Interfacing:

Interface is a medium through which ~~to~~ two seperate components of a computer system exchange information or interact with each other. Interfacing a microprocessor is to connect it with various peripherals to perform various operation to obtained a desired output.

The interfacing process involves matching the memory requirement with the microprocessors signal. The interfacing circuit therefore should be designed in such a way that it matches the memory signal requirement with the signals of the microprocessor.

There are two types of interfacing in context of the 8085 microprocessor.

(a) ~~A= Memor~~ memory interfacing

(b) I/O interfacing

1. Memory interfacing:

while executing an instruction, there is necessity for the microprocessor to access memory frequently for reading various instruction codes and data stored in the memory. The interfacing circuit aids (helps) in accessing the memory.

2. I/O interfacing:

Keyboards and displays are used as communication channel with outside world. So it is necessary ~~to~~ to interface keyboard and displays with the µp. The process of connecting or interfacing I/O devices with the µp is called I/O interfacing. In this type of interfacing latches and buffers are used

for interfacing the keyboard and displays with the µp.

concept of memory interfacing:
The programs and data that are executed by the µp have to be stored in ROM/EPROM and RAM which are basically semiconductor memory chips. µp needs to access memory quite frequently to read instructions and data stored in memory, the interface circuit enables that access.

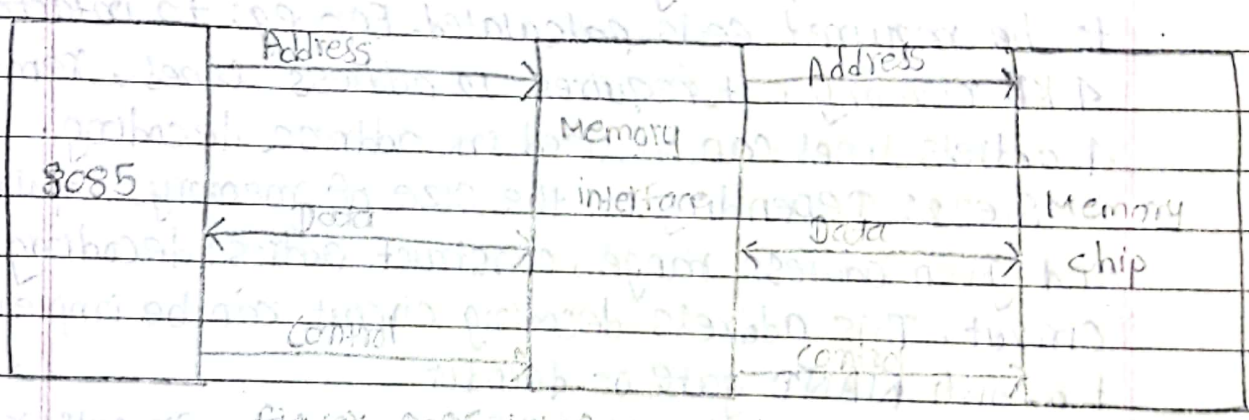


fig:or. 8085 interfacing with memory chip

The interface process involves designing a circuit that will match the memory requirements with the µp signal. Memory has certain signal requirements to read from and write into memory. Similary µp initiates the set of signals when it wants to read from and write into memory.

* 8085 has 16 address lines (A_0-A_{15}), Hence a maximum of 64 KB equals to 2^{16} bytes of memory locations can be interfaced with it.
* The memory address space of the 8085 takes values

from 0000H to FFFFH.

* The 8085 initiates set of ~~seg~~ signals such as ~~IO/M~~ $IO/\overline{M}$, $\overline{RD}$ and $\overline{WR}$ when it wants to read from and write into memory.
* Similarly each memory chip has signals such as CE or CS (chip enable or chip select), OE or $\overline{RD}$ (output enable or Read) and WE or $\overline{WR}$ (write enable or write) associated with it.

Steps involved in interfacing memory with 8085 µP:

Step 1: First decide the size of memory required to be interfaced. Depending on this address lines that is to be required ~~ee~~ is calculated. For eg: to interface 4 KB memory, it requires 12 address lines, remaining 4 address lines can be used in address decoding.

→ Step 2: Depending on the size of memory required and given address range, construct address decoding circuit. This address decoding circuit can be implemented with NAND gate or decoder.

→ Step 3: Connect data bus of memory to processor data bus.

→ Step 4: Generate the control ~~sig~~ signal (CS) required for memory ~~use~~ using $IO/\overline{M}$, $\overline{WR}$, $\overline{RD}$ signals of 8085 processor

Q. WAP

(a) Load the no. 1BH in D. → MVI D, 1BH

(b) Load the no. B5H in B. → MVI B, 5BH

(c) Increment the content of B by 1 → INR B

(d) Decrement the content of D by 1 → DCR D

(e) Subtract the content of D from the content of B.

(f) Display the result at out PORT 1.

(e) MOV A, B
SUB D

(f) OUT PORT1
~~HLD~~ HLT

(g) Store the content of D at C000H

Sol^n
MOV A, D
STA C000H

Q. WAP to add two numbers in register and store the content or result at B500H.

MVI A, 20H (Move the content in A)
MVI B, 10H (Move the content in B)
ADD B (Add B
STA B500H (S

Q. WAP to display the largest number among the following. C001H, 0001H.
Q. WAP to multiply two numbers.

8085 instruction sets:

* Data transfer instructions:

1. MOV ⇒ This instruction use to copies the contents from source to destination.
2. MVI # This instruction is use to move contents immediately.
3. LDA # This instruction is use to load content in Accumulator.
4. LDAX # This instruction is use to load content in register pair.
5. LXI: This instruction is use to load content in register pair immediately.
6. LHLD: This instruction is used to load content directly in H and L register.
7. STA: This instruction is used to store content directly in accumulator.
8. STAX: This instruction is used to store content indirectly in ~~accumulator~~ register pair.
9. XCHG: This instruction is used to exchange the content of register H and L with D and E.
10. SPHL: This instruction is used to copy content from H and L registers to the stack pointer.
11. XTHL: This instruction is used to exchange the content of H and L register with top of stack.
12. PUSH: This instruction is used to Push content of register pair onto stack.
13. POP: This instruction is used to POP off Stack to register pair.
14. OUT: This instruction is used to give output data from accumulator to a port with 8-bit address.
15. IN: This instruction is used to take input data to accumulator from a port with 8-bit address.

* Arithmetic instructions:

16. ADD: This instruction is used to add register or memory to accumulator.
17. ADC: This instruction is used to add register or memory to accumulator with carry.
18. ADI: This instruction is used to add content immediately to accumulator.
19. ACI: This instruction is used to add content immediately to accumulator with carry.
20. DAD: This instruction is used to add content in register pair to H and L registers.
21. SUB: This instruction is used to subtract content from accumulator.
22. SBB: This instruction is used to subtract source and borrow from accumulator.
23. SUI: This instruction is used to subtract content immediately from accumulator.
24. SBI: This instruction is used to subtract content immediately from accumulator with borrow.
25. INR: This instruction is used to increment the content by 1.
25. INX: This instruction is used to increment register pair by 1.
26. DCR: This instruction is used to decrement register or memory by 1.
27. DCX: This instruction is used to decrement register pair by 1.
28. DAA: This instruction is used to adjust decimal in accumulator.

Branching instructions:

29. JMP: This instruction is used to jump memory location unconditionally.
30. ~~Jump Conditionally Con~~
31. JNC: This instruction is used to jump conditionally if there is no carry (CY = 0)
32. JP: This instruction is used to jump conditionally if there is positive sign. (S = 0)
33. JM: This instruction is used to jump conditionally if there is minus sign. (S = 1)
34. JZ: This instruction is used to jump conditionally if there is zero. (Z = 1)
35. JNZ: This instruction is used to jump conditionally if there is no zero. (Z = 0)
36. JPE: This instruction is used to jump conditionally if there is even parity. (P = 1)
37. JPO: This instruction is used to jump conditionally if there is odd parity. (P = 0)
38. CALL: This instruction is used to call memory location unconditionally subroutine.
39. CC: This instruction is used to call conditionally if there is carry. (CY = 1)
40. CNC: This instruction is used to call conditionally if there is no carry. (CY = 0)
41. CP: This instruction is used to call conditionally if there is positive sign. (S = 0)
42. CM: This instruction is used to call conditionally if there is minus sign. (S = 1).
43. CZ: This instruction is used to call conditionally if there is zero. (Z = 1)
44. CNZ: This instruction is used to call conditionally if there is no zero. (Z = 0)

45. CPE: This instruction is used to call conditionally if there is even parity. (P=1)

46. CPO: This instruction is used to call conditionally if there is odd parity. (P=0)

47. RET: This instruction is used to return from subroutine unconditionally.

48. RC: This instruction is used to return from Subroutine conditionally if there is carry. (CY=1)

49. RNC: This instruction is used to return from subroutine conditionally if there is no carry. (CY=1)

50. RP: This instruction is used to return from subroutine conditionally if there is positive sign. (S=0)

51. RM: This instruction is used to return from subroutine conditionally if there is minus sign. (S=1)

52. RZ: This instruction is used to return from subroutine conditionally if there is zero. (Z=1)

53. RNZ: This instruction is used to return from subroutine conditionally if there is no zero. (Z=0)

54. RPE: This instruction is used to return from subroutine conditionally if there is even parity. (P=1)

55. RPO: This instruction is used to return from subroutine conditionally if there is odd parity. (P=0)

56. PCHL: This instruction is used to load program counter with HL registers contents.

57. RST: This instruction is used to restart the program.

Logical instruction:

58. CMP: This instruction is used to compare register or memory with accumulator.

59. CPI: This instruction is used to compare immediately with accumulator.

XTHL, DAD, ACI, ADI, DAA, SUI, SBI, RLC, CMA, STC, ANA

60. ANA: This instruction is used For logical AND register or memory with accumulator.
61. ANI: This instruction is used for logical AND immediate with accumulator.
62. XRA: This instruction is used for Exclusive OR register or memory with accumulator.
63. XRI: This instruction is used for exclusive OR immedia-te with accumulator.
64. ORA: This instruction is used for logical OR register or memory with accumulator.
65. ORI: This instruction is used for logical OR immediate with accumulator.
66. RLC: This instruction is used to rotate accumulator left.
67. RRC: This instruction is used to rotate accumulator right.
68. RAL: This instruction is used to rotate accumulator left through carry.
69. RAR: This instruction is used to rotate accumulator right through carry.
70. CMA: This instruction is used for complement accumulator.
71. CMC: This instruction is used for complement carry.
72. STC: This instruction is used to set carry.

Control instructions:

73. NOP: This instruction is used for no operation.
74. HLT: This instruction is used to disable interrupts.
75. EI: This instruction is used to enable interrupts.
76. RIM: This instruction is used for read interrupt mask
77. SIM: This instruction is used for set interrupt mask

Q. Interface two ~~4K into 8~~ 4K×8 ROM with 3:8 decode at the address of your choice. Show the address MAP for each ROM and explain your circuit.

Soln For ROM_1

Assume base address = 0000H

End address = base address + (No. of locations in 4KB ROM - 1)

= 0000H + (4 × 1024 − 1) D

= 0000H + (4096 − 1) D

= 0000H + 4095 D

= 0000H + 0FFF H

= 0FFFH

16	4095	F
16	255	F
	15	F

For 4KB size, the number of address lines N is given by

$2^N = 4 \times 1024$

$2^N = 4096$

$2^N = 2^{12}$

$\therefore N = 12$

i.e A_0 to A_{11} are povided to internal decoder of ROM_1.

For ROM_2:

~~Assume~~ base address = End address of ROM_1 + 0001H

= 0fffH + 0001H

= 1000H

End address = base address + (No. of locations in 4K ROM - 1)

= 1000H + (4 × 1024 − 1) D

= 1000H + (4096 − 1) D

= 1000H + 4095D

= 1000 H + 0 fff H

= 1fffH

Similarly N = 12

i.e $A_0 - A_{11}$ are provided to internal decoder of ROM2.

The address map for ROM1 and ROM2 is expressed as

ROM1

Base address : 0000H

End address : 0fffH

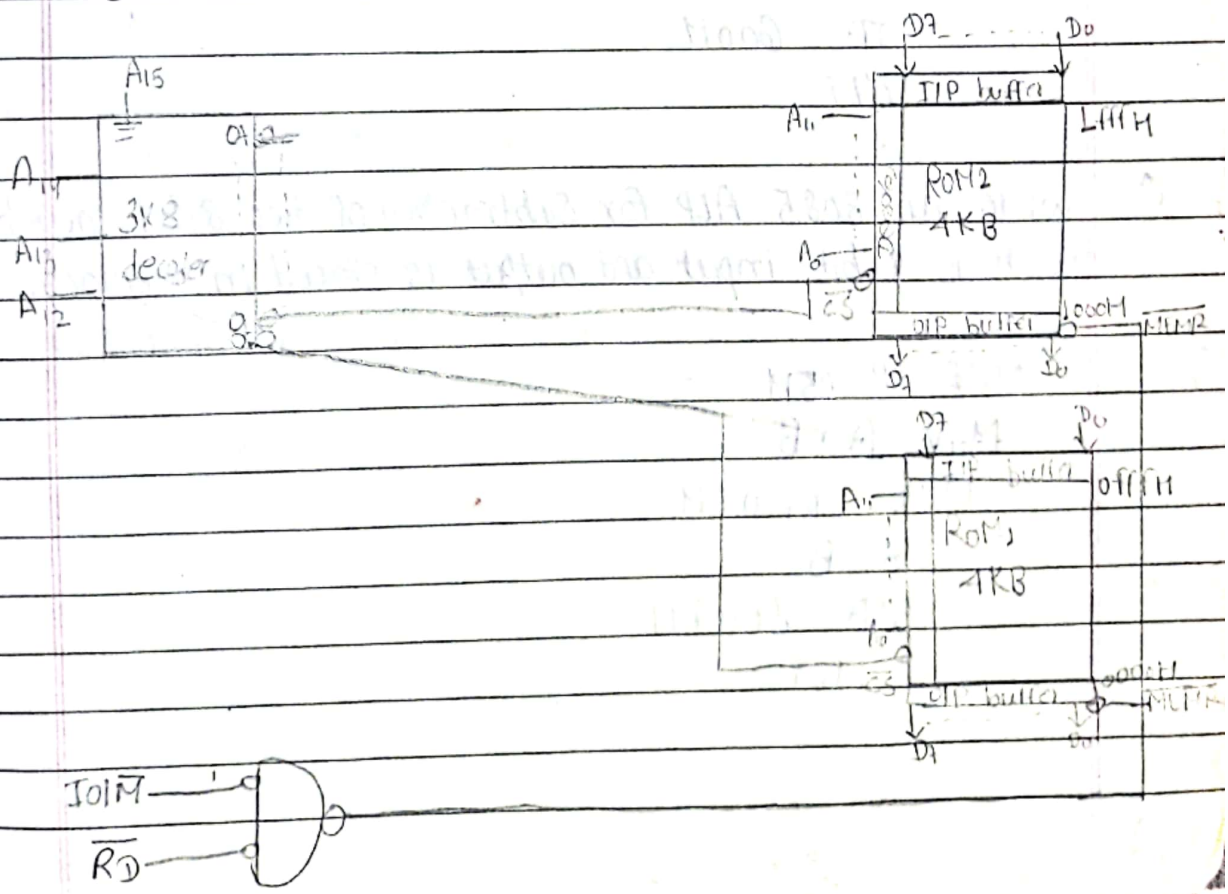
ROM2

Base address: 1000H

End address: 1fffH

	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
ROM1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1
ROM2	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1

Since we have to use 3:8 decoder it is required to take A_{14}, A_{13} and A_{12} as decoder inputs



Q. Write An ALP to add two decimal numbers

```
MVI H, 8000H
MOV A, M
INX H
ADD M
STA 8003H
HLT
```

Q. Write an 8085 ALP for addition of two 8 bit numbers and result is 8 bit. Input and output are stored in memory.

```
MVI B, 04H
MOV A, B
MVI B, 02H
ADD B
STA 600H
HLT
```

Q. Write an 8085 ALP for Subtraction of two 8 bit number and result is 8 bit. input and output is stored in memory.

```
MVI B, 05H
MOV A, B
MVI B, 07H
SUB B
STA 8004H
HLT
```

Q. write an ALP to add two number stored at 8000H and 8001H and store the result at 8005H

```
LDA 8000H
MOV A,B
LDA 8001H
ADD B
STA 8005H
HLT
```

Q. write an ALP to compare two numbers and display the larger one.

```
LDA 2800H
MOV B,A
LDA 2801H
  CMP B
JNC NEXT
  MOV A,B
NEXT: STA 2802H
  HLT
```

Q. write an ALP to multiply two numbers:

```
LDA 2800H
MOV B,A
LDA 2801H
  MOV C,A
  XRA A
NEXT: ADD B
  DCR C
JNC NEXT
STA 2802H
HLT
```

Q. write an ALP to swap two 8-bit number.

```
LDA 2800H
MOV B,A
LDA 2801H
STA 2800H
MOV A,B
STA 2801H
HLT
```

Q. write an ALP to convert two BCD numbers in memory to Hexadecimal number.

```
LXI H,4150H
MOV A,M
ADD A
MOV B,A
ADD A
ADD A
ADD B
INX H
ADD M
INX H
MOV M,A
HLT
```

Q. write an ALP to Find the largest number in an array of data.

```
LXI H,2500H
MOV B,M
```

```
INX H
MOV A,B
DCR B
LOOP: INX H
CMP M
JNC AHEAD
MOV A,M
AHEAD: DCR B
JNZ LOOP
STA 2508H
HLT
```

Q. Write an ALP to count the number of 1 in the given string 10100110. Display the result at 2605H.

```
MVI A,A6H
MVI B,00H
MVI C,08H
LOOP1: RAL
JNC LOOP2
INR B
LOOP2: DCR C
JNZ LOOP1
MOV A,B
STA 2605
HLT
```

Chapter 5:

I/O Interfacing Technique:

There are 2 technique through which I/O can be interfaced to microprocessor.

1. memory mapped I/O
2. I/O mapped I/O

1. Memory mapped I/O:

In memory mapped I/O, only one address space is used. This particular one address space is allocated to both memory and I/O devices. In total memory address, some addresses are assigned to memory and some to I/O devices.

It is a method in which total memory address is partation into two part i.e memory address space and I/O memory address space. I/O devices are treated here as memory location

2. I/O mapped I/O

In this method separate address space is given to I/O devices. I/O mapped I/O is a method to perform I/O operations between the CPU and Peripheral devices in a computer that uses two separate address space for memory and I/O devices. I/O mapped I/O uses separate instruction for read and write operation in I/O and memory.

IMP // Difference between memory mapped I/O and I/O mapped I/O.

	Memory mapped I/O	I/O mapped I/O
1.	Both devices (I/O / memory) are treated as memory	I/O devices are treated as I/O and memory as memory.
2.	Memory and I/O share the entire address range of processor.	Processor provides separate address space for memory and I/O devices.
3.	Device address is 20 bits.	Device address is 8/16 bits.
4.	Processor provides more address lines for accessing memory. Hence, more decoding is required	Processor provides less address lines for accessing I/O. Hence less decoding is required
5.	Control signals used are MEMR and MEMW.	control signals used for I/O are IOWR and IORD
6.	More hardware is required	less hardware is required.
7.	Arithmetic and logical operations can be ~~performed~~ perform on direct data from I/O devices	Arithmetic and logical operations are not possible with direct data from I/O devices.
8.	It is less efficient	It is more efficient.

DMA (Direct memory Access):

The transfer of data between the peripherals and memory without the interaction of CPU and letting the peripheral device manage the memory bus directly is termed as DMA.

The data transfer technique in which peripherals manage the memory buses for direct interaction with main memory without involving the CPU is called DMA. Using DMA technique, large amount of data can be transferred between memory and the peripherals

without severaly impacting CPU's performance. During DMA transfer, the CPU is ideal and has no control of the memory buses. A DMA contraler takes over the buses to manage the transfer directly between the I/O devices and main memory. The structure or block diagram of DMA controller is described below:

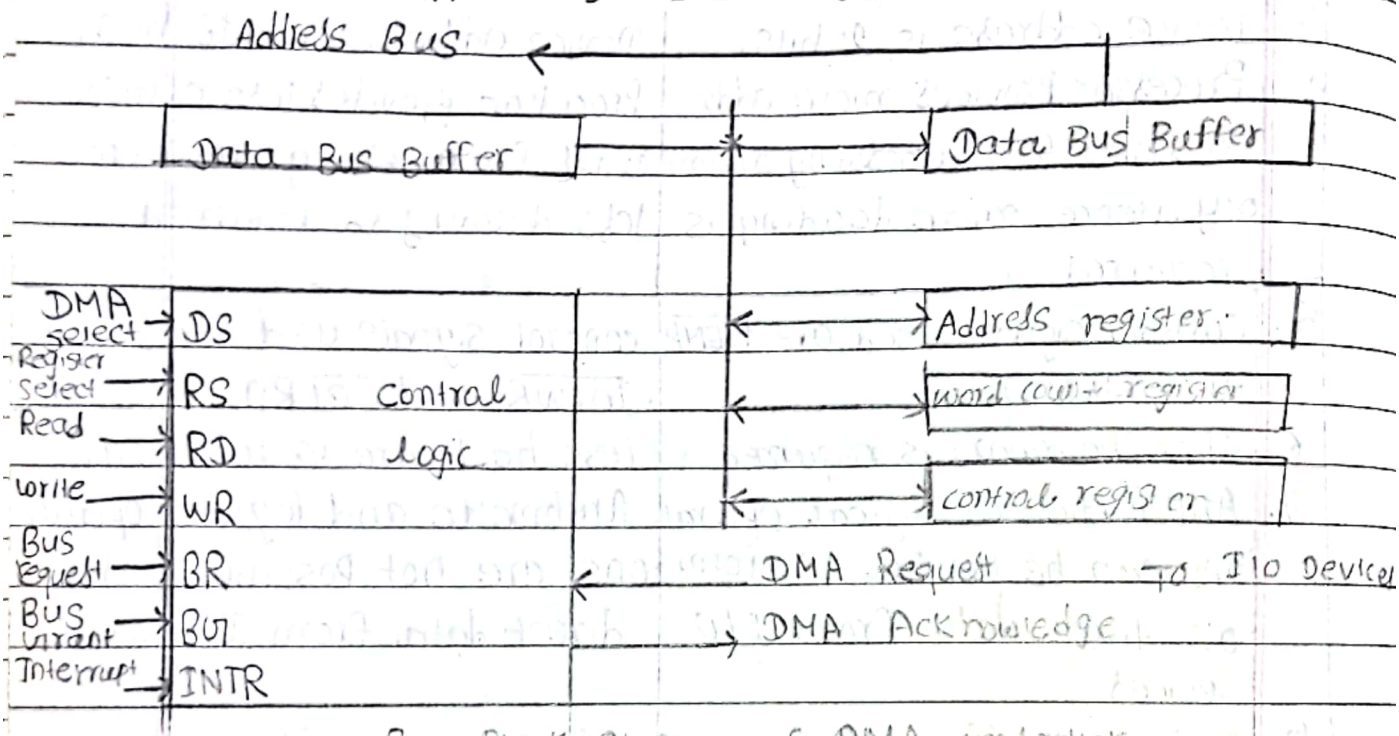


fig: Block Diagram of DMA controller

The control unit communicates with the CPU via databu and control lines. The DMA contras the system bus using BR (Bus request) and BG (Bus grant) signals. DM operates read and write operations via RD and WR signals. DMA sents requests and acknowledge to I/O devices via DMA requests and DMA acknowledge signals. The registers in DMA are selected by CPU throug the address bus by enabling by DS (DMA select) and RS (register select) inputs. All registers in the DMA appeared to the CPU as I/O interface registers. The address registers contains an address specify th

desire location in memory. It is incremented after each word that is transferred to the memory. The word count register holds the number of words to be transform. It is decremented by 1 after each word transferred and internally tested for 0. The control register specifies the mode of transfer.

IMP DMA transfer:/Sequence of events in DMA transfer:
The DMA controller requests CPU to handle control of buses to the DMA using bus request (BR signal). The CPU grants the control of buses to DMA using bus grant (BG) signals. After placing the address bus, data bus, read and write lines into high impedance state (which behaves like open circuit).

CPU initializes the DMA by sending following information through the data bus.

1. Starting address of memory block for read or write operation.
2. The word count which is the number of words in the memory block.
3. control to specify the mode of transfer such as read or write.
4. A control to start the DMA transfer.

The DMA takes control over the buses and directly interact with memory and I/o units to transfer the data without CPU intervention.

5. when the transfer completes, DMA disables the BR line. Thus CPU disables BG line, it takes control over the buses and returns to its normal operation.

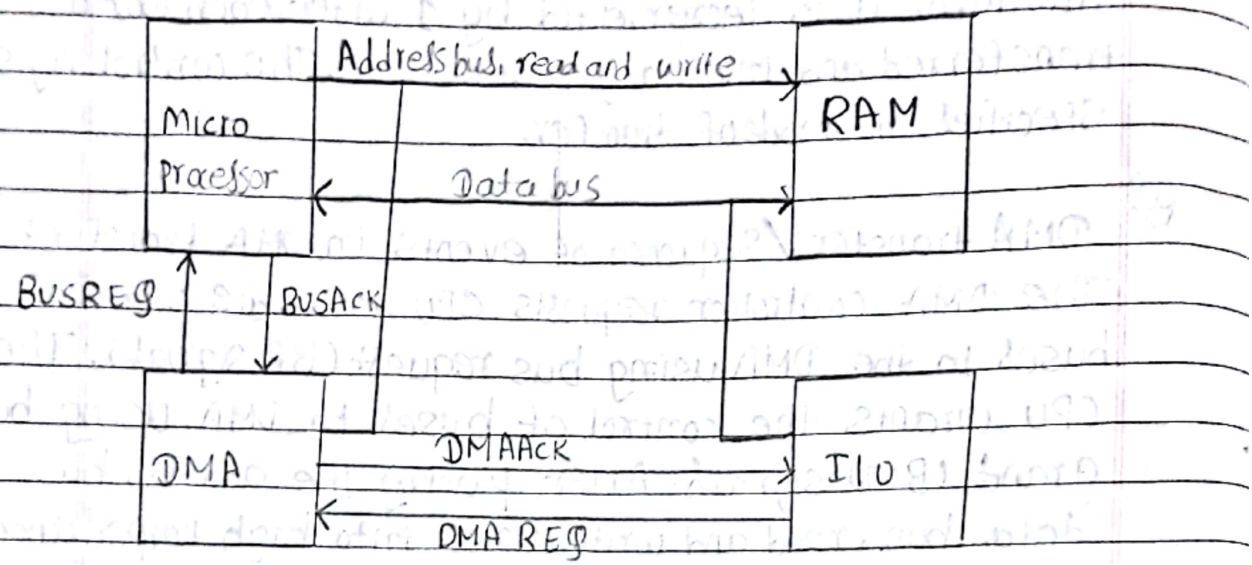
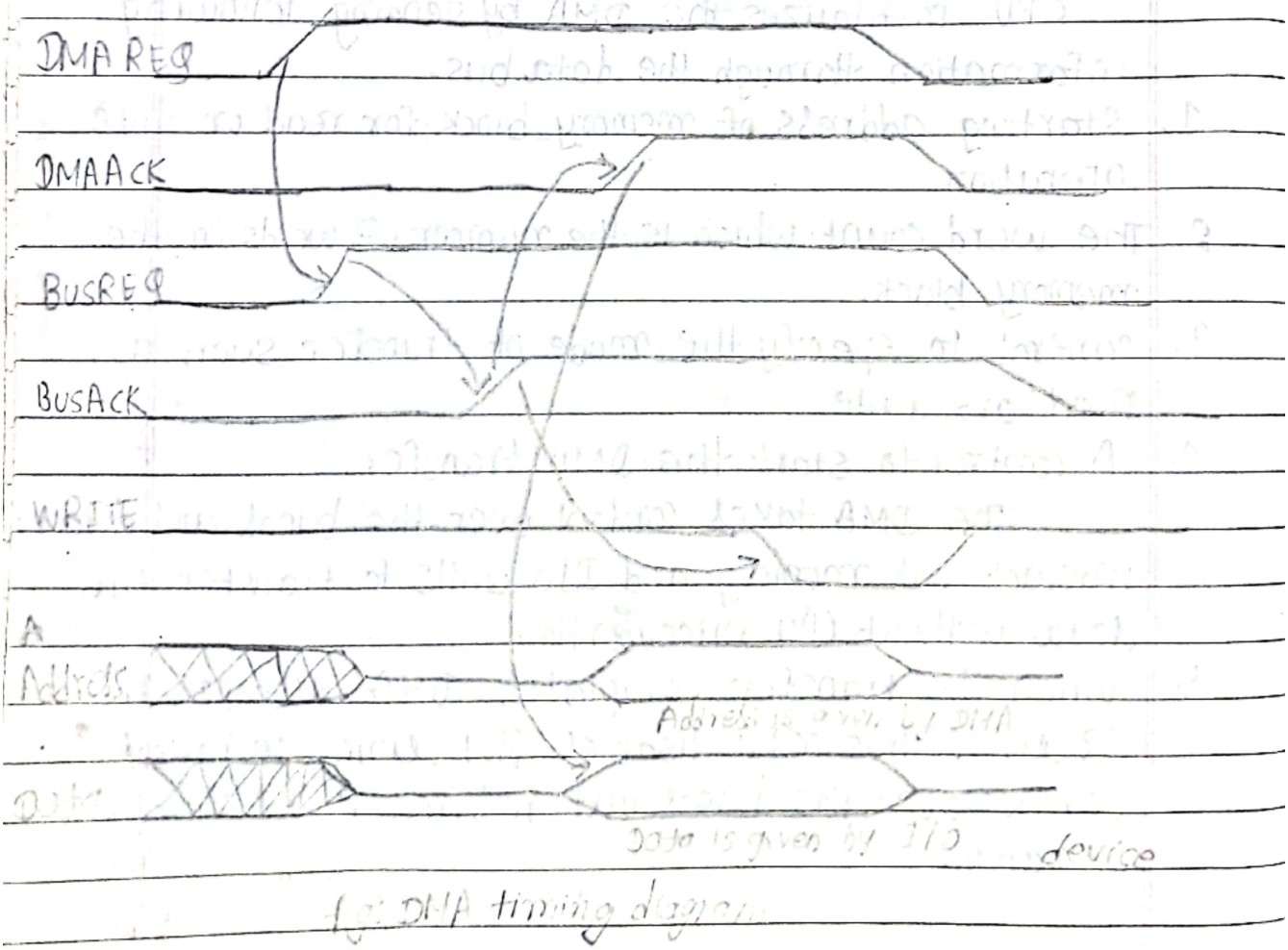


fig: Illustration for DMA transfer in a computer.



(c) DMA timing diagram

DMA transfers mode:

There are three modes of DMA transfer. They vary by how DMA controller determines when to transfer data.

(a) Burst mode:

In this mode, the entire block of data is transferred in one continuous sequence. once the DMA controller gets the access to the system, it transfer all bytes of data in the data block before relinquishing control of system buses back to the CPU. This mode is useful for loading programs or data files into memory, but it keeps CPU idle for relatively long period of time.

(b) Cycle stealing mode:

In this mode of transfer, DMA controller obtains access to system bus as in burst mode; transfers one byte of data and returns the control of the system bus to CPU. It continually issue requests using Bus request (BR) signals, transferring one byte of data per request, unit it has transferred its entire block of data. DMA controller steals the memory cycles from CPU for those stolen cycles, CPU remains idle. The data block is not transferred as quickly as in burst mode, but the CPU is not idled for long period of time as in ~~burst~~ burst mode.

(C) Transparent mode:

In this mode, the DMA controller only transfers data when CPU is performing operations that do not use system buses. The main advantage of this mode is that CPU never

Stops executing its program. The main disadvantage of this mode are:-

* Hardware needed to determine whether the CPU is not using the system buses can be quite complex and relatively expensive.
* Requires highest time to transfer a block of data as compared to above two modes.

Advantages of DMA

1. Quick data transfer because a dedicated piece of hardware transfers data from one computer location to another and only one or two bus read/write cycles are required per piece of data transfer.

2. Minimizes latency (delay) in servicing a data acquisition device because the dedicated hardware responds more quickly then interrupts and transfer time is short.

3. Minimized latency reduces the amount of temporary storage (memory) required on an I/O device.

4. Processor is not used for holding the transfer activity and is available for other processing activity.

5. Increases overall system utilization since data transfer actually occur in parallel.

Application of DMA:

1. DMA is extensively used for computer-based data acquisition applications including streaming data to disk, real time screen data display and continuous data acquisition applications.

2. DMA was used for floopy disk I/O in the original PC and for harddisk I/O in later versions.
3. Data acquisition application can take advantage of this technology.

Addressing modes:

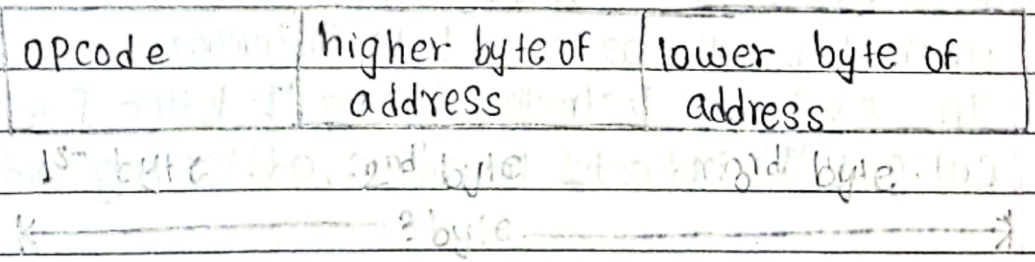
Each instruction requires certain data in which it has to operate. There are various technique to specify data for instructions. These technique are called addressing mode.

The different ways in which a processor can access data are referred as its addressing mode.

The various addressing modes are:

1. Direct addressing mode (Absolute addressing):

In this mode of addressing, the address of the operand (data) is given in the instruction itself. The instruction size is either 2-bytes or 3-bytes with first (1st) byte being opcode followed by 1 or 2 bytes of address of data.



Example:

STA 2400H: Store the content of accumulator to memory location 2400H.

LDA 3000H: Load the content of memory location 3000H into accumulator.

INT 02H: Input data from the input port 02H in accumulator (However it is 2 byte instruction)

LHLD 1000H: Load the content of 1000H address memory to L and 1001H memory to H.

JMP 4000H: Transfer the program sequence to memory location specified by 16 bit address (i.e 4000H)

Call 5000H: Change the program sequence to location of a Subroutine specified ~~16 bit~~ by 16 bit address.

2. Immediate Addressing Mode:

In this addressing mode, the operand is specified within the instruction itself. The term immediate implies that the data immediately follow the hexadecimal opcode in the memory.

Instruction	Source	Destination
MVI C, 3AH	Data 3AH	Register C

It transfers the source immediate byte/word of data in destination register or memory location. If the immediate data is of 8 bit then the whole instruction will be of 2 byte. If the immediate data is of 16 bit then the instruction will be a 3 byte instruction.

In 8085, the instructions having 'I' letter fall under this category. 'I' indicates immediate addressing mode.

Example:

MVI A, 30H: Moves 8 bit immediate data (30H) into the accumulator.

LXI D, 10ffH: Moves 16 bit immediate data (10FFH) into the DE-register pair such that D=10H and E=FFH.

ADI 06H; Add 06H to the contents of the accumulator.

MOV A, 90H: COPY 90H into A.

LXI H, 1234H; copy 1234H into H
Note: Immediate data are constant data.

3. Register addressing mode:
In this addressing mode, the instruction specifies the name of the register in which the data is available. It is the most common form of data addressing. In this mode it transfers a copy of byte/ word from source register to destination register. It is carried out with 8 bit register A, B, C, D, E, H & L. It is important to use register of same size. Never mix an 8 bit register with a 16 bit register i.e MOV A, SP.

Instruction	Source	Destination
MOV A, B	Register B	Register A

Examples:
MOV A, B: Move the content of register B to accumulator
ADD B; add the content of register B to the contents of accumulator.
SPHL: Move the content of HL register into Stack pointer

4. Register Indirect addressing mode:
In this mode, the instruction ~~specifies~~ specifies the name of the register pair in which the address of the data is available. Here the data will be in the memory and its 16 bit address will be seen in the 16-bit register pair
Examples:
MOV A, M; Move the contents of memory location pointed by HL register pair to accumulator.
LDA X.B; Load the accumulator with the content of memory location pointed by BC register pair.

ADD M; Add the content of memory location pointed by HL register to the content of accumulator.

5. Implied Addressing mode:
In implied addressing mode, the instruction itself specifies the data to be operated. These instructions do not specify the operand explicitely in the instruction itself but it is implicated.
Example:
CMA; complement of content of accumulator.
XCHG: Exchange the content of HL register pair with DE register pair.
STC: set carry flag
CMC: complementary carry flag.

8237 DMA controller and Interfacing:
It is a ~~multi~~ programmable DMA controller housed in a 40-pin package. It must intreface with two devices ~~(CPU)~~ MPU and peripherals (eg. floopy disk, Pendrive). It is a data transfer processor to peripheral devices.
The block diagram shows a logical pinout and internal registers of the ~~8287~~ 8237. It also shows the interface with the 8085 using a 3:8 decoder. 8237 has four independent channels CH_0 to CH_3. 16 bit registers are internally associated with each channel. In the 8 bit register from top, one stores starting address of byte to be copied and the next store the count. The next eight registers are accessed by MPU. The address of these register are determined by address lines, A_3 to A_0 and the chip select (CS)

~~DMA~~

DMA signals:

The 8237 signals are divided into two groups:

1. Signals on left (used to communicate with MPU)
2. Signals on right (used to communicate with peripherals).

* To obtain DMA service, a request is generated by activating the DREQ line of the channel.
* DACK are output lines to inform the individual peripherals that DMA is granted.
* AEN and ADSTB - Address Enable and Address strobe are used to latch a high order byte to generate a 16-bit address.
* After receiving the HRQ (Hold request), the MPU completes the bus cycle in process and issues the HLDA (Hold acknowledgement) signal

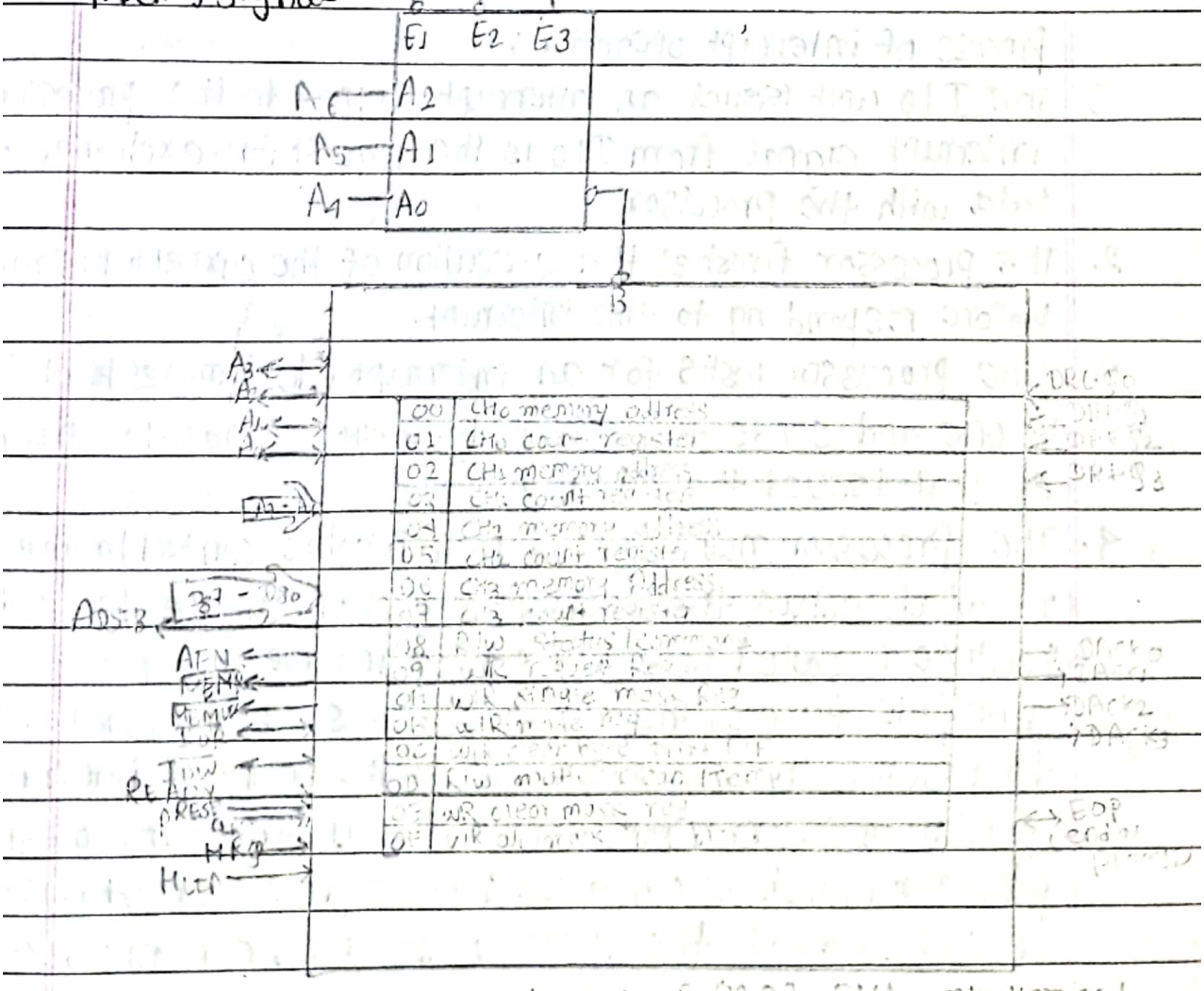


fig: block diagram of 8237 DMA controller and memory

Interrupt:

An interrupt is a signal that a peripheral board sends to the central processor in order to request attention. In response to an interrupt, the processor stops what it is currently doing and executes a service routine. when the execution of the ~~routine~~ service routine is terminated, the original process may resume its previous operation.

Interrupt are the signals generated by the external devices to request the microprocess to perform a task. There are 5 interrupt signals ie TRAP, RST 7.5, RST 6.5, RST5.5 and INTR.

Mainly in the µp based system, the interrupts are used for data transfer between the peripheral and the µp.

Process of interrupt operation;

1. The I/O unit issues an interrupt signal to the processor. An interrupt signal from I/O is the request for exchange of data with the processor.
2. The processor finishes the execution of the current instruction before responding to the interrupt.
3. The processor tests for an interrupt, determines that there is one and sends an acknowledgement signal to the device that issued the interrupt.
4. The processor now begins to transfer the control to the routine which serves the interrupt request from the device. This routine is called interrupt service routine.

For this process, the CPU needs to save information needed to resume the current program at the point of interrupt. The minimum information required is (i) The status of the processor, which is contained by the process or status word (PSW) and (ii) the location of the next instruction to be

executed which is contained by the program counter (PC) these all are pushed onto the stack.

5. The processor then loads the PC with the entry location of the interrupt service routine that will respond to this interrupt. once the PC has been loaded, the control is transferred to the interrupt handler program.
6. The fundamental requirement of the interrupt service routine is that it should begin by saving the contents of all the registers on the stack (as the state of main Program should be safe).
7. The interrupt handler now proceeds to process the interrupt.
8. when ~~the~~ interrupt processing is complete the saved registers value (of the main program) are retriered from the stack and restored to the register.
9. The final function is to restore the Psw and PC values from the stack. As a result, the next instruction to be executed will be from the ~~previ~~ previously interrupted main program.

Types of interrupts

There are three major types of interrupts that causes a break in the normal execution of a program. They are:

1. External interrupts.
2. Internal interrupts
3. Software interrupts.

1. External interrupts:

External interrupts are initiated via the µp's interrupt pins by external devices, I/o devices, timing devices

circuit monitoring the power supply etc. The causes of these interrupt may be I/O device requesting transfer of data, I/O device finished transfer of data, elapsed time of an event or power failure. External interrupt can be further divided into two types:

(a) Maskable interrupt
(b) Non-maskable interrupt

(a) Maskable interrupt:
A maskable interrupt is one which can be enabled or disabled by executing instructions such as EI (enable interrupts) and DI (disable interrupt): Eg: INTR, RST 7.5, RST 6.5, RST 5.5.

(b) Non-maskable interrupt.
A non-maskable interrupt is one which can not be enabled or disabled by executing instructions. This type has higher priority over the maskable interrupt. TRAP is an example of non-maskable interrupt.

2. Internal interrupt:
Internal interrupt arise from illegal or erroneous use of an instruction or data. The cause of this interrupt may be register overflow, attempt to divide by zero, an invalid operation code, stack overflow, etc.

3. Software interrupt:
A software interrupt is initiated by executing an instruction. Software interrupt is a special call instruction that behaves like an interrupt rather than a subroutine call. It can be used by the programmer to initiate an interrupt procedure at any desired point in the program. The software interrupts are RST0, RST1, RST2, RST3, RST4, RST5, RST6 and RST7

Vectored and Non-vectored interrupts:

A vectored interrupt is where the CPU actually knows the address of the interrupt service routine. In ~~advance~~ advance vectored interrupts are those which have fixed vector address (Starting address of sub routine).

Non vectored interrupts are those in which vector address is not predefined. INTR is the only non-vectored interrupt in 8085 µp. A non-vectored interrupts is where the interrupting device never sends an interrupt vector.

Interrupt priority:

An interrupt priority is a system that establishes a priority over the various sources to determine which condition is to be serviced first. When two or more requests arrive simultaneously, The system may also determine which conditions are premitted to interrupt the computer while another interrupt is being serviced.

Higher priority interrupt levels are assigned to requests which, if delayed or interrupted, could have serious consequences. Device with higher transfers speed such as magnetic disks are given high priority, and slow devices such as Keyboard receive low priority. When two devices interrupt the processor at the same time, the processor services the device with the higher priority first.

There are mainly two ways of servicing mutiple interrupts. They are:

(a) Polled interrupts.
(b) Chained (vectored) interupt.

(a) Polled interrupt:

Polled interrupt are slower compared to vectored 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 and polls the interrupts sources in sequence. The order in which they are tested determines the priority of each interrupt. The highest priority source is tested first and if its interrupt signal is on, control branches to a service routine for this source otherwise the next lower priority source is tested and so on. Thus the initial servicer outline for all interrupts consists of a program that tests the interrupts consists of a program that tests the interrupt sources in sequence and branches to one of many possible service routines.

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

(b) Chained (vectored) Interrupt:

In this technique the devices are connected in a chain fashion for setting up the priority system. 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 and then generates an interrupt acknowledgement signal (INTA) to the highest priority device, which is device 1. If this device has generated the interrupt it will accept the INTA signal from the processor; otherwise it will pass INTA signal from the processor on to the next device until the INTA is accepted by the interrupting device.

Once accepted, the device provides same answer to the

Processor for finding the interrupt address vector with the help of hardware it generates interrupt vector address.

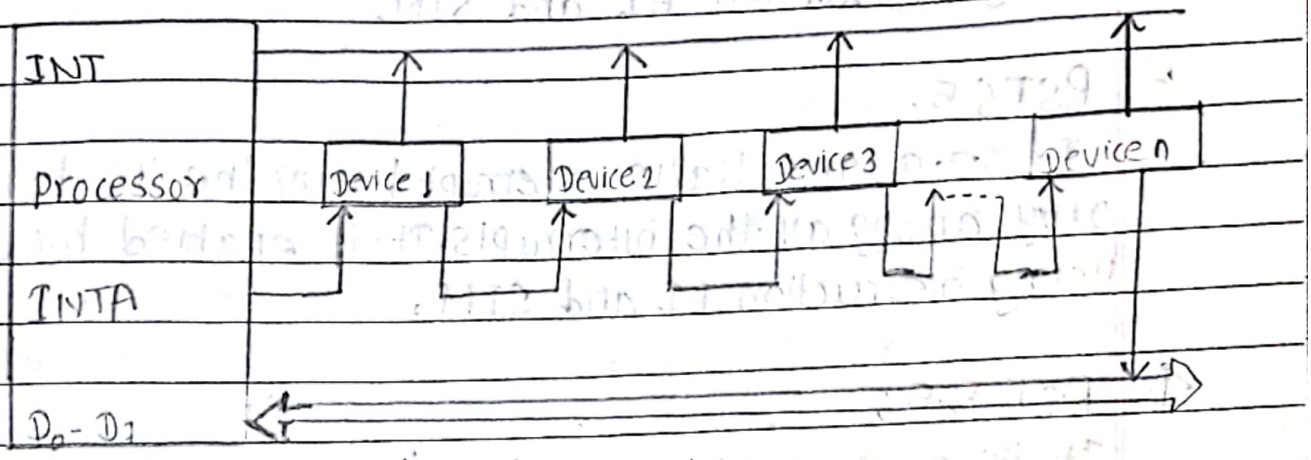


fig: chained interrupt

V.VI Interrupts of 8085 | Hardware interrupts:

The 8085 has five hardware interrupts:

1. TRAP — Highest priority
2. RST 7.5
3. RST 6.5
4. RST 5.5
5. INTR — lowest priority

1. TRAP:

It is a non maskable interrupt having the highest priority among all interrupts. It need not to be enabled and it can not be disabled. When this interrupt is triggered, the program control is transferred to the location 0024H without any external hardware or the interrupt ~~ean~~ enable instruction. TRAP is generally used for such critical events as Power failure and emergency shut off.

2. RST 7.5:

It is a maskable interrupt having the second highest priority among all interupts. It is enabled by software using instruction EI and SIM.

3. RST 6.5:

It is a maskable interrupt having the third highest priority among all the interrupts. It is enabled by software using instruction EI and SIM.

4. RST 5.5;

It is a maskable interrupt having the fourth highest Priority among all interrupts. It is enabled by software using instruction EI and SIM.

5. INTR:

It is a maskable interrupt having lowest priority among all interrupts. It can be enabled by instruction EI and can be disabled by instruction DI. The INTR interrupt requires external hardware to transfer program sequence to specific CALL locations.

Non maskable interrupt VS maskable interrupt (NMI VS MI)

	NMI	MI
1.	It cannot be ignored or masked	It can be ignored or masked.
2.	It helps to handle higher priority tasks.	It helps to handle lower priority tasks.
3.	Its response time is low	Its response time is high
4.	It is used for emergency purpose e.g: Power failure, smoke detector etc.	It is used to interface with peripheral device.

5.	All are vectored interrupts	It may be vectored or non-vectored.
6.	TRAP is an example of NMI	RST 7.5, RST 6.5, RST 5.5, INTR are example of MI

Software interrupt VS Hardware interrupt:

	Software interrupt	Hardware interrupt
1.	It is a synchronous event	It is an asynchronous event.
2.	An interrupt that is requested by executing an instruction (Softw	An interrupt that is requested by execuled device.
3.	PC is incremented	PC is not incremented
4.	It can not be ignored or masked	It can be masked
5.	It has highest priority among all interrupts.	It has lowest priority than software interrupts.
6.	It is used in program debugging and does not improve system throughput.	It is used to interface peripheral in interrupt driven. I/O and improve as throughput of the system.

External interrupts VS Internal interrupt

	External interrupt	Internal interrupt
1.	The interrupts generated by the hardware circuits outside the CPU are known as external interrupt.	The interrupt generated within the CPU are known as internal interrupt
2.	External interrupts are asynchronous.	Internal interrupts are synchronous.
3.	These are basically hardware interrupts.	These maybe hardware or software interrupts.
4.	This interrupt is triggered by external hardware module.	This interepts are triggered by software instruction.
5.	Example: A power sensing circuit may generate an external interrupt	Example: (a) Divide by zero (b) Stack overflow (c) protection violation.

SIM (set interrupt mask):
This is a 1 byte instruction and can be used to implement the 8085 interrupts (RST 7.5, 6.5 and 5.5) and serial data output. The instruction interprets the accumulator contents as follows

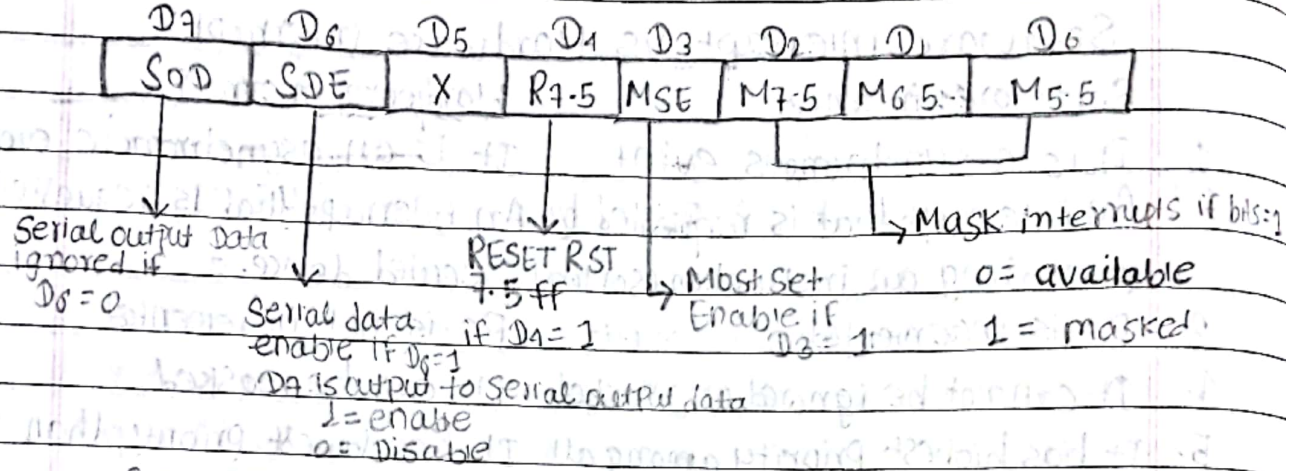


fig: SIM instruction

The various function of SIM instructions are

1. To enable or disable the interrupts Rst 7.5, 6.5 and 5.5, logic 0 on bits D0, D1 and D2 will enable the corresponding interrupts and logic 1 will disable (most) the interrupts. Bit D3 is a control bit and should be at logic 1 for bits D0, D1 and D2 to be effective
2. To reset RST 7.5 flipflop. Bit D4 (R7.5) is additional control for RST 7.5. If D4 = 1, RST 7.5 is reset and ignored without being served.
3. To implement Serial I/O. Bit D6 and D7 are used for serial I/O and do not affect the interrupts.

RIM (Read interrupts mask):
As there are several input lines when one interrupt request is being served other interrupt request may occur and remain pending. The 8085 has an instruction called RIM to sense those pending interrupts. It is a 1 byte instruction and used

to read the status of the interrupts RST 7.5, 6.5 and 5.5 and to read the serial input data bit. The accumulator contents for the RIM instruction is shown as:

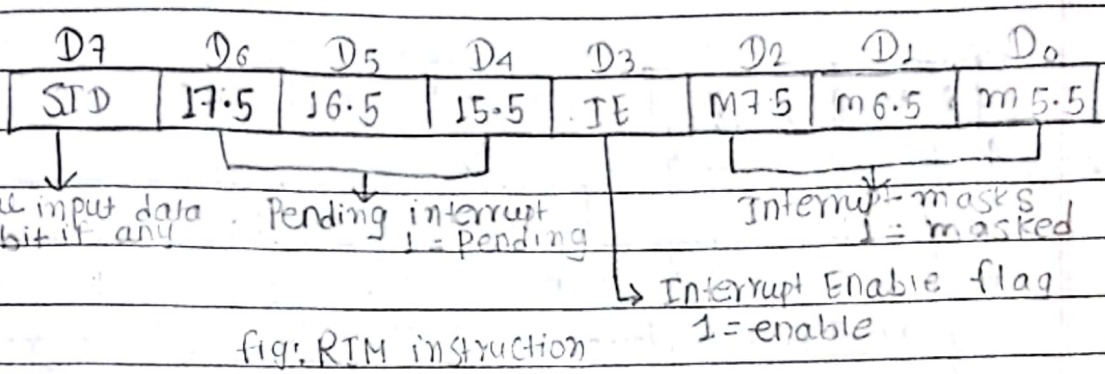


fig: RIM instruction

The RIM instruction can be used for the following purpose:

1. To read interrupt masks. Bit D0, D1 and D2 indicate the current status of the interrupt masks.
2. To identify pending interrupts. Bit D4, D5 and D6 identify the pending interrupts.
3. To receive serial data Bit D7 is used to receive serial data.

8259 Programmable interrupt controller (PIC):

Salient features:

1. PIC 8259 is a programmable interrupt controller that can be work with 8085, 8086 etc.
2. It is used to increase the number of interrupts.
3. A single 8259 provides 8 interrupts while a cascaded configuration of 1 master 8259 and 1 slave 8259s can provide up to 64 interrupts.
4. 8259 can handle edge as well as level triggered interrupts
5. In 8259 interrupts can be masked individually.
6. 8259 has to be compulsorily initialized by giving commands to decide several properties such as vector numbers, priority masking, Triggering etc.

7. In a cascaded configuration, each 8259 has to be individually initialized, master as well as each slave

Architecture or block diagram of 8259 PIC:

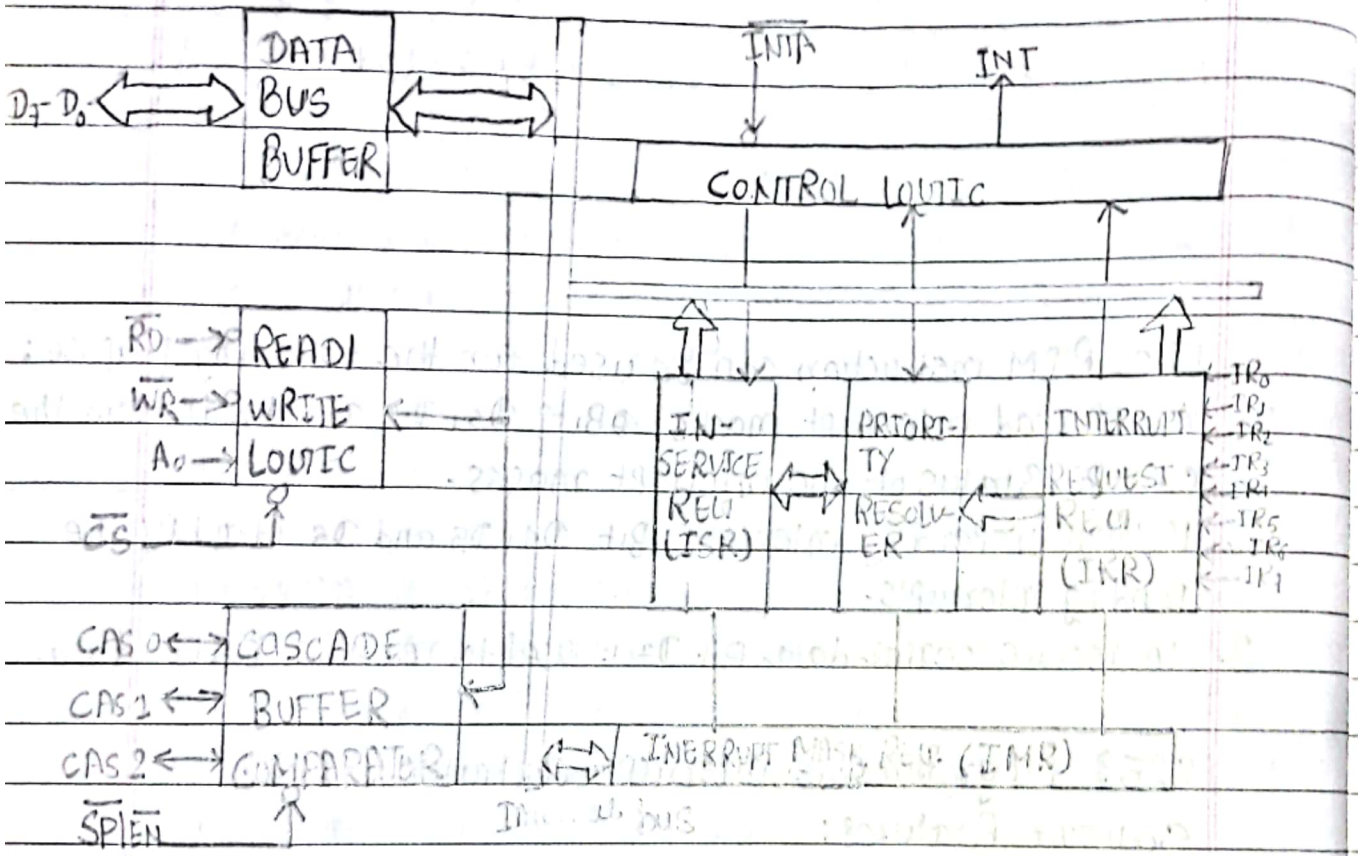


Fig: Architecture of 8259 PIC

The figure shows the internal block diagram of 8259A. It includes eight blocks.

1. Interrupt Request Register (IRR):

8259 has 8 interrupt input lines IR_7 ---- IR_0. The IRR is an 8 bit register having one bit for each of the interrupt lines when an interrupt request occurs an any of these lines, the corresponding bit is set in the interrupt request register (IRR).

2. In-Service Register (ISR or In-SR):
It is an 8 bit register which store the level of the interrupt request which is currently being serviced.

3. Interrupt Mask register (IMR):
It is an 8 bit register which stores the masking pattern for the interrupt of 8259. It stores one bit per interrupt level.

4. Priority Resolver:
It examines the IRR, InSR and IMR and determines which interrupt is of highest priority and should be sent to the µp.

5. Control Logic:
It has INT output connected to the INTR of the µp. to send the interrupt to µp. It also has the $\overline{INTA}$ input signal connected to the $\overline{INTA}$ of µp, to receive the interrupt acknowledge. It is also used to control the remaining blocks.

6. Data Bus Buffer:
It is a bidirectional buffer used to interface the internal data bus of 8259 with the external (system) data bus.

7. Read/write logic:
It is used to accept the $\overline{RD}$, $\overline{WR}$, A0 and CS signal when the address line A0 is at logic 0 the controller is selected to write a command or read a status. The chip Select logic and A0 determine the port address of the controller.

8. cascade Buffer/comparator:
It is used in cascade mode of operation. This block is

used to expand the number of interrupt levels by cas ding two or more 8259 AS. It has two components:

(i) CAS_2, CAS_1, CAS_0 lines:
These lines are output for the master, input for slave The master sends the address of the slave on these lin (hence output). The slaves read the address on those lines (hence input). As there are 8 interrupt levels for the mas ter, there are 3 CAS lines ($2^3=8$)

(ii) $\overline{SP}$/$\overline{EN}$ (Slave program/master Enable):
In Buffered mode, it functions as the EN line and is used to enable the buffer. In non-buffered mode, it functions as the SP output line. For master 8259, SP should be high and for the slave SP should be low.

Priority modes of 8259:

1. Fully Nested mode (FNM):
This is a general purpose mode in which all interrupt requests are arranged from highest to lowest, with IR_0 as the highest and IR_7 as the lowest.
In addition, any IR can be assigned the highest prio in this mode. The IR with the smallest value has highest priority.

2. Automatic Rotation mode:
This is a rotating Priority mode. It is preferred when severa interrupt sources are of equal priority. In this mode, a device after being served it gets the lowest priority. Assu IR_2 has just been serviced, it will get the lowest priority.

3. Specific Rotation mode:

It is also ~~set~~ a rotating priority mode but here the user can select any IR level for lowest priority, that fixing all other properties

Additional features of the 8259A:

The 8259A is a complex device with various modes of operation

(a) Interrupt Triggering:

The 8259A can accept an interrupt request with either edge triggered mode or level triggered mode

(b) Interrupt Status:

The status of the three instruction registers (IRR, ISR and IMR) can be read and this status information can be used to make the interrupt process versatile.

(c) Poll method:

The 8259A can be set up to function in a polled enviroment. The MPU polls the 8259A rather than each peripheral.

Q Difference both CALL and JUMP instructions

JUMP Instruction	CALL instruction
1) By using Jump, the Program control transfers to a location which is also a part of the main program.	1) By using CALL instruction, the program control transfer to a location which is not a part of main Program
2) Here the addressing mode is immediate	2) Here the addressing mode is immediate and Register Indirect.
3) We do not need to initialize the stack Pointer (SP) to perform the JUMP instruction.	3) We have to initialize the stack Pointer (SP) before using Some CALL instruction.
4) The Program Counter value is not pushed into stack before going to the pointed location.	4) The value of Program Counter is pushed into stack before going to the pointed location.
5) 10 T-states are required to perform JUMP instruction	5) 18 T-states are required to perform CALL instruction
6) 3 machine cycles are needed for JUMP.	6) 5 machine cycles are need for CALL
7) The value of stack Pointer remain unchange	7) The value of stack Pointer is decremented by 2.

Q. Differences bet^n 8085 and 8086 microprocessor

	8085 microprocessor	8086 microprocessor
1.	The data bus is of 8 bits.	1) The data bus is of 16 bits
2.	The address bus is of 16 bits.	2) The address bus is of 20 bits.
3.	The memory capacity is 64KB. Also it can perform operation upto 2^8 i.e. 256 numbers.	3) The memory capacity is 1MB. Also it can perform operation upto 2^16 i.e. 65,536 numbers.
4.	The operating frequency is 3.2 MHz	4) The operating frequency is 5 MHz, 8 MHz, 10 MHz.
5.	8085 up has single mode of operation.	5) 8086 up has two modes of operation.
6.	It dont have multiplication and division instructions.	6) It have multiplication and division instructions.
7.	It doesnot support pipelining.	7) It supports pipelining as it has two independent units Execution unit (EU) and Bus Interface unit (BIU)
8.	It doesnot support instruction queue.	8) It supports instruction queue.
9.	Memory space is not segmented.	9) Memory space is segmented.
10.	It consists of 5 flags (sign flag, zero flag, Auxiliary carry flag, Parity flag, carry flag).	10) It consists of 9 flags (carry flag, Parity flag, Auxiliary carry flag, Sign flag, Zero flag, Trap flag, Interrupt flag, Direction flag, overflow flag).

Interrupt Service Routine (ISR)

→ In Computer Systems Programming, an interrupt handler, also known as an interrupt Service Routine (ISR), is a special block block of code associated with a specific interrupt condition.

→ In general, ISR are the portions of the program code that handle the interrupt

- It is a subroutine. 8085 calls an ISR in response to an interrupt request by an external device.
- ISR must be located in memory at predetermined addresses known as interrupt vectors.

Features of 8085 microprocessor

1. It is an 8-bit microprocessor i.e. it can accept, process, or provide 8-bit data simultaneously.
2. It operates on a single +5V power supply connected at V_{cc}; Power Supply ground is connected to V_{ss}.
3. It operates on clock cycle with 50% duty cycle.
4. It has on chip clock generator. The internal clock generator divides oscillator frequency by 2 and generates clock signal, which can be used for synchronizing external devices.
5. It can operate with a 3MHz clock frequency
6. It has 16 address lines, hence it can access (2^{16}) 64 kb of memory
7. It provides 8 bit I/O address to access (2^8) 256 I/O ports.
8. In 8085, the lower 8 bit address bus (A_0-A_7) and data bus (D_0-D_7) are multiplexed to reduce number of external pins.
9. It has 8-bit accumulator, array of register (six 8-bit general purpose registers (B, C, D, E, H and L) and two 16-bit registers (SP & PC), it also consists of arithmetic logic unit, the encoder/decoder and timing & control circuits linked by internal data bus.
10. It have flag register, serial I/O control and interrupt control.

Features of 8086

→ It was the first 16-bit processor having 16-bit ALU, 16 bit registers, internal data bus, and 16-bit external data bus resulting in faster Processing.

→ It has an instruction queue, which is capable of storing six instruction bytes from the memory resulting in faster processing.

→ It is available in 3 versions based on the frequency of operation.
- 8086 → 5 MHz
- 8086-2 → 8 MHz
- 8086-1 → 10 MHz

→ It uses two stages of pipelining, i.e. Fetch Stage & Execute Stage, which improves performance.

→ Fetch Stage can pre fetch upto 6 bytes of instructions & stores them in queue

→ Execute stage executes those instructions.

→ It has 256 vectored interrupts.

→ It consists of 29,000 +- transistors.

→ It also have multiplication & division instruction.

→ It has a 20 bit address bus, so it can address 2^{20} or 1,048,576 memory location.

→ Memory Space is segmented i.e. 16 bit words will be stored in two Consecutive memory locations.