_1 is selected
when A_2 is High ⇒ chip M_2 is selected

Ques:

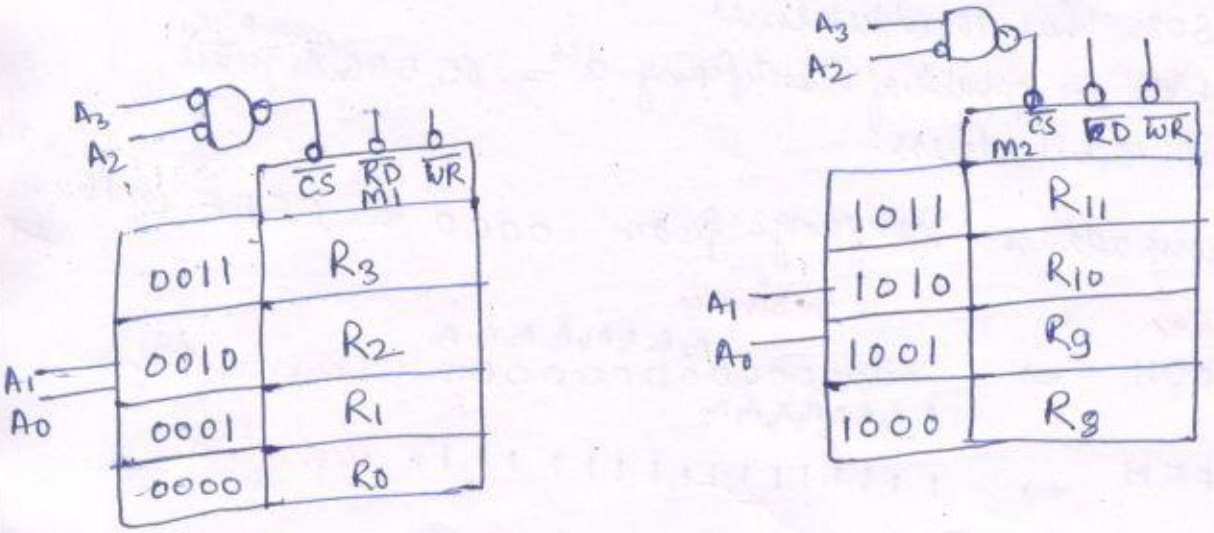
Assume that we have available four address lines and two memory chips with four registers each with 8 cells.

⇒ Given address lines = 4
4 address lines can address 2^4 ⇒ 16 registers.

⇒ Two memory chips with 4 registers ⇒ Total registers = 8
8 register requires address lines = 3.

⇒ chip has 4 register ⇒ has two address lines.

Total address lines given = 4
address lines in chip = 2
for chip selection = 4 − 2 = 2 remaining two address lines will be used for chip selection.

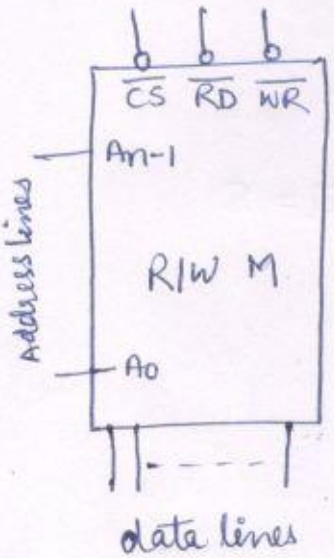


→ M1 is selected when $A_2 = 0$, $A_3 = 0$
→ M2 is selected when $A_2 = 0$ $A_3 = 1$

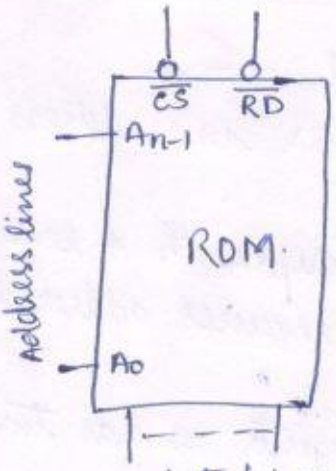
In this example we have taken all 4 lines for addres decoding such type of decoding is called Absolute or complete decoding.

Another option was to leave 4 line (A_3) as don't care.

R/W Memory Model (or RAM)



ROM Model



Memory Map and Addresses

→ Microprocessor 8085 has 16 address lines

→ So 16 binary bits are capable of identifying 2^{16} = 65,536 memory registers, each register with 16 bit address.

→ The entire memory addresses can range from 0000 to FFFF in Hex.

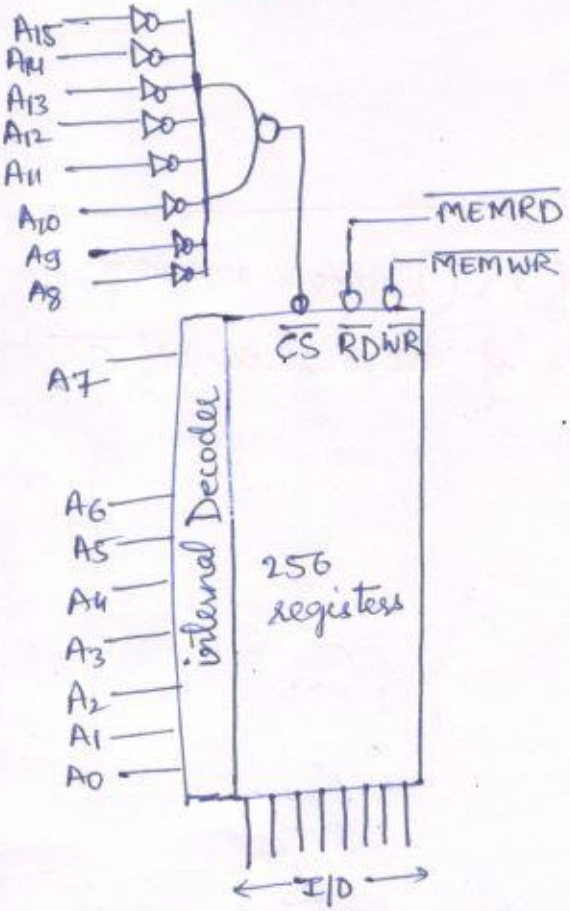
in Hex		in binary ($A_7 A_6 A_5 A_4 A_3 A_2 A_1 A_0$ / $A_{15} A_{14} A_{13} A_{12} A_{11} A_{10} A_9 A_8$)	
0000H	⇒	0000000000000000	Lowest address
FFFFH	⇒	1111111111111111	Highest Address

Ques: If we need only 256 register out of 65536 registers. The 256 registers require only 8 lines for addressing. Now what about (16-8) = 8 remaining address lines?

we can use remaining 8 lines to assign fixed logic to generate constant (fixed) number. This can be accomplished by using the

remaining eight lines for chip select through appropriate logic gates.

Example Illustrate the memory address range of the chip with 256 bytes of memory (given in figure.a) and explain how the range can be changed by modifying the hardware of the chip select.



(a)

$A_{15}A_{14}A_{13}A_{12}A_{11}A_{10}A_9A_8\ A_7A_6A_5A_4A_3A_2A_1A_0$

0 0 0 0 0 0 0 0 , 0 0 0 0 0 0 0 0

= 0000H

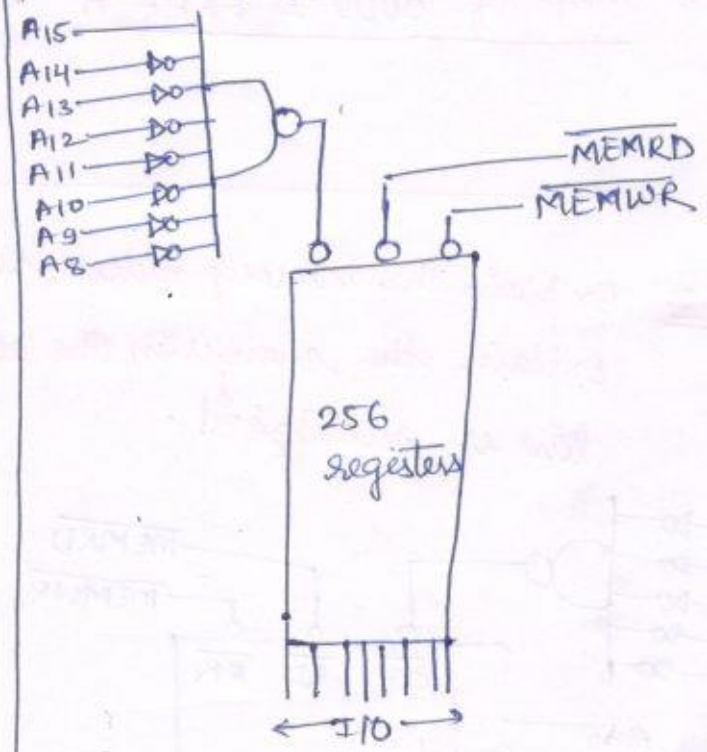
chip select or chip enable.

1 1 1 1 1 1 1 1

= 00FF H.

Register select.

Modifying the hardware.



(b)

chip select ← Register select →

$A_{15}\ A_{14}\ A_{13}\ A_{12}\ A_{11}\ A_{10}\ A_9\ A_8\ A_7\ A_6\ A_5\ A_4\ A_3\ A_2\ A_1\ A_0$

1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

= 8000 H

1 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1

= 80FF H

Here we remove the inverter of A_{15} line. The address required on A15-A8 to enable the chip = 80H

10000000 H

Ques: calculate the address lines required for an 8K Byte memory

Number of address lines required

$$n = \log_2(8 \times 1024)$$

$$= \log_2(2^3 \times 2^{10})$$

$$= 13$$

Ques: calculate the number of memory chips needed to design 8-K byte memory if memory chip size is 1024X1

$$\text{Number of chips required} = \frac{8 \times 1024 \times 8}{1024 \times 1}$$

$$= 64$$

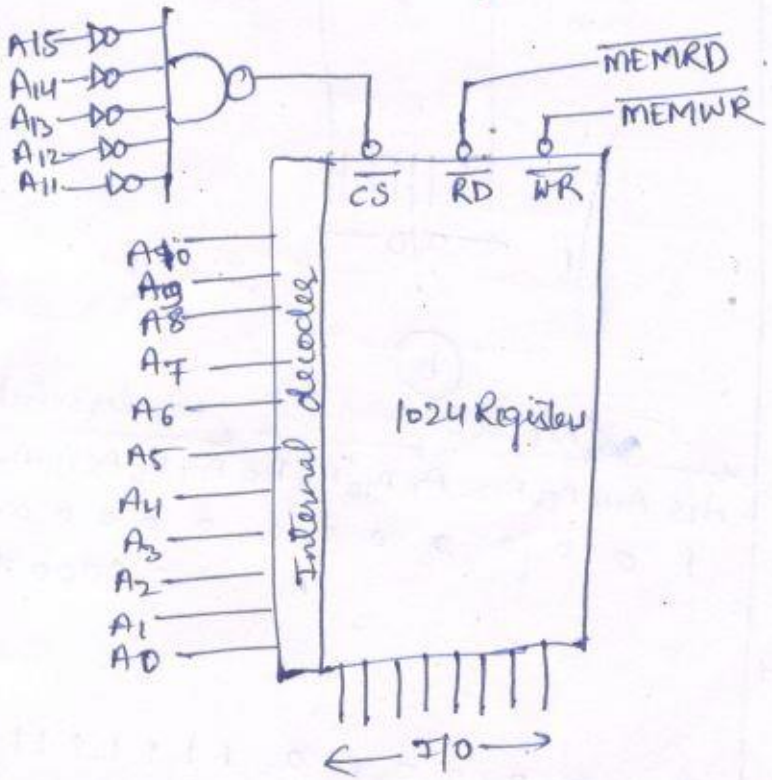
Ques: connect 8KB EPROM with 8085. The IC available is 2KX8 EPROM. Also draw the address decoding table.

Memory Map:

The term memory map is used generally for the entire address range of the memory chips in a given system.

The memory chips in the previous example (fig ⓐ and ⓑ) are the same chips. However by changing the hardware of the chip select logic, the location of the memory in the map can be changed, and memory can be assigned addresses in various location over the entire map of 0000 to FFFF H

Qu:- Explain the memory address range of 1K (1024x8) memory. Explain the changes in the addresses if the hardware of $\overline{CS}$ line is modified.



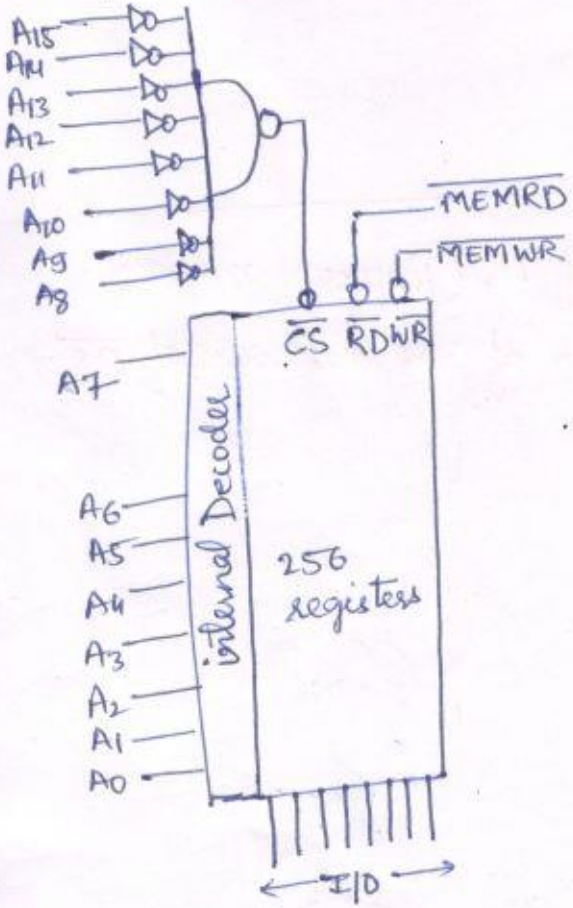
explain = ?

Hint

Address range 0000H to 03FFH

remaining eight lines for chip select through appropriate logic gates.

Example Illustrate the memory address range of the chip with 256 bytes of memory (given in figure.a) and explain how the range can be changed by modifying the hardware of the chip select.



(a)

$A_{15}A_{14}A_{13}A_{12}A_{11}A_{10}A_9A_8\ A_7A_6A_5A_4A_3A_2A_1A_0$

0 0 0 0 0 0 0 0 , 0 0 0 0 0 0 0 0

= 0000H

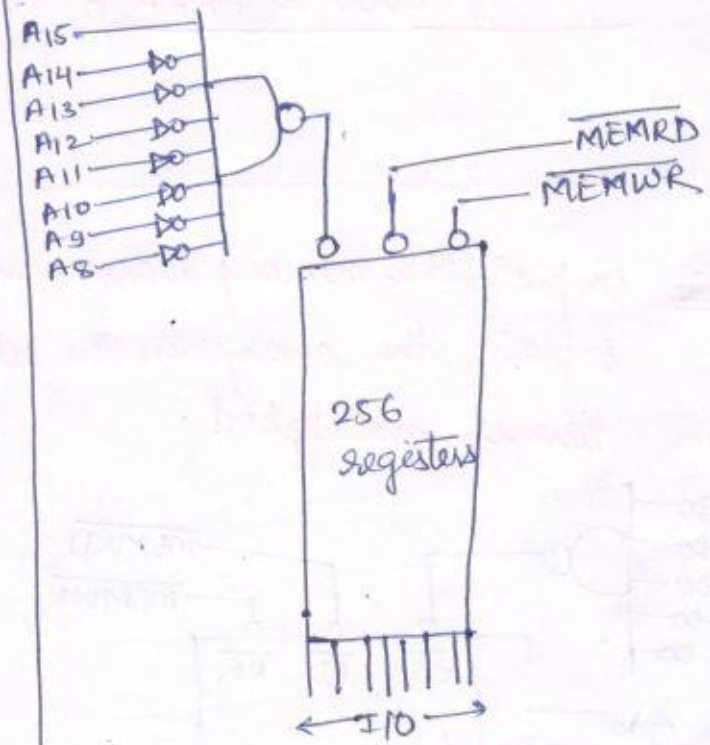
chip select or chip enable

1 1 1 1 1 1 1 1

= 00FF H

Register select

Modifying the hardware.



(b)

chip select ← → Register select

$A_{15}\ A_{14}\ A_{13}\ A_{12}\ A_{11}\ A_{10}\ A_9\ A_8\ A_7\ A_6\ A_5\ A_4\ A_3\ A_2\ A_1$

1 0 0 0 0 0 0 0 0 0 0 0 0 0 0

= 8000 H

1 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1

= 80FF H

Here we remove the inverter of A_{15} line. The address required on A15-A8 to enable the chip = 80H

10000000 H

Memory Interfacing:

→ How to interface a memory chip with the microprocessor.

→ while executing a program, the microprocessor needs to access memory quite frequently to read instruction codes and data stored in memory; interfacing circuit enables that access.

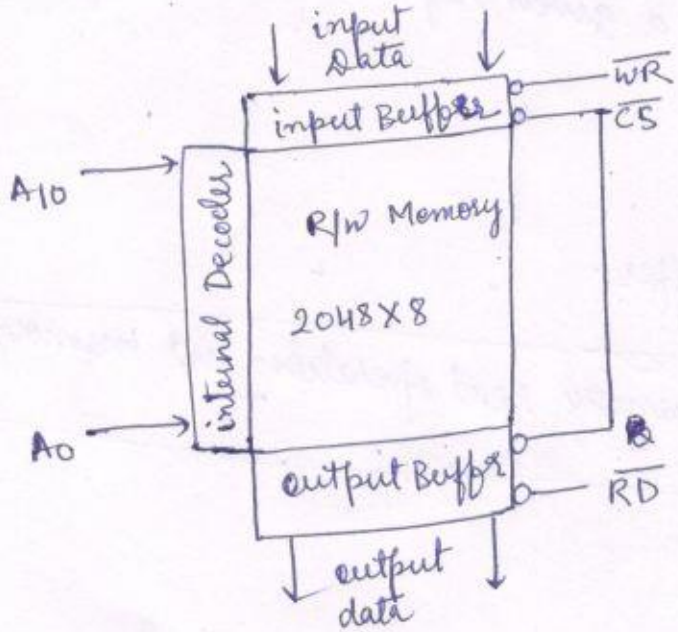
→ Memory has certain signals requirement to write into and read from its registers. Similarly, microprocessor initiates a set of signals for memory access.

→ The interfacing circuit involves designing a circuit that will match the memory requirements with the microprocessor signals

Memory structure and its requirements:

Memory (R/W M) → a group of registers to store binary information

Example structure of R/W memory with 2048 registers with 8 bit each.



$\overline{CS}$ ⇒ for chip selection

$\overline{RD}$ ⇒ to enable output buffer

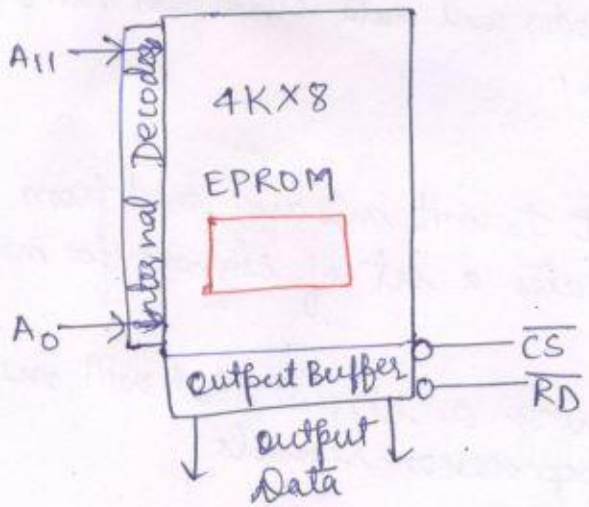
$\overline{WR}$ ⇒ control to enable input buffer.

Address lines - A_0 to A_{10}

data lines ⇒ 8

Example

Structure of ROM consisting 4K register with 8 bit each.



→ Address lines ⇒ $A_0 - A_{11}$

$\overline{CS}$ for chip select.

$\overline{RD}$ ⇒ control to enable output buffer

This chip must be programmed (written into) before it can be used as a read-only memory.

→ This chip structure shows a quartz window on the chip that is used to expose the chip to ultraviolet rays for erasing the program.

→ Once the chip is programmed, the window is covered with opaque tape to avoid accidental erasing.

Basic concepts in memory interfacing:

- The primary function of memory interfacing is that the microprocessor should be able to read from and write into a given register of a memory chip.
- Microprocessor should
  - → be able to select the chip
  - → identify the register
  - → enable the appropriate buffer.

Let examine the timing diagram of memory Read operation and memory write operation

Memory Read cycle:

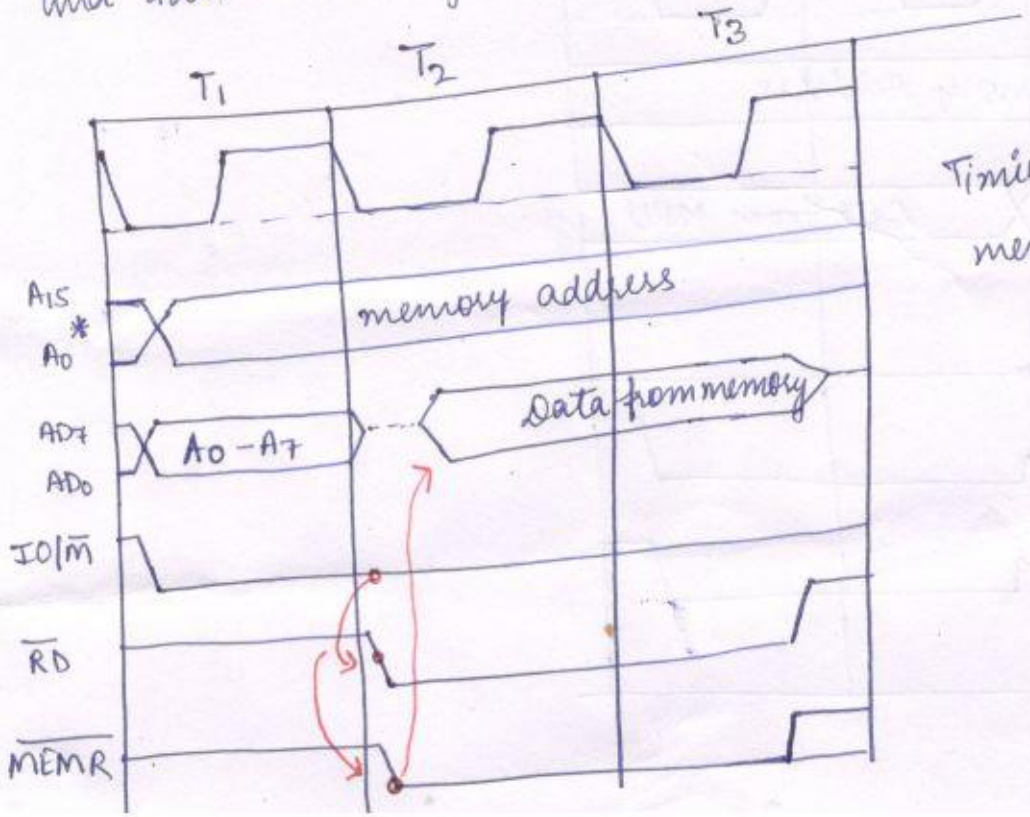
(1) → The 8085 places a 16-bit address on the address bus, and with this address only one register should be selected. (Example of 2048 registers)

→ For example, 2048 registers only 11 address lines are required to identify 2048 registers. Therefore, we can connect the low-order address lines A_{10}-A_0 of the 8085 address bus to the memory chip. The internal decoder of the memory chip will identify and select the register for the EPROM.

(2) → The remaining 8085 address lines (A_{15}-A_{11}) should be decoded to generate a chip select ($\overline{CS}$) signal unique to that combination of address logic.

(3) → The 8085 provides two signals – IO/$\overline{M}$ and $\overline{RD}$ to indicate that it is memory read operation. The IO/$\overline{M}$ and $\overline{RD}$ can be combined to generate the $\overline{MEMR}$ (memory read) control signal that can be used to enable the output buffer by the connecting to the memory signal $\overline{RD}$

(4) the memory places the data byte from addressed register during T_2 t-state and that is read by the microprocess before the end of T_3 t-state.



Timing diagram for memory read cycle

Memory Write cycle:

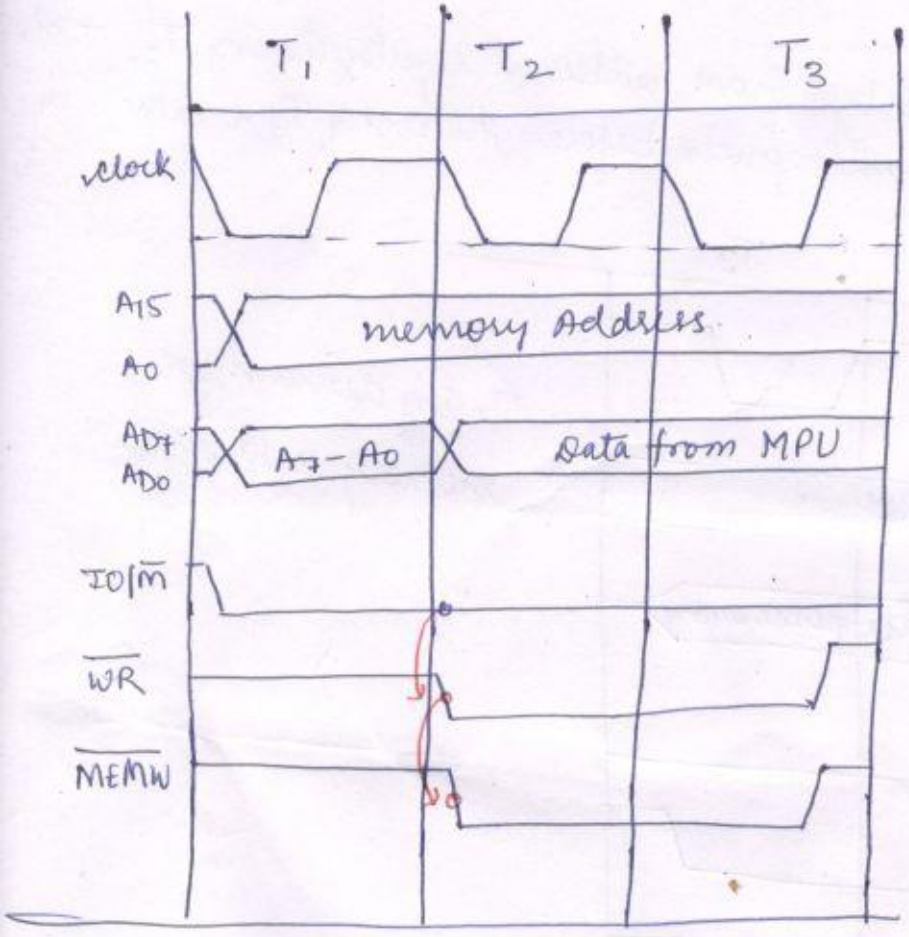
To write into a register.

→ The 8085 places the address and data and asserts the IO/$\overline{M}$ signal

→ After allowing sufficient time for data to become stable, it asserts write ($\overline{WR}$) signal. The $\overline{WR}$ and IO/$\overline{M}$ signals can be combined to generate $\overline{MEMWR}$ control signal that enables the input buffer of the memory chip and stores the bytes in the selected memory register.

Memory Interfacing:

① Connect the required address lines of the address bus to the address



Instruction classification

→ An instruction is a binary pattern designed inside a microprocessor to perform a specific function.

→ The entire group of instructions, called the instruction set, determines what functions the microprocessor can perform.

→ The 8085 instructions can be classified into the following five functional calgories:

- Data transfer (copy) operations
- Arithmetic operations
- Logical operations
- Branching operations
- Machine Control operations

Data Transfer (copy) operations:

- This group of instructions copies data from a location called a source to another location called destination, without modifying the contents of the source.
- Various types of data transfer.

Types	Examples
* Between registers	copy content of reg B into reg D.
* Specific data byte to a register or a memory location	Load register B with data byte 32H
* Between a memory location and a register	From memory location 2000H to register B.
* Between an I/O device and the Accumulator	From keyboard to accumulator

Arithmetic operations

→ These instructions perform arithmetic operations such as addition, subtraction, Increment and decrement.

→ Addition :- Any 8 bit number, or the contents of register or the contents of a memory location can be added to the contents of accumulator and the sum is stored in the accumulator.

- No two other 8 bit register can be added directly (example register B contents cannot be added to contents of register C).

- The instruction DAD is an exception; it adds 16-bit data in register pair.

→ Subtraction :- Any 8-bit number or the contents of register, or content of memory location are subtracted from the contents of accumulator and the Results are stored in accumulator.

- The subtraction is performed in 2's complement, an the result is negative, then it is expressed in 2's complement.

→ Increment / Decrement :-

- The 8-bit contents of register or a memory location can be Incremented or decremented by 1.

- the 16-bit contents of register pair can be incremented or decremented by 1.

- These operation can be performed in any one of register or in a memory location.

Logical operations:

These instructions perform various logical operations with the contents of the accumulator.

AND, OR, X-OR: Any 8-bit number, contents of a register or contents of a memory location can be ANDed, ORed or X-ORed with the contents of accumulator.

Rotate: Each bit in the accumulator can be shifted either left or right to the next position.

compare: Any 8-bit number, or the contents of a register or the contents of memory location can be compared for equality, greater than or less than, with the contents of the accumulator.

complement: The contents of the accumulator can be complemented.

Branching operations:

This group of instruction alters the sequence of program execution either conditionally or unconditionally.

Jump:

- conditional jumps are important aspect of the decision making process in programming.
- These instructions test for a certain condition (e.g. Zero or Carry f
and alter the program sequence when the condition is met.
- uncondition jumps are also there.

Call, Return and Restart:

- These instructions change the sequence of a program either by calling a subroutine or returning from a subroutine.
- The conditional call and Return instructions also can test condit
flags.

Machine control operations:

- These machine instructions control machine functions such as Halt, Interrupt or do nothing.

Instruction word size:

- Instruction is a command to the ~~processor~~ microprocessor to perform a given task on the specified data.

- Each instruction has two parts;
  - opcode → task to be performed.
  - operand → the data to be operated on.

- The 8085 instruction set is classified into three groups according to the word size or byte size.
  - 1-byte instruction
  - 2-byte instruction
  - 3-byte instruction

  } instruction formats

One-Byte Instruction:

A 1-byte instruction includes the opcode and the operand in the same byte.

Example

MOV C, A

↑ opcode ↑ operand.

0100 1111

↑ Binary code

4F H

↑ Hex code.

⇒ MOV C,A means copy the content of accumulator in register C.

Two-Byte Instruction:

In a two byte instruction, the first byte specifies the operation code and the second byte specifies the operand. for example.

	opcode	operand	Hex code	
MVI A, 32H	MVI	A, 32H	3E	First byte
			32	second byte.

- Load the 8-bit data byte (i.e 32H) into accumulator.
- These instructions would require two memory locations to be stored..

Three byte instruction:

In a 3-byte instruction, first byte specifies the opcode. the following two bytes specify the 16-bit address.

* Note that second byte is the lower order address and the third byte is the higher order address

For example

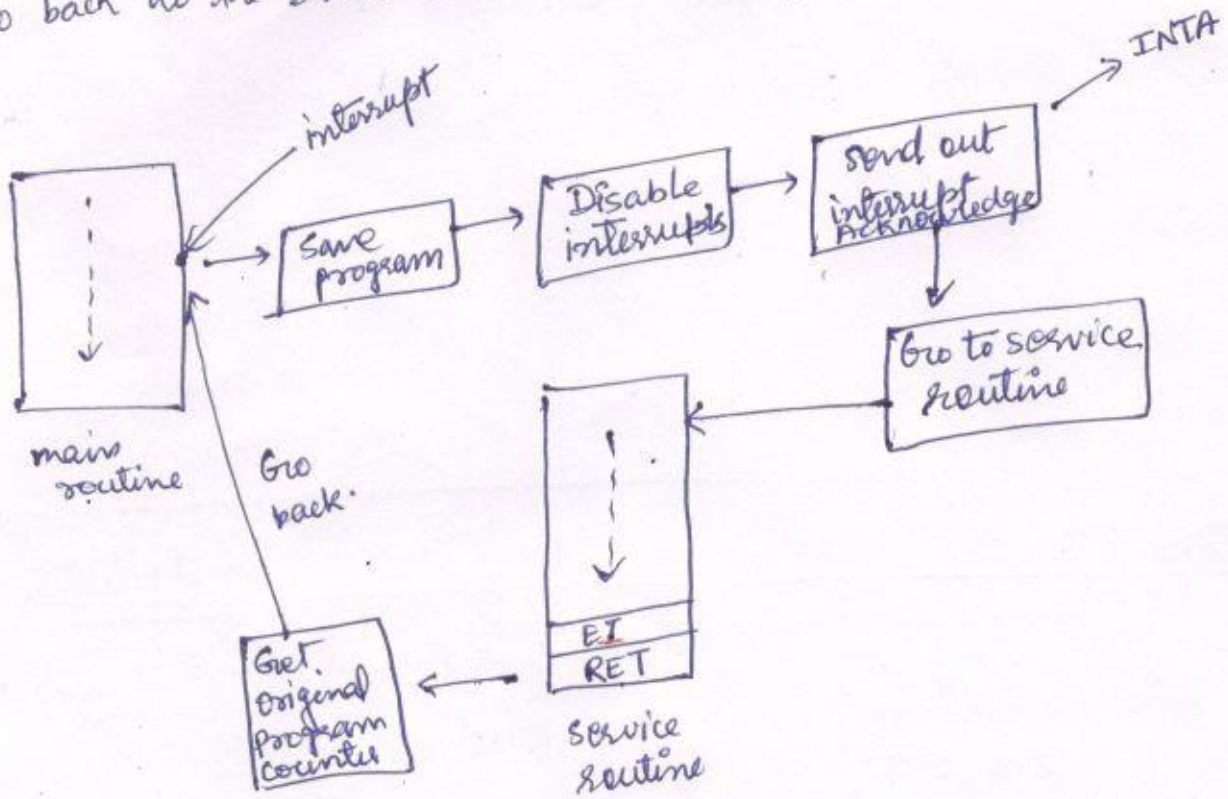
	opcode	operand	Hex code	
LDA	LDA	2050H	3A	First byte
			50	second byte
			20	Third byte

⇒ load the contents of memory 2050H into accumulator.

⇒ These instructions would require three memory locations each to store the binary codes.

Interrupts (Interrupts in 8085)

In order to execute an interrupt routine, the processor -
- should be able to accept interrupts (interrupt enable)
- save the last content of the program counter
- Know where to go in the program memory to execute the ISR
- Tell the outside world that it is executing an interrupt.
- Go back to the saved PC location when finished.



8085 has 5 interrupt inputs

→ - The INTR input
- The INTR is the only non-vectored interrupts
- INTR is maskable using EI/DI instruction pair.

→ RST 5.5, RST 6.5, RST 7.5 are all automatically vectored.
- RST 5.5, RST 6.5 and RST 7.5 are all maskable.

→ TRAP is the only non-maskable interrupt in 8085
TRAP is also automatically vectored

8085 Interrupts:

issues: ① How interrupts are serviced in 8085
both [non vectored
Vectored

② How to prioritize the multiple interrupts
— using Priority Encoder.

8085 interrupts:

8085 has 5 interrupt inputs

① INTR
② RST 5.5, RST 6.5, RST 7.5
③ TRAP

Interrupt	Maskable	Vectored
INTR	Yes	No
RST 5.5	Yes	Yes
RST 6.5	Yes	Yes
RST 7.5	Yes	Yes
TRAP	No	Yes

The 8085 nonvectored Interrupt Process

(1) The interrupt process should be enabled using the EI instruction (Enable Interrupt instruction).

(2) The 8085 checks for an interrupt during the execution of every instruction.

(3) If there is an interrupt, the microprocessor will complete the executing instruction, and starts a RESTART sequence.

(4) The RESTART sequence resets the interrupt flip flop and activates the interrupt acknowledge signal (INTA)

(5) upon receiving the INTA signal, the interrupting device is expected to return the opcode of one of the RST instructions (RST0, RST1, RST2, RST3, RST4, RST5, RST6, RST7).

(6) when the microprocessor executes the RST instruction received from the device, it saves the address of the next instruction on to the stack and jumps to the appropriate entry in IVT.

(7) The IVT entry must redirect the microprocessor to the actual service routine.

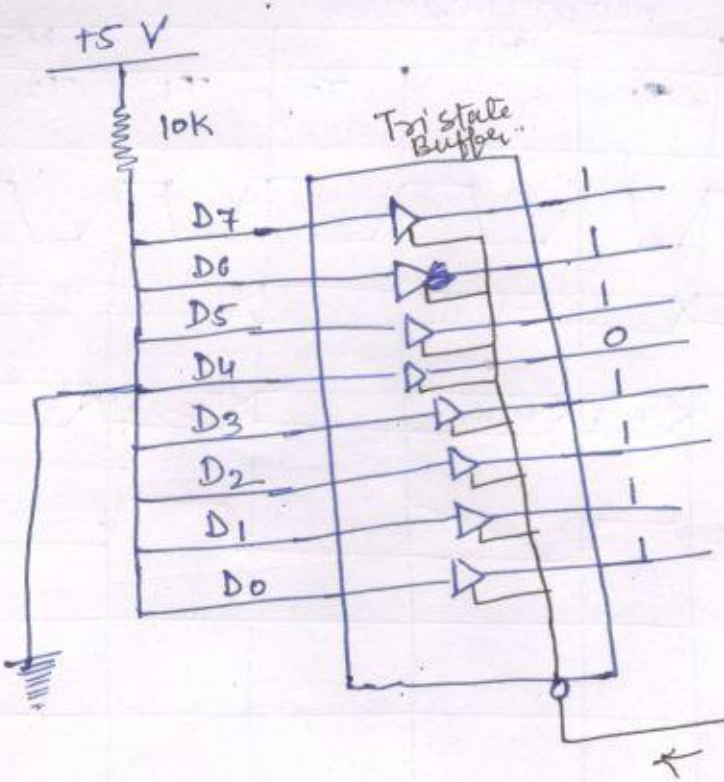
(8) The service routine must include the instruction EI to re-enable the interrupt process.

(9) At the end of the service routine, the RET instruction returns the execution to where the program was interrupted.

Restart instructions

Mnemonics	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	Hexcode	Call location in Hex
RST0	1	1	0	0	0	1	1	1	C7	0000H
RST1	1	1	0	0	1	1	1	1	CF	0008H
RST2	1	1	0	1	0	1	1	1	D7	0010H
RST3	1	1	0	1	1	1	1	1	DF	0018H
RST4	1	1	1	0	0	1	1	1	E7	0020H
RST5	1	1	1	0	1	1	1	1	EF	0028H
RST6	1	1	1	1	0	1	1	1	F7	0030H
RST7	1	1	1	1	1	1	1	1	FF	0038H

circuit to implement RST5



EF ⇒ data bus.

There are 8 restart instruction RST0–RST7 and each of this would send the execution to a particular predetermined hardware memory address.

8085 Interrupt Acknowledge machine cycle: (or Restart sequence)

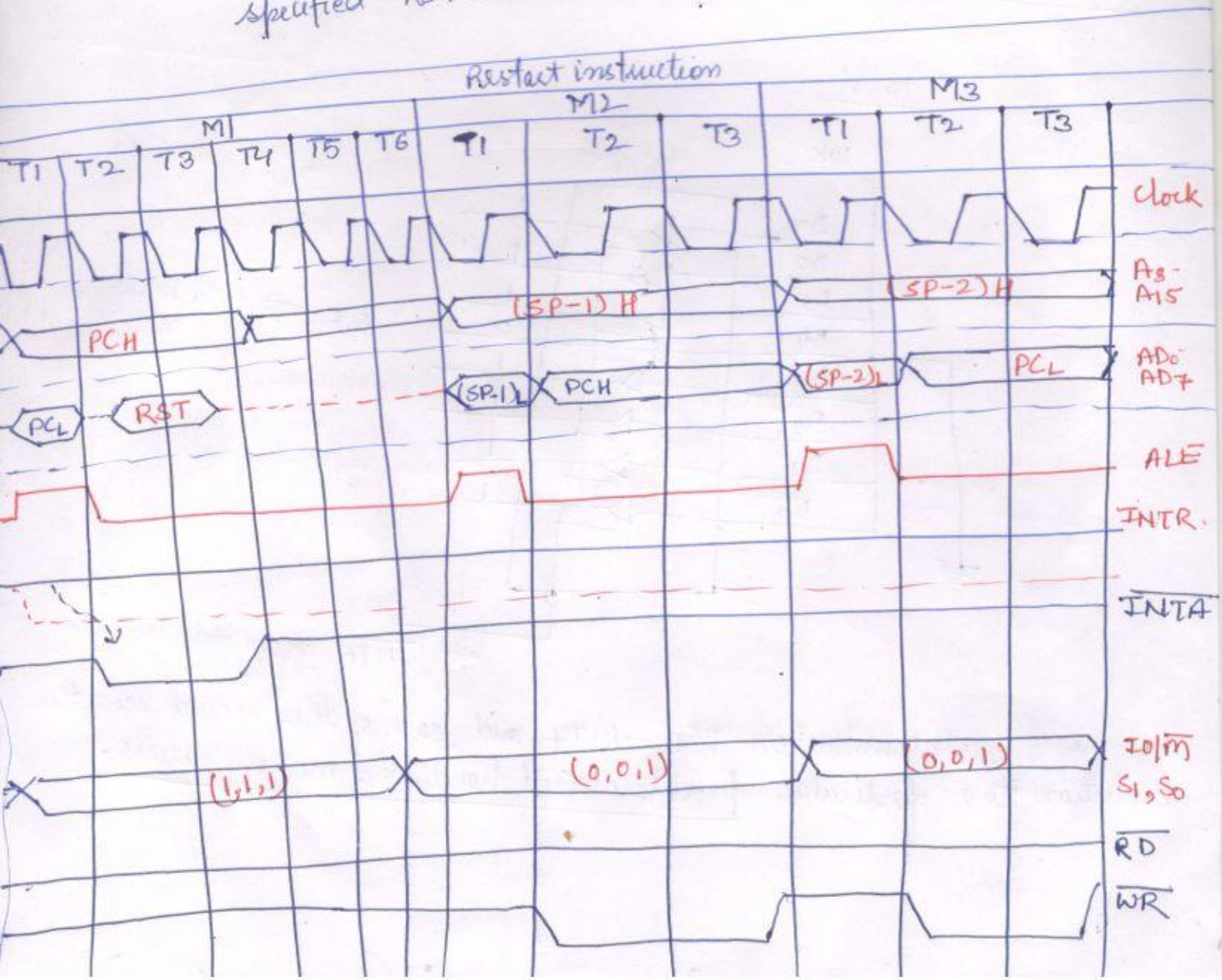
→ The restart sequence is made up of three machine cycles.

In the first machine cycle:

- The microprocessor sends $\overline{INTA}$ signal
- while $\overline{INTA}$ is active the microprocessor reads the data lines expecting to receive, from the interrupting device, the opcode for the specific RST instruction.

In second machine cycle and 3rd machine cycle:

- the 16 bit address of the next instruction is saved onto the stack.
- Then microprocessor jumps to the address associated with the specified RST instruction.



8085 Vectored Interrupts

The 8085 has 4 masked/vectored interrupts inputs.

→ - RST 5.5, RST6.5, RST7.5, and TRAP
- all four are automatically vectored.

Priority	Interrupts	Call Locations
Highest ↑	TRAP	0024H
	RST7.5	003CH
	RST6.5	0034H
~~Lowest~~	RST5.5	002CH

INTR has lowest priority. ~~TRA~~

TRAP > RST7.5 > RST6.5 > RST5.5 > INTR

→ RST7.5, RST6.5, and RST5.5 are maskable interrupt

→ TRAP is non maskable interrupt

→ TRAP is generally used for such critical events as power failure and emergency shut off.

Ques: why RST 5.5 is call 5 and half.

when RST 5.5 occurs, microprocessor jumps to 002CH which is between RST5 and RST6 that is why it is called RST (0028H) and (0030H)

→ Similarly RST 6.5 between RST 6 and RST7.

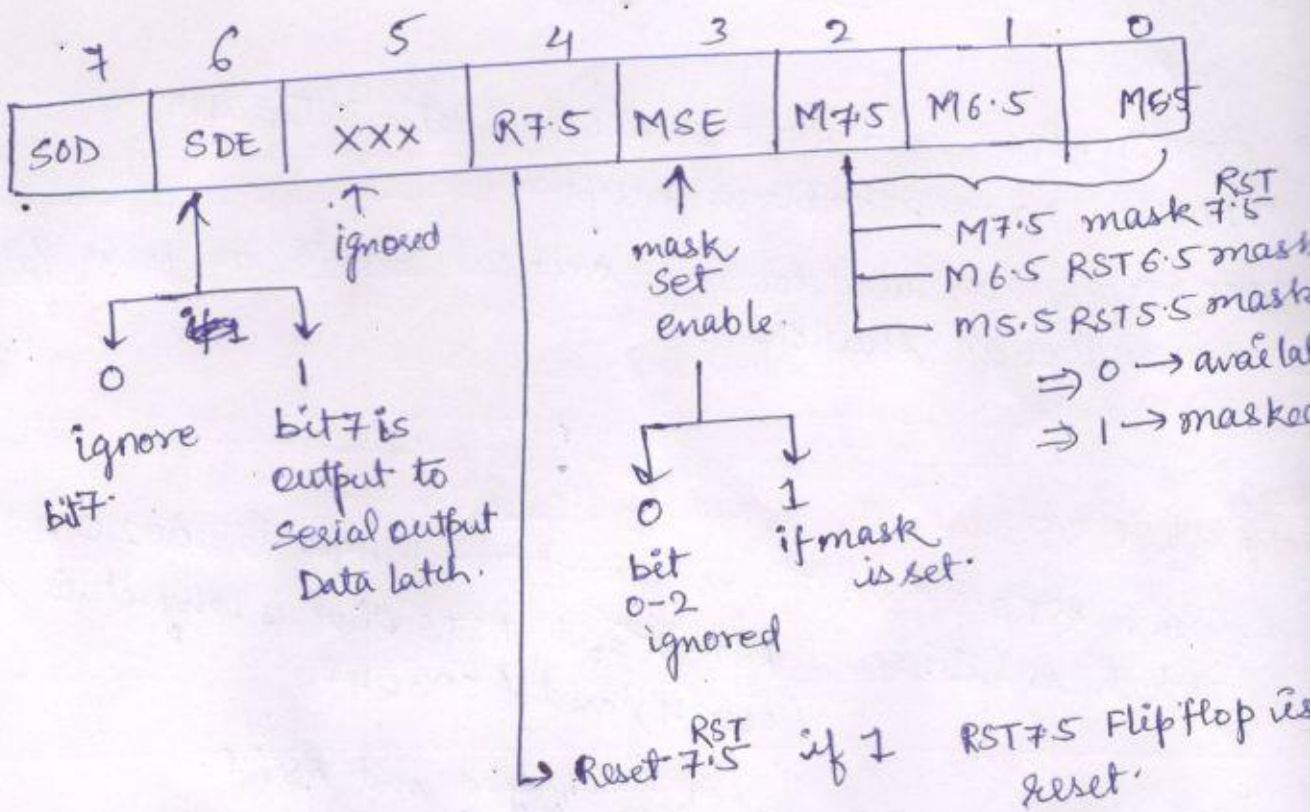
→ After 7, RST7.5 is 4 bytes away from 7.

SIM Instruction:

→ RST7.5, RST6.5, RST5.5 maskable interrupts are enabled under program control with two instructions
- EI (Enable interrupts)
- SIM (Set Interrupt mask)

→ SIM is a 1-byte instruction and can be used for three functions

① To set mask for RST7.5, RST6.5, and RST5.5 interrupts. This instruction reads the content of accumulator and enables or disables the interrupts according to the content of accumula



② To reset RST7.5 flip flop.

③ to implement Serial I/O.

SIM instruction must be executed in order to use the interrupts.

- Enable all interrupts in an 8085 system

```
EI          Enable interrupts
MVI A, 08H  load bit pattern to enable RST7.5, RST6.5, RST5.
SIM         Enable RST7.5, 6.5 and 5.5
```

bit $D_3 = 1$ in accumulator makes the instruction SIM functional and bits D_2, D_1 and D_0 enable the interrupts 7.5, 6.5 and 5.5.

Now reset 7.5 interrupt

```
MVI A, 18H      Set D4 = 1
SIM             Reset 7.5 interrupt flip flop.
```

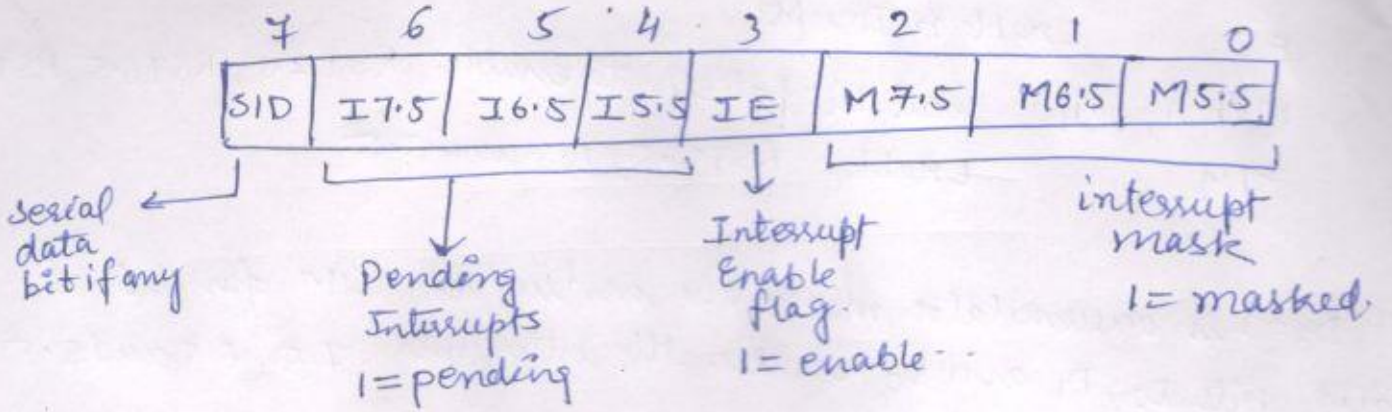
RIM (Read Interrupt Mask)

- Because there are several interrupt lines, when one interrupt is being served, other interrupt request may occur and remain pending.
- 8085 has an additional instruction RIM to sense the pending interrupts

RIM instruction – is a 1-byte instruction can be used for the following functions:

- To read interrupt masks
- To identify ~~not~~ pending interrupts
- To receive serial data.

RIM


7 6 5 4 3 2 1 0
SID
I7.5
I6.5
I5.5
IE
M7.5
M6.5
M5.5
serial data bit if any
Pending Interrupts 1 = pending
Interrupt Enable flag. 1 = enable
interrupt mask 1 = masked

Assembler Directives:

what is an assembler?

→ An assembler is a program that translates assembly language mnemonics or source code into binary executable code.

→ This translation requires that the source program be written strictly according to the specified syntax of the assembler.

Assembler directives:

→ The assembler directives are the instructions to the assembler concerning the program being assembled;

→ also called pseudo instructions or pseudo codes.

* These instructions are neither translated into machine code nor assigned any memory locations in the object file.

what assembler directives specifically do: (Functions of Assemble directives)

→ show the begining and end of the program provided to assembles.

→ used to provide storage locations to the data

→ used to given values to the variables.

→ define the start and end of different segments, procedures or macros of a program.

Types of Assembler Directives

1. ORG
2. END
3. EQU
4. DB
5. DW
6. DS

① ORG (origin) ⇒ Originate

Example ORG 2000H

⇒ The next block of instructions should be stored in memory locations starting at 2000H.

② END END ⇒ End Program

~~END~~

③ EQU (Equate) FACTOR EQU 03H

This assembler directive used to give a name to some value or to a symbol.

④ DB - DB directive is used to declare a byte type variable to store a byte in memory locations.

PRICE DB 49H, 98H, 29H initialize PRICE byte by byte

⑤ DW:-

is used to define a variable of type word or to reserve storage locations of type word in memory.

DW 2050H. two bytes at a time.

~~MBuild~~ DW 2050H 2051H 2052H

⑦ DS: OUTBUF: DS 4

reserves specified number of storage location

it provides 4 locations to OUTBUF.