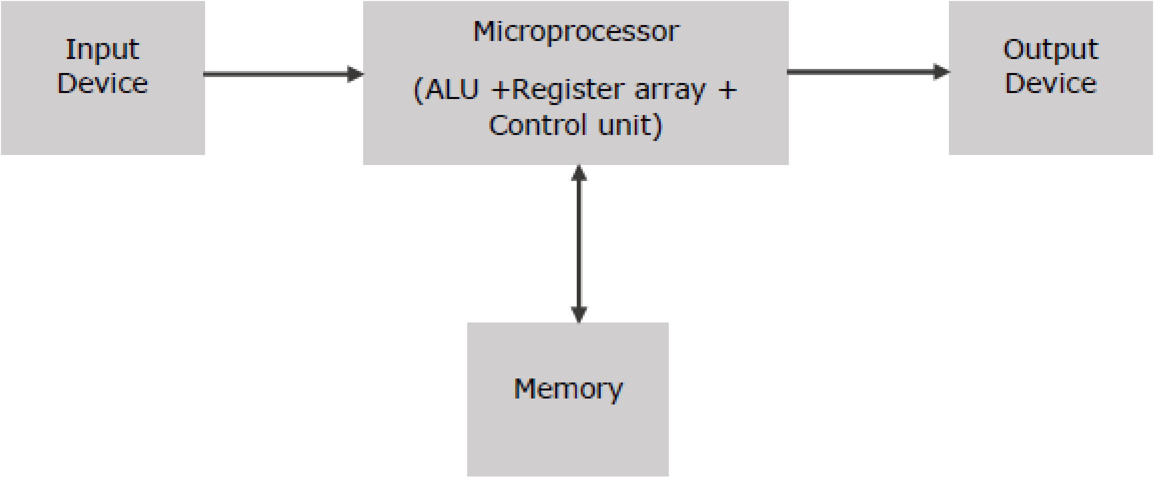
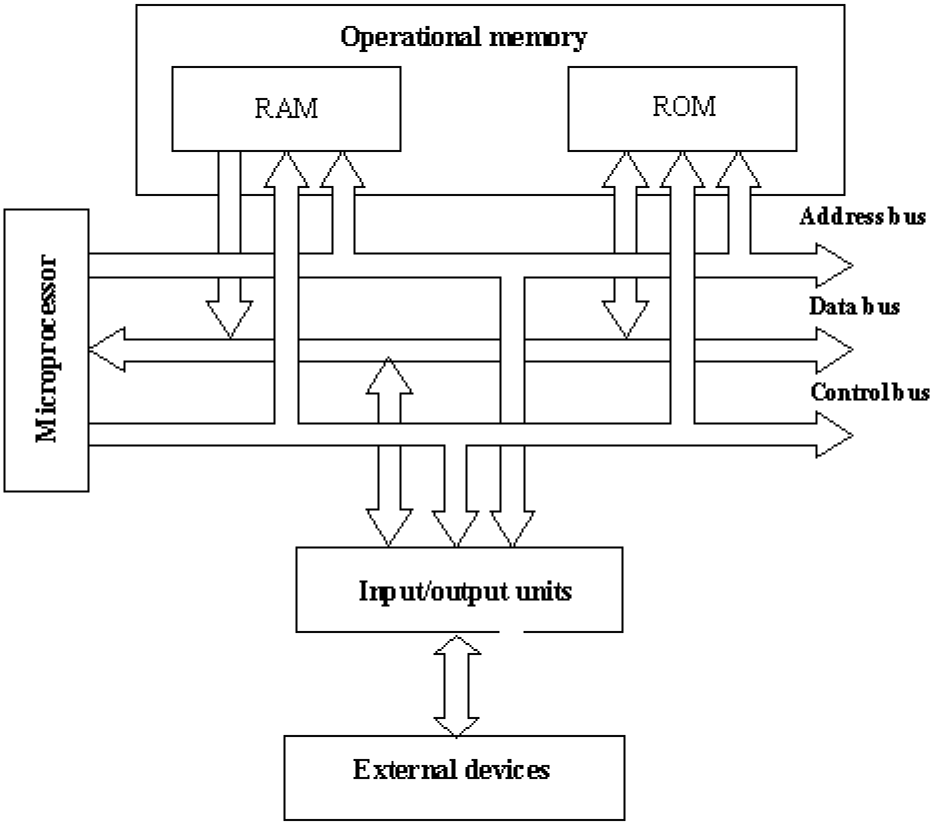
UNIT-1

Microcomputer System



A microcomputer is a computer built on the basis of a microprocessor i.e. a processor implemented as an integrated circuit. Since all processors are now produced in the form of integrated circuits, we can say that all computers are microcomputers. The general method for constructing microcomputers consists in connecting to the microprocessor busses additional sub-systems such as memories and peripheral device controllers (input/output units).

The basic block diagram of a simple microcomputer is shown in the figure below. We can see there a microprocessor with three its busses going out: data bus, address bus and control bus. To these busses, the following devices are connected: operational memory composed of RAM (Random Access Memory)and ROM (Read Only Memory) memories, as well as input/output units to which peripheral devices are connected.



Simplified general scheme of a simple microcomputer

MICRO-PROCESSORS

MICRO-PROCESSORS :- Micro processor is nothing but CPU which is the essential part of the computer It is a silicon chip that comparises million of transistors and other components that process million of instructions / second.

A micro-processor is a versatile chip which is pre-programmed by a software. It accepts digital data as input and processes it according to the instruction stored in memory.

The microprocessor has many functions like data storage, interact with other devices and time releated function, but main work is to send and receive the data to make the function of computer well.

FEATURES OF 8085 Micro-Processor

1. 40 pin I.C

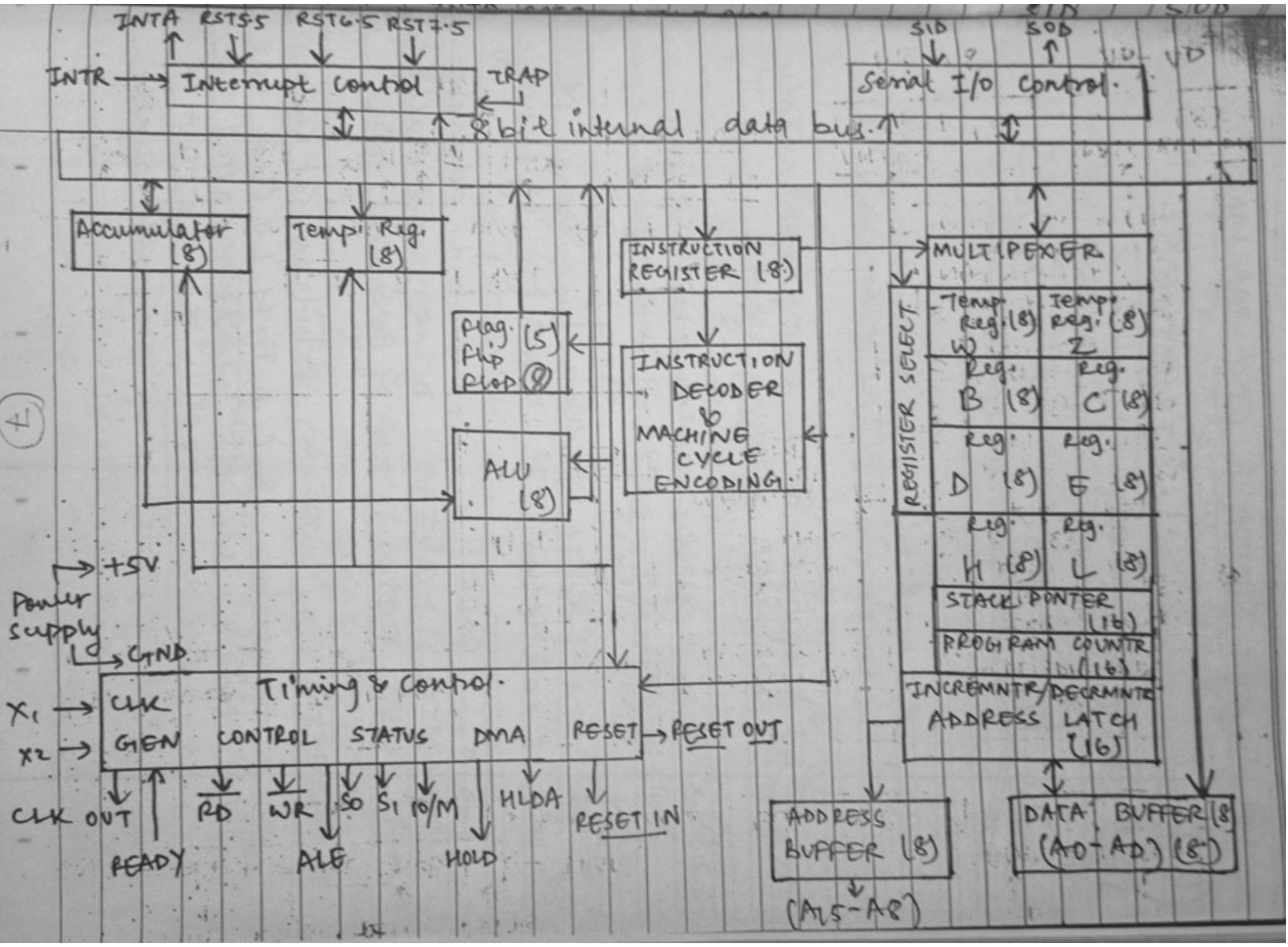
2. N-MOS Technology based
3. +5 V power supply.
4. Clock speed - 3.5 MHz
5. 8 bit Micro processor

6. 8 bit data bus, 16 bit address bu

7. Address data bus Multiplexing

8. 5 interrupts [TRAP; RST 7.5, RST 6.5; RST 5.5, INTR]

9. 5 flags.
10. 64 KB Memory.
11. No. of 8 bit Register - 7
12. No. of 16 bit " - 2.
13. 3 Register pairs.


INTA
RST5.5
RST6.5
RST7.5
INTR
Interrupt control
TRAP
SID
SOD
Serial I/O control.
8 bit internal data bus.
Accumulator (8)
Temp. Reg. (8)
Flag. flip flop (5)
ALU (8)
INSTRUCTION REGISTER (8)
INSTRUCTION DECODER & MACHINE CYCLE ENCODING
MULTIPEXER
REGISTER SELECT
Temp. Reg. (8) W
Temp. Reg. (8) Z
Reg. B (8)
Reg. C (8)
Reg. D (8)
Reg. E (8)
Reg. H (8)
Reg. L (8)
STACK POINTER (16)
PROGRAM COUNTR (16)
INCREMNTR/DECRMNTR ADDRESS LATCH (16)
+5V
Power supply
GND
Timing & Control
X1
X2
CLK GEN
CONTROL
STATUS
DMA
RESET
RESET OUT
CLK OUT
READY
RD
WR
ALE
S0
S1
IO/M
HOLD
HLDA
RESET IN
ADDRESS BUFFER (8)
(A15-A8)
DATA BUFFER (8)
(AD-AD) (8)

→ It provides 6 interrupts →
TRAP, RST 5.5, RST 6.5, RST 7.5,
INTR.

⇒ It provides control signals to control the Bus cycles.

[8085 Block diagram already made]
↓

General Purpose Registers:-

→ Registers B, C, D, E, H, L, are all general purpose register; they all are 8-bit registers.
But, they can be used in pairs like:- B-C, D-E, H-L, we can store 16-bit data in Register pair where higher bites are stored in Register B, D, [H] and lower bites in C, E, [L] respectively.

1. # → Temporary Registers:- programmers can't access this temp. registers W & Z are temp. registers which are not available for programmer.

→ Special Registers:- Accumulator:-

: It is a 8-bit register used in all arithematic and logic operations. [In these operations, accumulators hold one operand. and after execution, it ~~[illegible]~~ /holds the answer.

Flag Registers :- It is an 8-bit register, which provides the → information of the arthematic and logic operations., the format of flag Register is:—

S	Z	X	AC	X	P	X	Cy.

→ The five Status flag of 8085 are :—

① Carry Flag (Cy) :-

② Parity Flag (P) :-

③ Auxilary Carry Flag.(AC) :-

④ Zero Flag (Z) :-

⑤ Sign Flag (S) :-

ex: (9B+75) → convert to Binary & add

```
  10011011      → Carry flag = Set
+ 01110101
 [1]00010000    (we dont account the
                 carry in Parity
                 counting.)
```

Carry Flag bit: This particular bit is set, if there is a carry from MSB position, during an addition operation or if there is a ~~bow~~ borrow from subtraction operation, otherwise, it's a reset.

Parity Flag-bit:- The parity flag is set if the result of an operation contains even no. of 1's, otherwise it is Reset. ~~Auxilary carry flag~~ Bit:- ~~This~~

Auxilary carry-bit set:- It is set if there is a carry from D_3 bit to D_4 bit of the accumulator during the process of executing operation connected with a accumulator., otherwise it is Reset.

especially this flag is useful for DAA

(Decimal Adjust accumulator)
Instruction:

Zero- flag bit is set, if the result of an operation is zero(0) otherwise it's reset

Sign-flag:- This flag is set, if the MSB of the result is 1, other wise it's reset. used in signed operations.

INSTRUCTION REGISTER: This register holds opcode of the instructions and these opcode further send to Instruction decoder.

~~INSTRUCTIONS~~ PROGRAM COUNTER:-

This is of 16 bits, ~~it~~ ~~stores~~ it is used to hold the memory address of the next instruction to be executed, it keeps the track of memory addresses of the instructions in a program, while they are being executed.

The micro-processor in increments the content of progzam counter during an execution of instructions.

Stack pointer :- This Register is of 16-bit.

STACK :- The stack is a sequence of memory locations set aside by a programmer to store or retreive the contents of the accumulator, flags, Program Counter and general purpose resistors during the execution of a program.

Any portion of the memory can be used as a stack.
STACK works on (LIFO) principle.

During the execution of a program some times its becomes necessary to save the contents of some registers, which are needed for some other operations.

The contents of such registers are saved in the stack, then the Registers are used for some other operations.

→ The stack pointer controls the addressing of the stack, it holds the address of the top element of the data Stored in the stag.

⑧ # ALU :- It performs arithmatic and logical operations like ANDing, ORing, XORing, addition, subtraction, etc.

It is not accessible by user, and its of 8-bit, and its always controlled by timing and control circuits.

→ It provides, status or result of flag register.

TIMING & CONTROL :- Very imp.

unit as it synchronises the registers and flow of data through various registers and other units.

This unit consists of an oscillator and controller sequencer, which sends control signals needed for internal and external control of data and other units.

Signals associated with timing and control are -

(1) Control :- Ready., $\overline{WR}$, $\overline{RD}$, ALE

(2) Status :- S0, S1, $IO/\overline{M}$

(3) DMA :- HLDA, HOLD

(4) RESET: RESET IN, RESET OUT

~~INSTRUCTION~~ INTERRUPTS :-

Processor fetches and decodes & execute instructions in a sequence, but when special condition exists within a program, processor have to process these special conditions, after that it came back to its regular operations.

Occurence of this special condition is known as interrupt.
5 types of interrupts

(1) TRAP (2) INTR (3) RST 7.5 (4) RST 6.5
(5) RST 5.5

$\overline{INTA}$ → It is a interrupt acknowledge signal.

SERIAL I/O control:- when data is transmit bit by bit then serial communication is used.

Provides 2 signals:-

(1) SID (serial input data)
(2) SOD (serial out data)

SOD is used to send data serially and SID is used to receive data ~~d~~ serially.

—x—

INPUT, OUTPUT and MACHINE CONTROL INSTRUCTIONS:

(1) # IN, 8 bit port Address. -

Input to Accumulator from input, output port

ex: IN, 02H

[02] → A.

OUT, 8 bit port address.

0000 to FFFF] OUT, OFH.

[A] → [OFH]

Power Supply and clock frequency :-

V_{CC} = +5 V power Supply

V_{SS} : Ground reference

X_1, X_2 = crystal frequency. This freq. is internally divided by 2, ∴ to operate a system at 3 MHz, crystal should have a frequency of 6 MHz.

CLK (OUT) = (Clock output). This signal can be used as the system clock for other devices

5) Externally Initiated signals, including interrupts :-

- These can be used to interrupt a program execution.

INTR (Input) → Interrupt Request → This is used as a general-purpose interrupt

$\overline{\text{INTA}}$ - (output) → Interrupt Acknowledge → This is used to acknowledge an interrupt.

2 RST 7.5 (I/P) → Restart Interrupts :- These are vectored interrupts
RST 6.5 that transfer program control to specific m/ry locⁿ.
RST 5.5 They have higher prio. then 6.5 & then 5.5

Trap → (I/P) → Nonmaskable Interrupts. & has the highest priority.

HOLD (I/P) → This signal indicates that peripheral (DMA) is requesting the use of address & data buses.

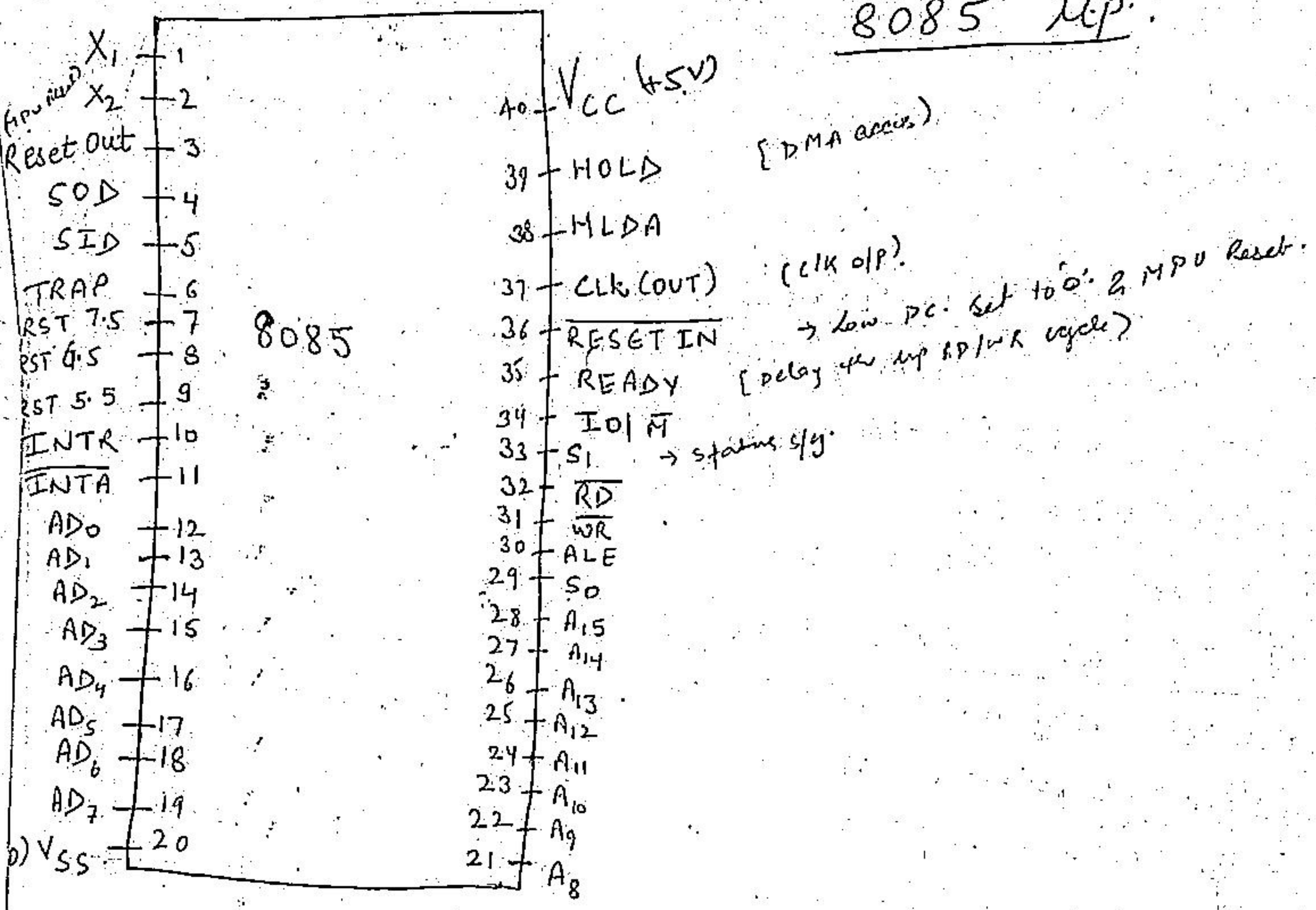
HLDA (O/P) → This signal acknowledges the HOLD request.

READY (I/P) - Signal is used to delay the µp read or write cycles until a slow-responding peripheral is ready to send or accept data.

$\overline{\text{RESET IN}}$ → active low signal. Program Counter is set to zero, buses are tri-stated & MPU is reset.

RESET OUT → MPU is being reset. This signal can be used to reset other devices.

PIN DIAGRAM OF 8085 μp.



All the signals of 8085 can be classified into six groups:-

) Address Bus

) Data Bus

) Control & Status signals.

4.) Power supply & freq. signals.

5.) Externally initiated signals.

6.) Serial I/O ports.

Address Bus :-

8085 has 16 signal lines used as the address bus.

These lines are split into two segments: $A_{15}-A_8$ & AD_7-AD_0.

These lines are uni-directional & are used for the most significant bits (MSB) $(A_{15}-A_8)$ called high-order address of 16-bit address

→ Signal lines AD_7-AD_0 are used for dual purpose.

2) Data Bus/ Multiplexed Address :-

→ Signal lines $AD_7 - AD_0$ are bidirectional.

→ They are used as the low order address bus as well as the data bus.

3) Control and Status Signals :-

This group consists of two Control signals ($\overline{RD}$ & $\overline{WR}$), th status signals ($IO/\overline{M}, S_1, S_0$) to identify nature of the operation, & one special signal (ALE) to indicate the beginning of the operation.

ALE (Address Latch Enable) :- It indicates that the b on $AD_7 - AD_0$ are address bits. This signal is used to latch the low-order address from the multiplexed bu & generate a separate set of eight address lines, A_7-

$\overline{RD}$ (Read) :- This signal indicates that selected I/o or memory device is to be read & data are available on data bus. (active low signal)

$\overline{WR}$ (Write) :- This signal indicates that data on data bus are to be written into a selected memory or I/o location.

$IO/\overline{M}$:- Status signal used to differentiate b/w I/o & memory operations. If $IO/\overline{M}$ = 1 then I
o
= 0 then M
o

S_1 & S_0 :- These status Signals combinely identify various operations.

M/C cycle	$IO/\overline{M}$	S_1	S_0	Control Signals
opcode fetch	0	1	1	$\overline{RD}$ = 0
M/ry read	0	1	0	$\overline{RD}$ = 0
M/ry 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
Int. Acknow.	1	1	1	$\overline{INTA}$ = 0

Opcode and Oprand:- Each instruction contains two parts: operation code (Opcode) and Oprand. The first part of an instruction which specifies the task to be performed by the µP is called the opcode. The second part of the instruction is the data to be operated on, called the oprand. The oprand given in the instruction may be in various form such as 8 bit or 16 bit data, 8 bit or 16 bit add^s internal reg. or a reg. or memory location. In some instructions the oprand is implicit, For Ex.: DAA (Decimal Adjust the Acc^r)

Mov A, B 78H is the opcode of this ins^N & A, B are the oprands.

and Mov A, B is called mnemonic (that term used for the understanding of the user.

Instruction Word size:- A digital computer or a µP understand instructions written in Binary codes (machine codes). The machine codes of all the instructions are not of the same length. According to the word size the 8085 instructions are classified into three types:

(1) 1- byte instruction
(2) 2- byte "
(3) 3- byte "

One Byte instruction: In one byte instruction opcode and operands occurs in the same byte. For Ex. → MOV A, B (Move the content of reg. B to reg. A. only opcode is used, no other 8 bit or 16 bit data used)

Two byte instruction:- In a two-byte instruction the Ist Byte of the instruction is its opcode and 2nd byte is either data or address.

Ex. MVI B, 05H ; Move 05H to Reg. B

06H, 05H ; MVI B, 05 in the codeform

Three Byte instruction:- In a 3-byte instruction The first byte of the ins^n is its Opcode and the 2nd and 3rd bytes. are either 16 bit data or 16 bit address.

Ex,

LXI H, 2400H ; Load H-L pair with 2400H

21H, 00H, 24H ; LXI H, 2400H in the codeform

T-state!- It is the part of the operation of performed in one clock cycle. One T-state is precisely equal to 1 clock period.

Machine Cycle! It is the cycle in which processor can't be interrupted in between. Once the processor enters into any machine cycle that it can't be intesrupted to do any other task.

Eg. Types of Machine cycles:-

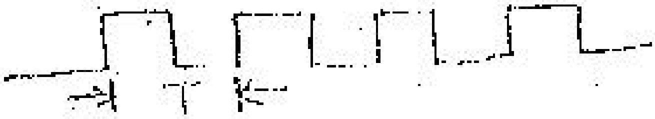
1- Opcode Fetch cycle:- It is the operation of fetching of opcode from the memory. opcode fetch cycle may consists of 4-6 T-states. very first cycle of a program of each insn is fetch cycle.

2. Memory Read Cycle!- The memory read cycle is an operation of reading the data from memory. It requires 3-T states.

3. Memory write cycle! It is the operation of writing the data on the memory. It requires 3-T states.

4- I/O Read cycle!- It is the reading of data from I/O device. It requires 3-T states.

5. I/O Write cycle!- It is the operation of writing the data on the I/O device. It requires 3-T states.



Instruction Cycle:- An instruction is a command
given to the computer to perform a specific
operation on a given data. CPU fetches one
instruction at a time and executes it.

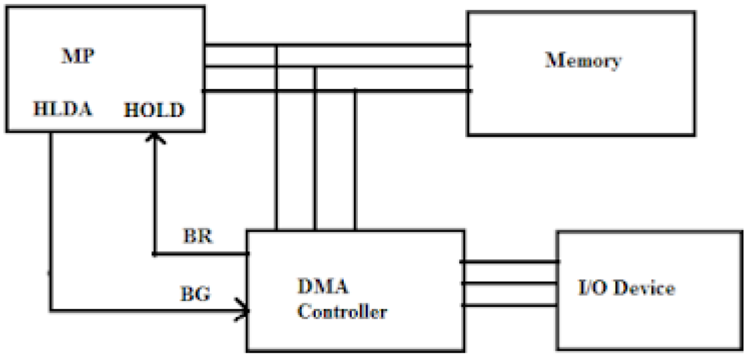
The necessary steps that a CPU(µP) carries
out to fetch an instruction and necessary d
from the memory and to execute it, con
tute an instruction cycle. An instruction cyc
consists of fetch cycle and execute cycle
In fetch cycle a CPU or µP fetches opcode
from the memory. The necessary steps to
Carried out to fetch an opcode from the
memory constitute a fetch cycle. The necess
steps which are carried out to get data
if any from the memory and to perform
specific operation specified in an instruction
constitute an execute cycle.

The time ~~slot~~ required to fe
an opcode (FC) is a fixed slot of time
the time required to execute an instruc
(EC) is variable which depends on the t
of instruction to be executed. The total
required to execute an instruction is gi
by $\boxed{IC = FC+EC}$ *

~~Tstto~~ . The life of instruction from the
it is brought into processor reg. till it is
completely executed is known as instructi
cycle.

Concept of Direct Memory Access (DMA)

- DMA is a process of communication for data transfer between memory and input/output, controlled by an external circuit called DMA controller, without involvement of CPU.
- 8085 MP has two pins HOLD and HLDA which are used for DMA operation.
- First, DMA controller sends a request by making Bus Request (BR) control line high. When MP receives high signal to HOLD pin, it first completes the execution of current machine cycle, it takes few clocks and sends HLDA signal to the DMA controller.
- After receiving HLDA through Bus Grant (BG) pin of DMA controller, the DMA controller takes control over system bus and transfers data directly between memory and I/O without involvement of CPU. During DMA operation, the processor is free to perform next job which does not need system bus.
- At the end of data transfer, the DMA controller terminates the request by sending low signal to HOLD pin and MP regains control of system bus by making HLDA low.



Direct Memory Access (DMA)

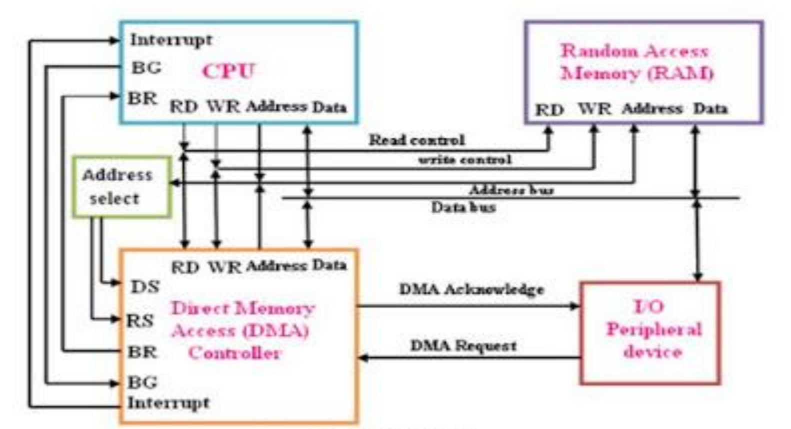


Fig: DMA Transfer

Use this link for instruction set

https://www.slideshare.net/gokulvlsi/8085-instruction-set

Addressing modes in 8085 microprocessor

The way of specifying data to be operated by an instruction is called addressing mode.
Types of addressing modes –
In 8085 microprocessor there are 5 types of addressing modes:

1. **Immediate Addressing Mode –**
   In immediate addressing mode the source operand is always data. If the data is 8-bit, then the instruction will be of 2 bytes, if the data is of 16-bit then the instruction will be of 3 bytes.
   Examples:
   MVI B 45 (move the data 45H immediately to register B)
   LXI H 3050 (load the H-L pair with the operand 3050H immediately)
   JMP address (jump to the operand address immediately)

2. **Register Addressing Mode –**
   In register addressing mode, the data to be operated is available inside the register(s) and register(s) is(are) operands. Therefore the operation is performed within various registers of the microprocessor.
   Examples:
   MOV A, B (move the contents of register B to register A)
   ADD B (add contents of registers A and B and store the result in register A)
   INR A (increment the contents of register A by one)

3. **Direct Addressing Mode –**
   In direct addressing mode, the data to be operated is available inside a memory location and that memory location is directly specified as an operand. The operand is directly available in the instruction itself.
   Examples:
   LDA 2050 (load the contents of memory location into accumulator A)
   LHLD address (load contents of 16-bit memory location into H-L register pair)
   IN 35 (read the data from port whose address is 01)

4. **Register Indirect Addressing Mode –**
   IN register indirect addressing mode, the data to be operated is available inside a memory location and that memory location is indirectly specified b a register pair.
   Examples:
   MOV A, M (move the contents of the memory location pointed by the H-L pair to the accumulator)
   LDAX B (move contains of B-C register to the accumulator)
   LXIH 9570 (load immediate the H-L pair with the address of the location 9570)

5. **Implied/Implicit Addressing Mode –**
   In implied/implicit addressing mode the operand is hidden and the data to be operated is available in the instruction itself.
   Examples:
   CMA (finds and stores the 1's complement of the contains of accumultor A in A)
   RRC (rotate accumulator A right by one bit)
   RLC (rotate accumulator A left by one bit)

the

Interrupts:- An interrupt is interruption in normal execution of a program to perform some specific task. The 8085 interrupt process is controlled by the Interrupt Enable F/F, which is internal to the processor and can be set or reset by using software instructions. There are 5 different interrupt in 8085 called INTR, RST5.5, RST6.5, RST7.5, TRAP.

Classification of Interrupts:-

1- Vectored and Non Vectored Interrupt:- These instruction interrupts are automatically vectored (transferred) to specific locations on memory without any external hardware:

RST7.5, RST6.5, RST5.5 and TRAP are the four vectored interrupt of 8085. When signal on the pion pin of either of these interrupts is made high a subroutine corresponding to that interrupt is called immediately. The addresses assigned to the vectored interrupts are as follows:

Non Vectored Interrupt:- INTR is the only non vectored interrupt of 8085. When INTR interrupt is given to the microprocessor, it completes the execution of current instruction being executed and then generate the interupt acknowledgement signal called INTA.

When interrupt acknowledgement signal is generated by microprocessor 8 different subroutine can be called by giving restart instructions through external hardware. The res instructions and their corresponding memory addres are given as follows:

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

Software and Hardware Interrupts:- Software When any of the restart instruction is included in the main program as software instruction, the instruction calls the subroutine stored at addresses corresp nding to these instruction. This one byte call of subroutine by restart instruction is called a software instruction.

Hardware Interrupts:- When an interrupt servic routine is called due to interrupt request from external hardware interrupt. INTR, RST RST 1.5 and TRAP are examples of hardware in

Maskable and Non Maskable Interrupt! - ~~INTR~~

Maskable Interrupt!- INTR, RST5.5, RST6.5, RST7.5 are maskable interrupts of 8085. The maskable interrupt can be enabled using the instructio EI in the program. These interrupts can be disabled ~~either by calling the subroutine corresponding to them or~~ by giving software instructions like DI and SIM. The instruction DI disables all interrupts (maskable) when it is executed. SIM (Set Interrupt Mask) is used for masking of RST 5.5, RST 6.5, RST 7.5 individually. When maskable interrupts are given to the µP it completes the execution of current instruction being executed and then call the subroutine of interrupt if it is vectored interrupt and generates interrupt acknowledgement signal if it is non vectored interrupt.

Non-Maskable Interrupts!- TRAP is the only non-maskable interrupt of 8085 µP. It cannot be disabled by any instruction or subroutine. When signal on TRAP pin is made high, the µP stops execution of the instruction (current) being executed and immediately call the subroutine stored at 0024H. without any external hardware or the interrupt enable instruction EI. TRAP is used in the most emergent situation like power failure and emergency shutoff.

Priority Level!-

TRAP ← RST7.5 - RST6.5 - RST5.5 - INTR

Highest Priority ——— Decreasing order ———> Lowest Priority.

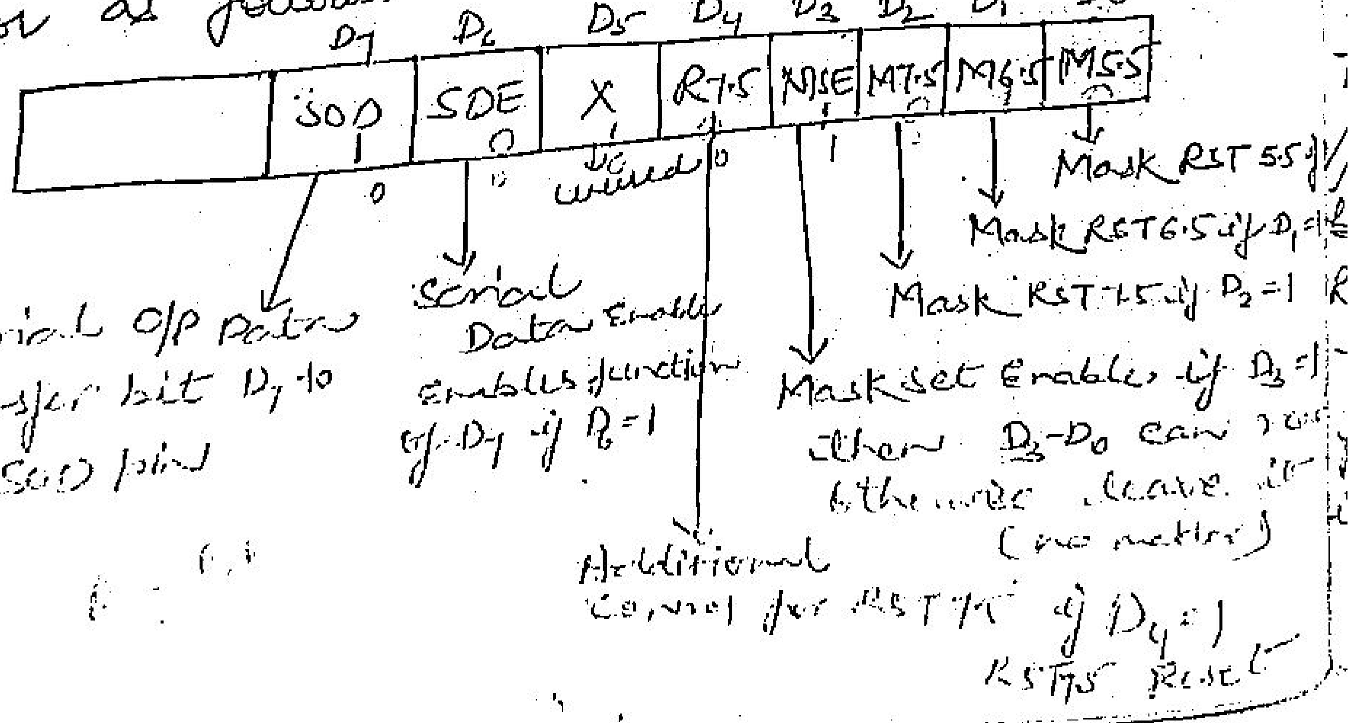
Instruction Related to the Interrupt:-

EI, DI, SIM & RIM

EI (Enable Interrupt) → The EI instruction must be given in the main program to enable the interrupt before acknowledgly any interrupt. When a The interrupt is disabled automatically when a particular interrupt is acknowledged. When a particular interrupt is ak acknowledged all the maskable interrupt is are disabled automati maskable interrupts are disabled. EI instruct For enabling the interrupts, again EI instruct must be used in the service routine of the interrupt.

DI - ~~I~~ The DI instruction is used to disable all the maskable interrupts. It is generally used in the portion of the program where interrupt is required to be disabled. ~~In~~ The interrupts are can be enabled again by using EI instruction

SIM → (Set Interrupt Mask) → To mask the maskable vectored interrupt RST 5.5, RST 6.5, RST 7.5, SIM instruction is used. SIM instruction performs various functions on the basis of contents of the accumulator at the time of execution of the instructions. It interprets the content of Accumulator as follows!



D7
D6
D5
D4
D3
D2
D1
D0
SOD
SDE
X
R7.5
MSE
M7.5
M6.5
M5.5
1
0
0
0
1
0
0
1
0
Dc
Mask RST 5.5 if 1
Mask RST 6.5 if D1 = 1
Mask RST 7.5 if D2 = 1
Serial O/P Data Transfer bit D7 to SOD pin
Serial Data Enable enables function of D7 if D6 = 1
Mask set Enable if D3 = 1 then D3-D0 can ... otherwise leave it (no matter)
Additional control for RST 7.5 if D4 = 1 RST7.5 Reset

RIM → (Read Interrupt Mask):- RIM instruction is used to read the status of pending interrupt and to read the data serially from SID pin to D7 bit of accumulator.

It interprets the content of accumulator as under:

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
SID	I7.5	I6.5	I5.5	IE	R7.5	R6.5	R5.5
0	1	0	0	1	0	0	1

Pending interrupts

Read data from SID pin to D_7 bit of A

$D_0 = 1$ if RST 5.5 is masked

$D_1 = 1$ if RST 6.5 masked

$D_2 = 1$ if RST 7.5 masked

Interrupt Enable f/f is set if $D_3 = 1$

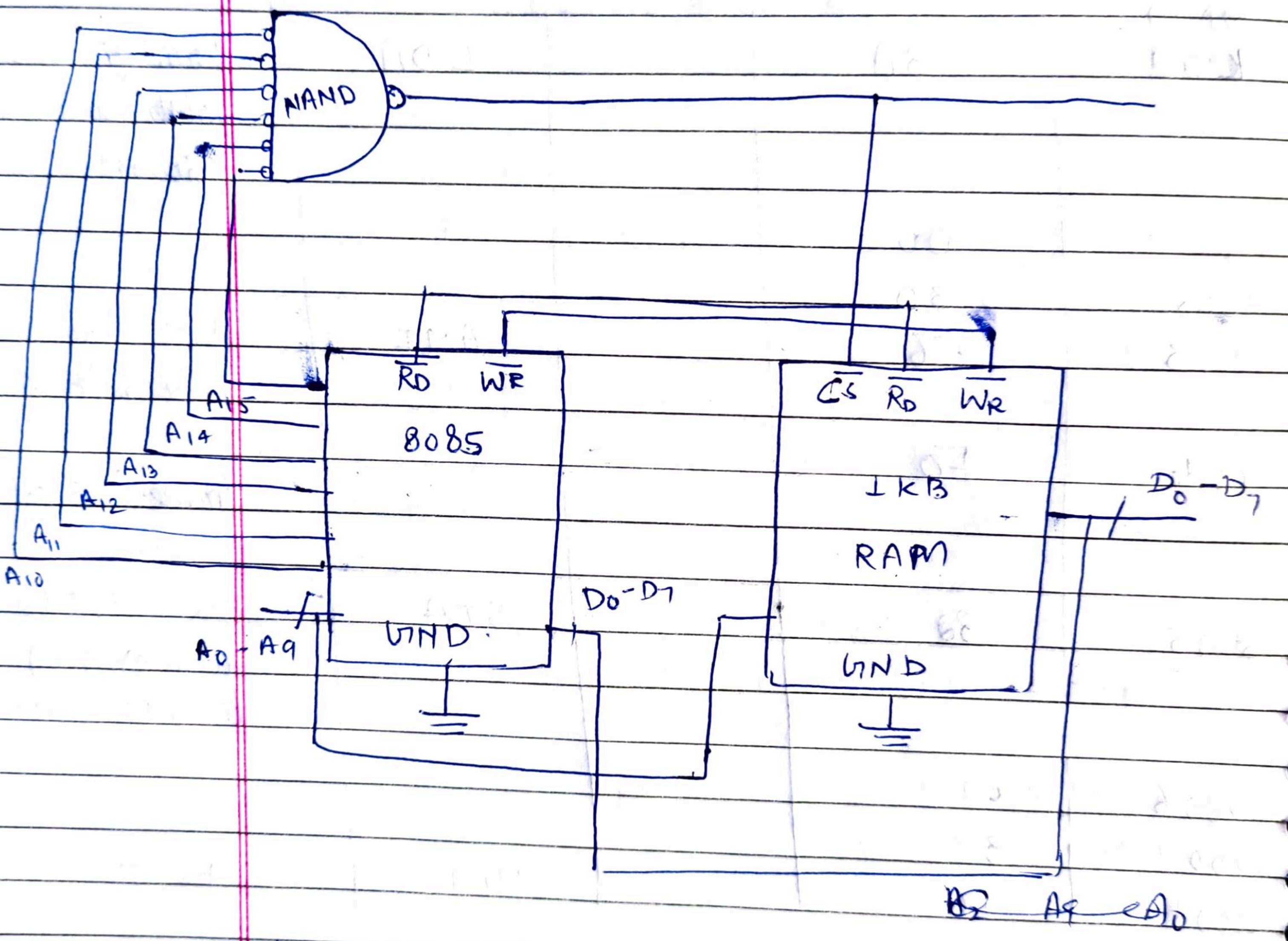
If $D_4 = 1$; RST 5.5 is pending

If $D_5 = 1$ RST 6.5 pending

If $D_6 = 1$ RST 7.5 pending

Q connect 1KB RAM with system lines of 8085 (How chayie)

address line = 10



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	0	0	1	1	1	1	1	1	1	1	1	1	→ 03FFH

Q Connect 2KB E-P ROM with system lines of 8085 with initial address 6000H

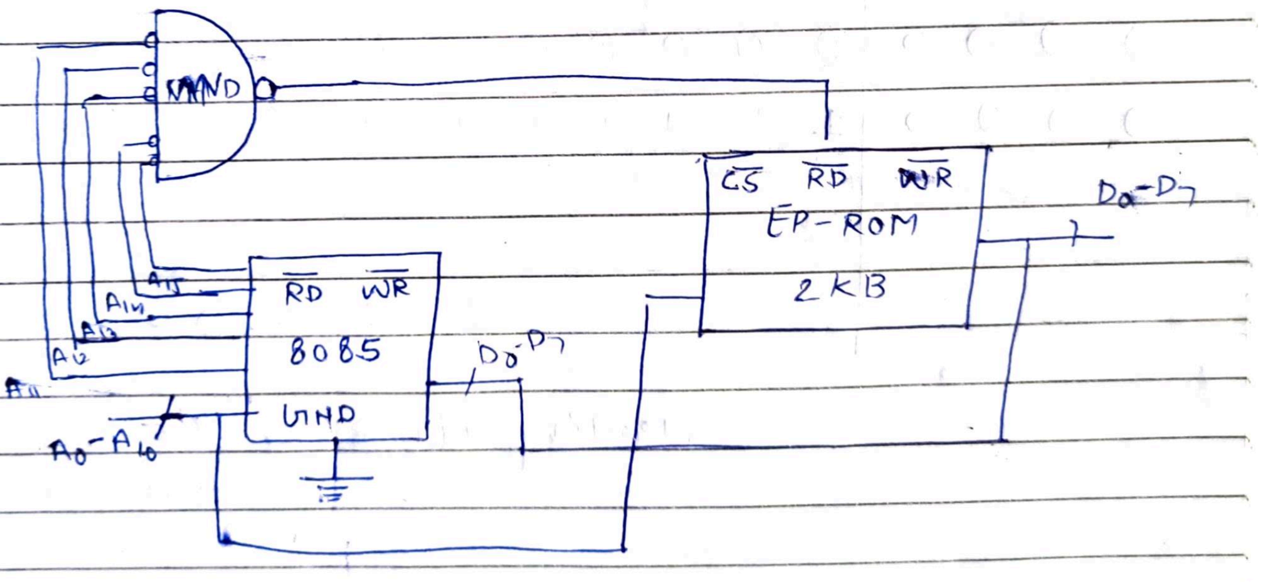
Ans ⇒ 2KB

address lines 11

data lines — 8

final address

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	
0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	6000
0	1	1	0	0	1	1	1	1	1	1	1	1	1	1	1	67FF



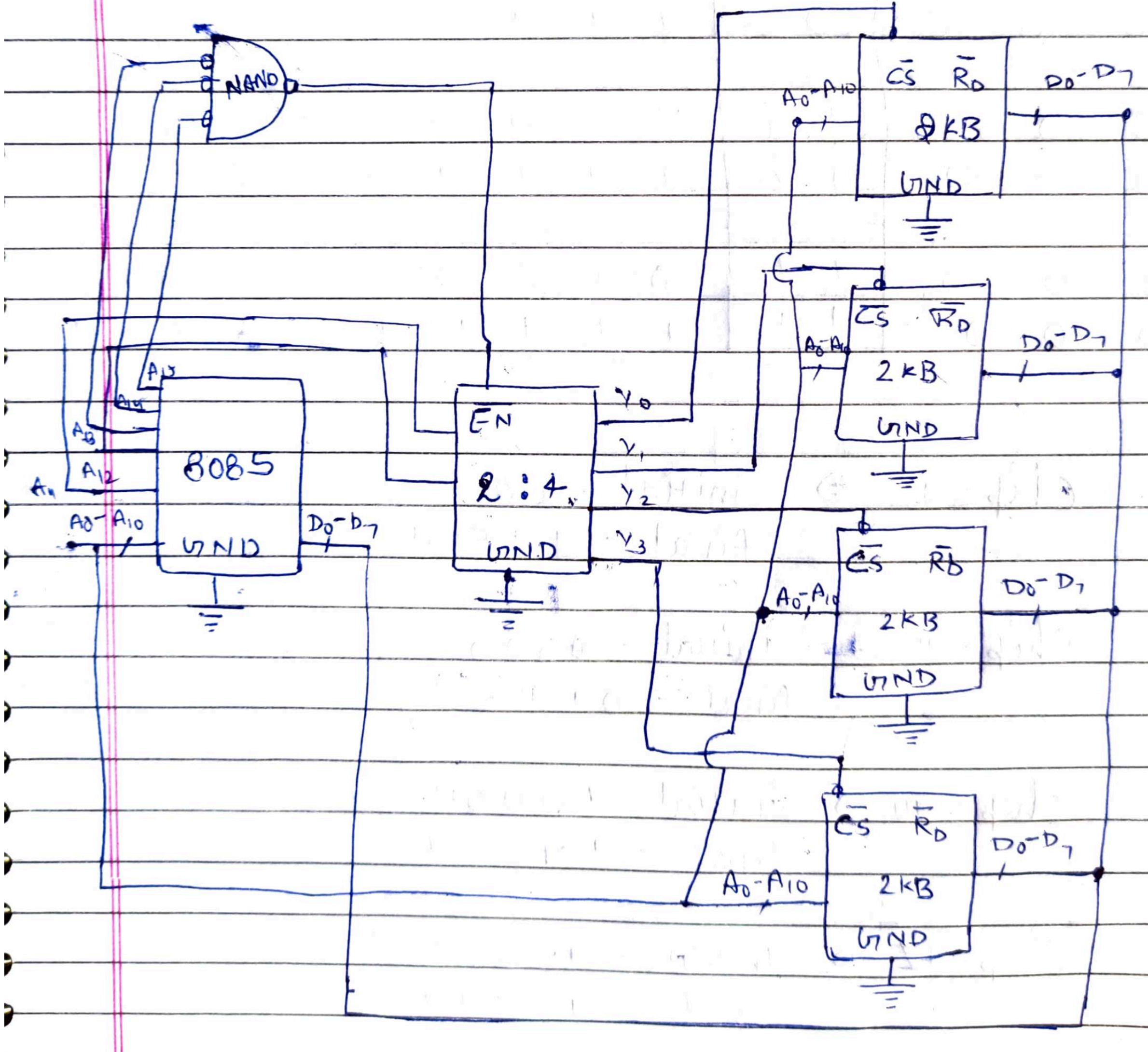
Q Connect 8 KB E-PROM to 8085 microprocessor but 2K × 8 chip is available use 2 : 4 decoder to generate chip select logic.

Ans ⇒ 8 KB 2K×8 8 KB

4 = 13

data = 8

address = 11



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

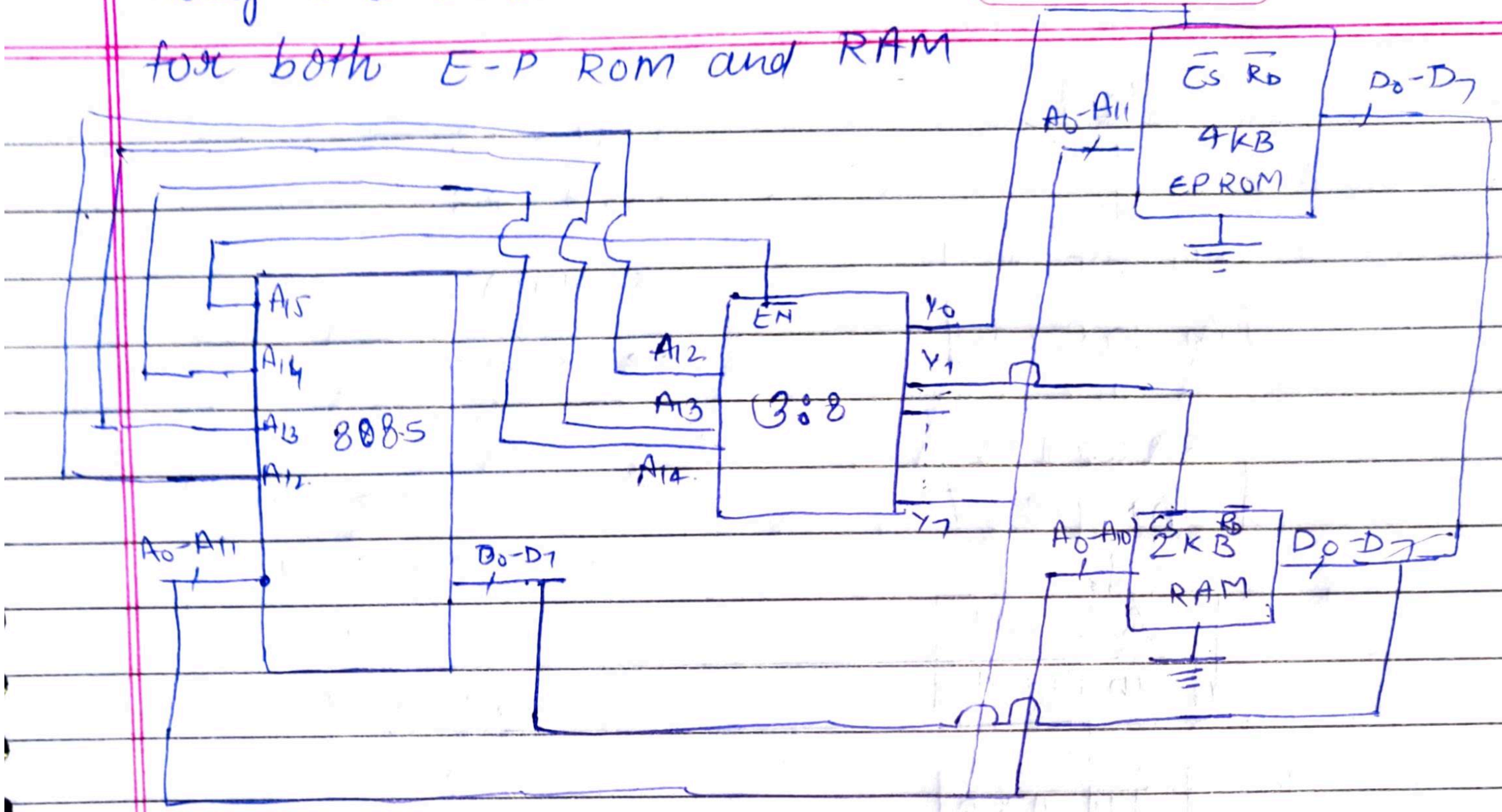
Chip - I ⇒ initial - 0000H
final - 1 FFFH.

Chip - II ⇒ initial - 0800H
final - 0FFFH

Chip - III ⇒ initial - 1000H
final - 17FFH

Chip - IV ⇒ initial - 1800H
final - 1FFFH

S Interface 8085 with 4KB E-PROM and 8KB RAM using 3:8 Decoder also write the range of address for both E-P ROM and RAM



X = 0 (say)

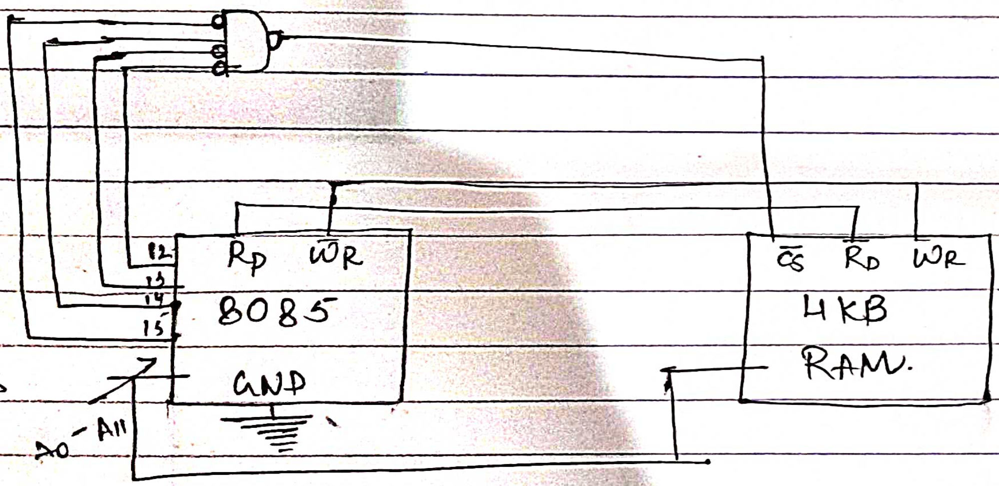
	A15	A14	A13	A12	A11	A10	A9	A8	A7	A6	A5	A4	A3	A2	A1
E-PROM	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Y0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1
RAM	0	0	0	X	X	0	0	0	0	0	0	0	0	0	0
Y1	0	0	0	1	X	1	1	1	1	1	1	1	1	1	1

~~Ans:-~~ Connect 4KB S-RAM with 8085. Assume final Add. $(4FFF)_H$.

Data line - 8.
Add line - 12

A15	A14	A13	A12	A11	A10	A9	A8	A7	A6	A5	A4	A3	A2	A1	A0	
0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	Initial Address
0	1	0	0	1	1	1	1	1	1	1	1	1	ɸ	ɸ	ɸ	final Add.

Initial Address = $(4000)_H$.



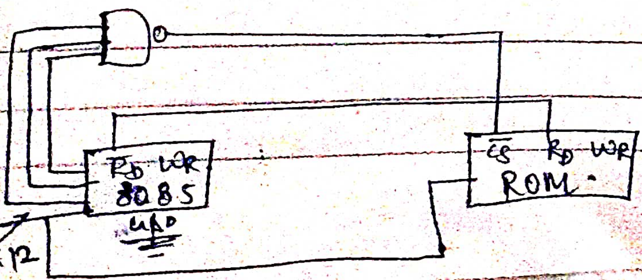
passed value is 1.

A0 - A11

~~Ans:-~~ Connect 8KB E-PROM with 8085 Micro-processor if the starting address is $(E000)_H$. → Address lines = 13.
Data lines = 8.

A15	A14	A13	A12	A11	A10	A9	A8	A7	A6	A5	A4	A3	A2	A1
1	1	1	0	0	0	0	0	0	0	0	0	0	0	0
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

final Address $(FFFF)_H$.



passed value is 1.

A0-A12

Timing Diagram

Timing Diagram is a graphical representation. It represents the execution time taken by each instruction in a graphical format. The execution time is represented in T-states.

Instruction Cycle:

The time required to execute an instruction is called instruction cycle.

Machine Cycle:

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

T-State:

✓The machine cycle and instruction cycle takes multiple clock periods.

✓A portion of an operation carried out in one system clock period is called as T-state.

1 Machine cycles of 8085

The 8085 microprocessor has 5 (seven) basic machine cycles. They are

✓Opcode fetch cycle (4T)

✓Memory read cycle (3 T)

✓Memory write cycle (3 T)

✓I/O read cycle (3 T)

✓I/O write cycle (3 T)

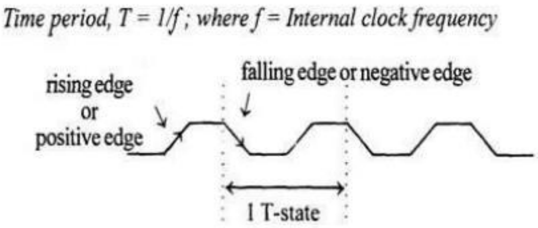
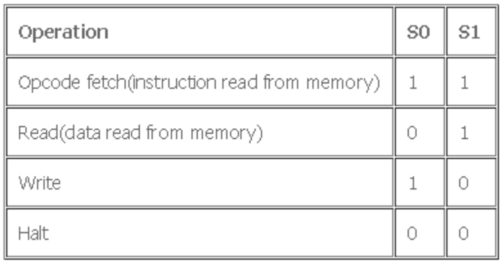

Time period, T = 1/f ; where f = Internal clock frequency
rising edge or positive edge
falling edge or negative edge
1 T-state

Fig 1.7 Clock Signal

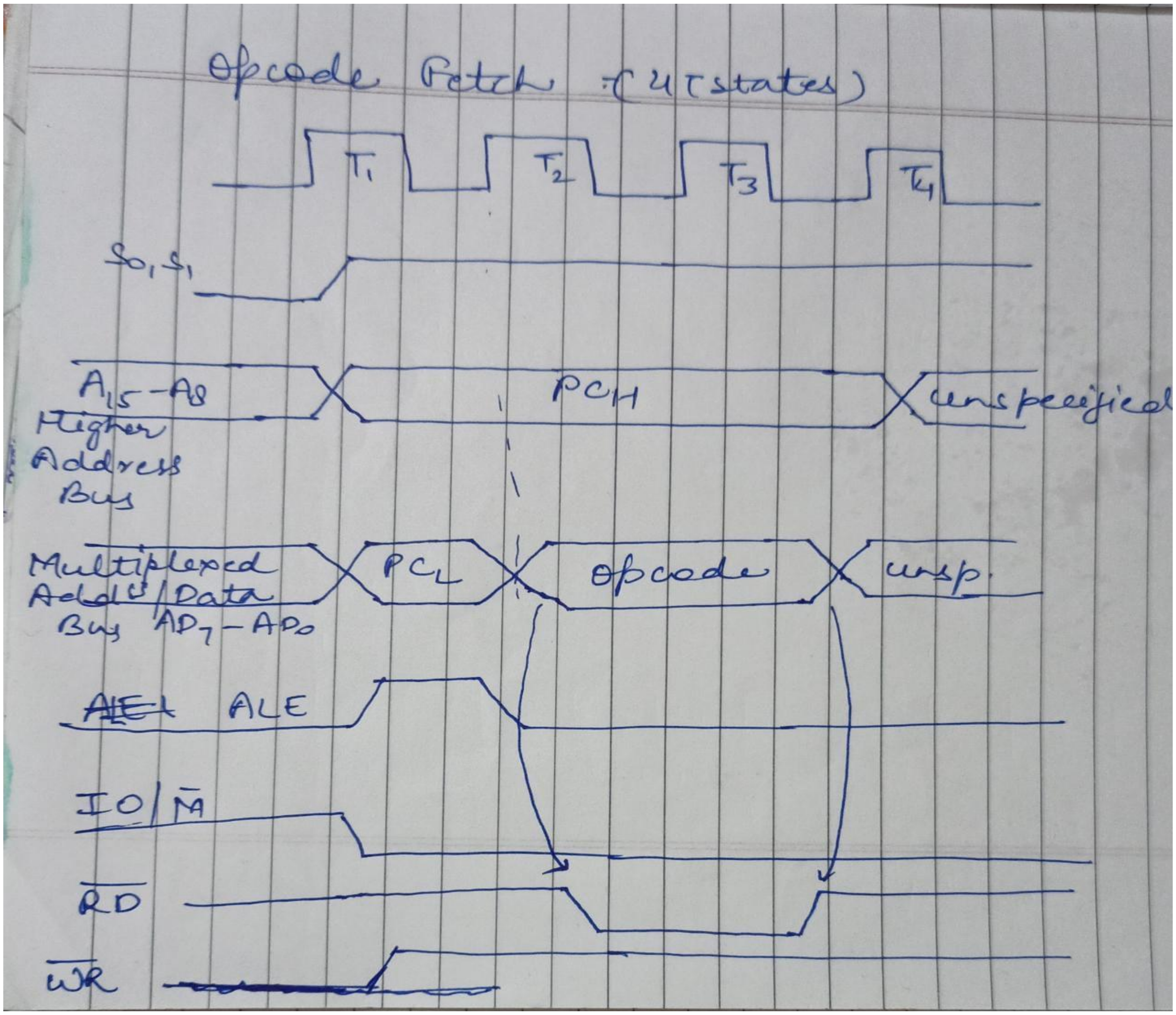


Operation	S0	S1
Opcode fetch(instruction read from memory)	1	1
Read(data read from memory)	0	1
Write	1	0
Halt	0	0

Signal 1.Opcode fetch machine cycle of 8085 :

Example: MOV B,C

ADDRESS	OPCODE	MNEMONIC
2000	41H	MOV B,C



✓Each instruction of the processor has one byte opcode.

✓The opcodes are stored in memory. So, the processor executes the opcode fetch machine cycle to fetch the opcode from memory.

✓Hence, every instruction starts with opcode fetch machine cycle.

✓The time taken by the processor to execute the opcode fetch cycle is 4T.

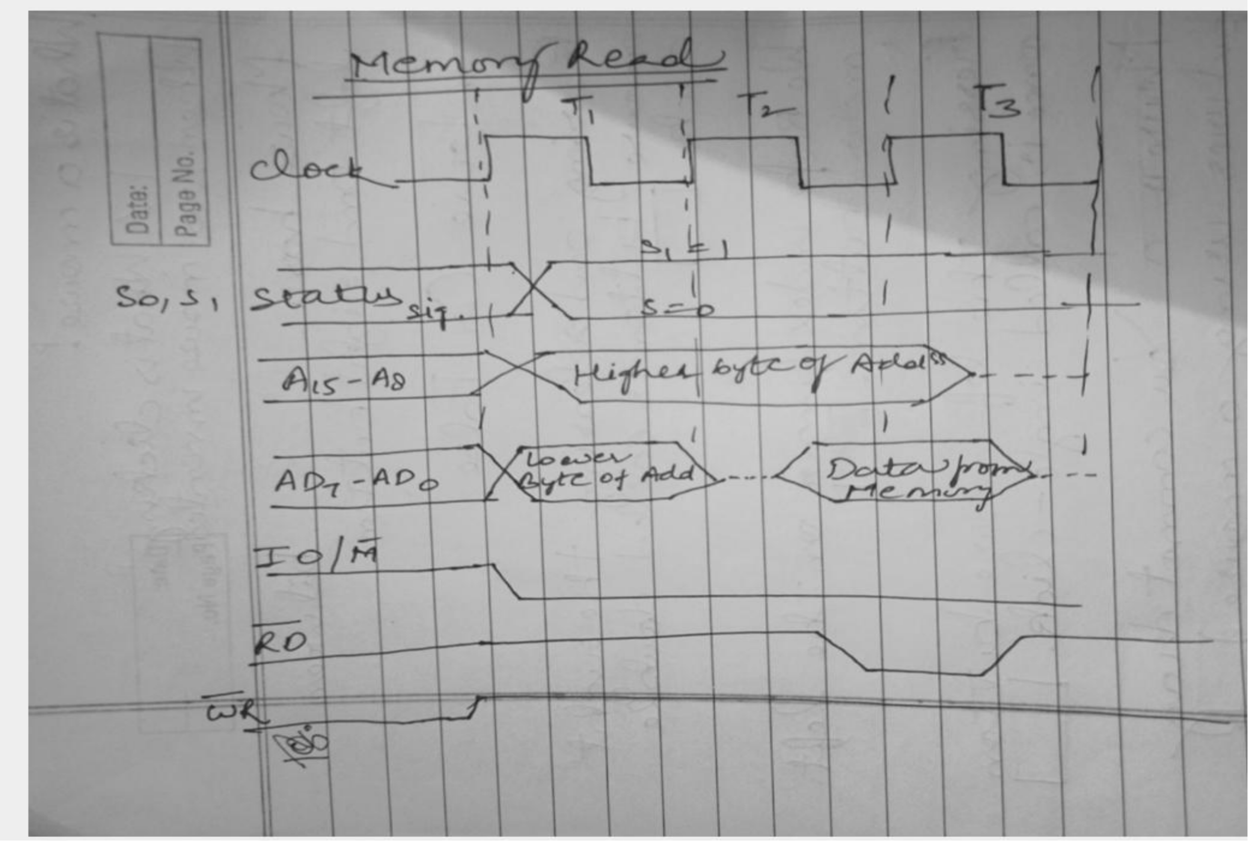
✓In this time, the first, 3 T-states are used for fetching the opcode from memory and the remaining T-states are used for internal operations by the processor.

·2. Memory Read Machine Cycle of 8085:

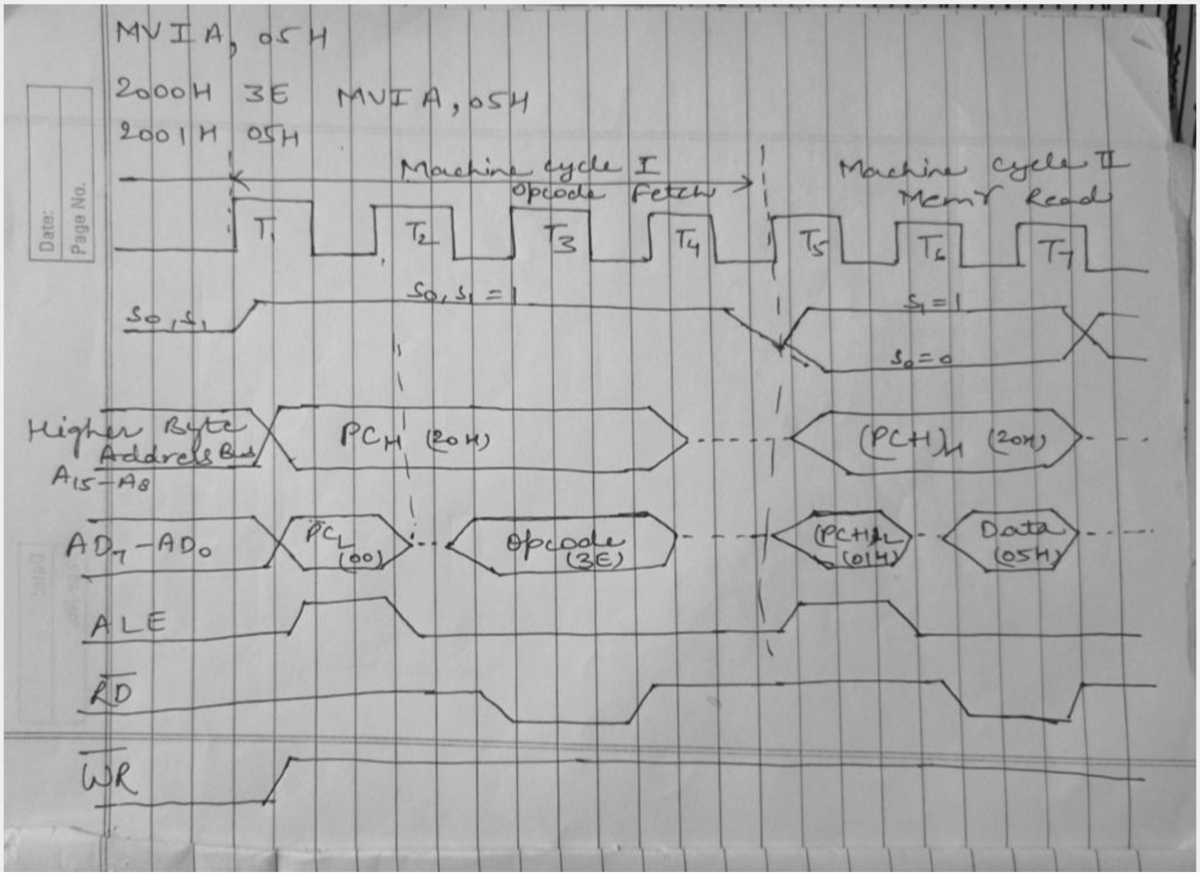
✓The memory read machine cycle is executed by the processor to read a data byte from memory.

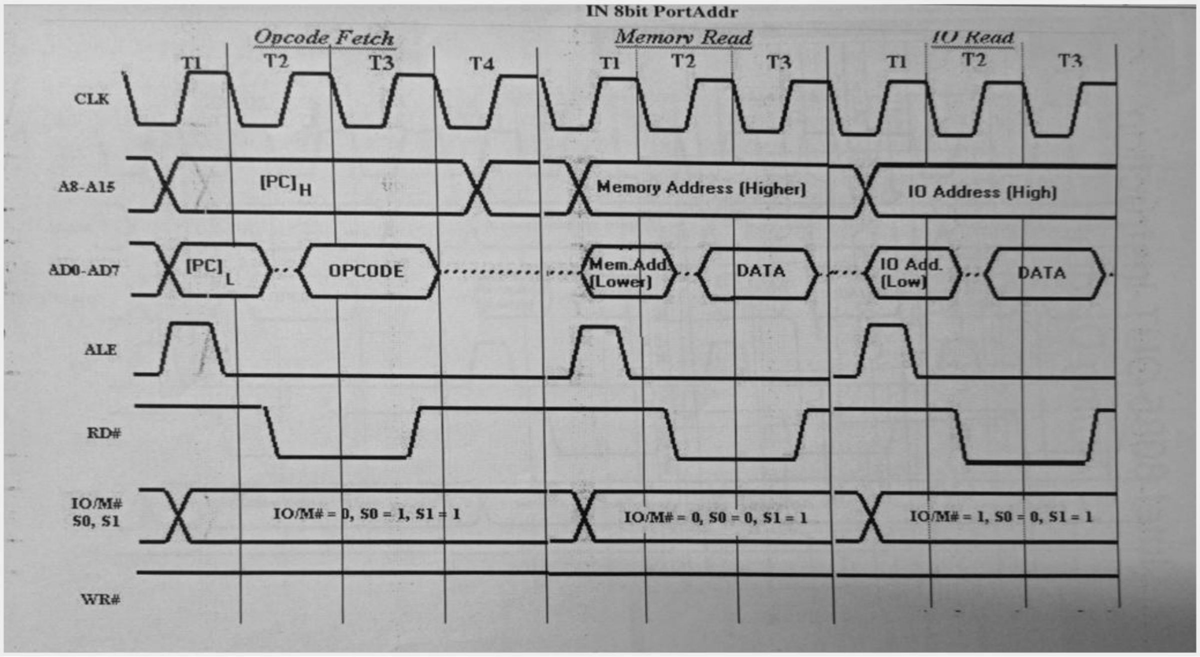
✓The processor takes 3T states to execute this cycle.

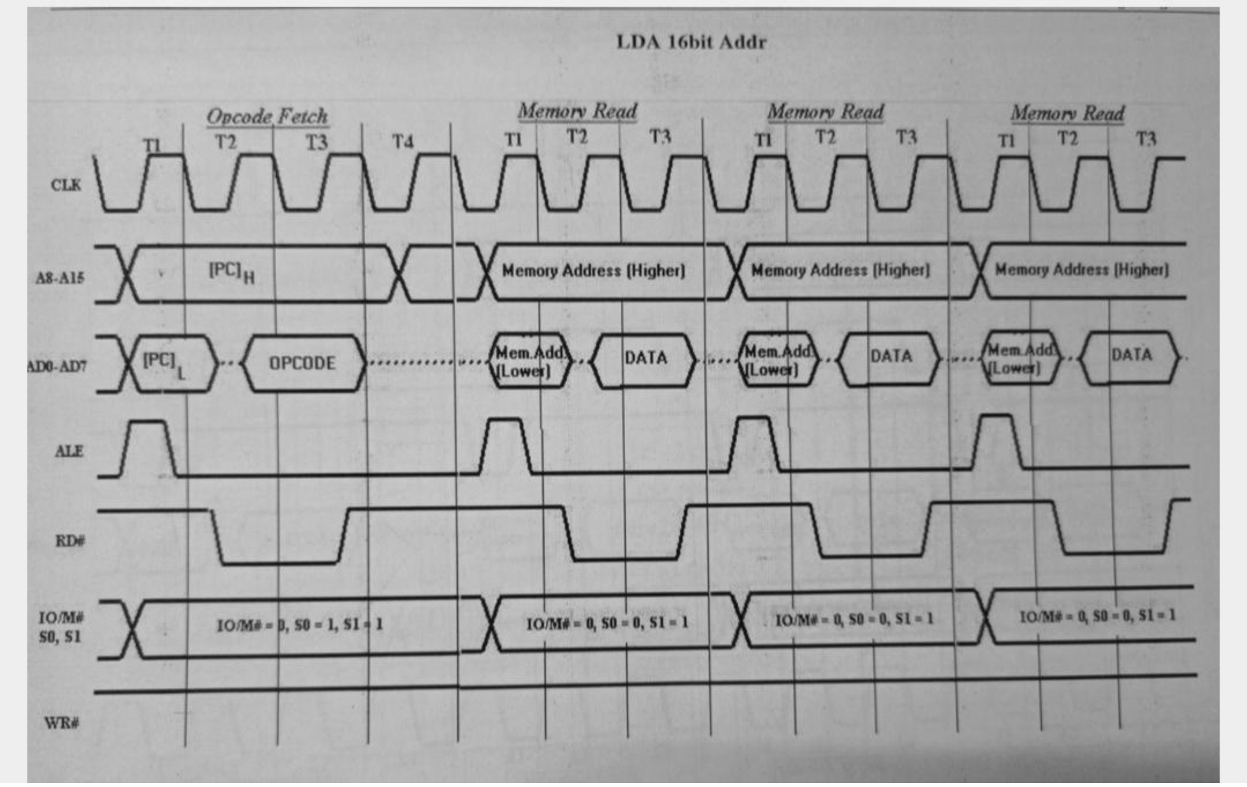
The instructions which have more than one byte word size will use the machine cycle after the opcode fetch machine cycle.



·


MVI A, 05H
2000H 3E MVI A, 05H
2001H 05H
Machine cycle I
Opcode Fetch
Machine cycle II
Mem^r Read
T1
T2
T3
T4
T5
T6
T7
S0, S1
S0, S1 = 1
S1 = 1
S0 = 0
Higher Byte Address Bus
A15 - A8
PCH (20H)
(PC+1)H (20H)
AD7 - AD0
PCL (00)
Opcode (3E)
(PC+1)L (01H)
Data (05H)
ALE
RD
WR


IN 8bit PortAddr
Opcode Fetch
Memory Read
IO Read
T1
T2
T3
T4
T1
T2
T3
T1
T2
T3
CLK
A8-A15
[PC]H
Memory Address (Higher)
IO Address (High)
AD0-AD7
[PC]L
OPCODE
Mem.Add. (Lower)
DATA
IO Add. (Low)
DATA
ALE
RD#
IO/M#
S0, S1
IO/M# = 0, S0 = 1, S1 = 1
IO/M# = 0, S0 = 0, S1 = 1
IO/M# = 1, S0 = 0, S1 = 1
WR#


LDA 16bit Addr
Opcode Fetch
Memory Read
Memory Read
Memory Read
T1
T2
T3
T4
T1
T2
T3
T1
T2
T3
T1
T2
T3
CLK
A8-A15
[PC]H
Memory Address (Higher)
Memory Address (Higher)
Memory Address (Higher)
AD0-AD7
[PC]L
OPCODE
Mem.Add (Lower)
DATA
Mem.Add (Lower)
DATA
Mem.Add (Lower)
DATA
ALE
RD#
IO/M# S0, S1
IO/M# = 0, S0 = 1, S1 = 1
IO/M# = 0, S0 = 0, S1 = 1
IO/M# = 0, S0 = 0, S1 = 1
IO/M# = 0, S0 = 0, S1 = 1
WR#

8086 Microprocessor

Definition: 8086 is a 16-bit microprocessor and was designed in 1978 by Intel. Unlike, 8085, an 8086 microprocessor has **20-bit address bus**. Thus, is able to access 2^{20} i.e., 1 MB address in the memory.

As we know that a microprocessor performs arithmetic and logic operations. And an 8086 microprocessor is able to perform these operations with 16-bit data in one cycle. Hence is a **16-bit microprocessor**.

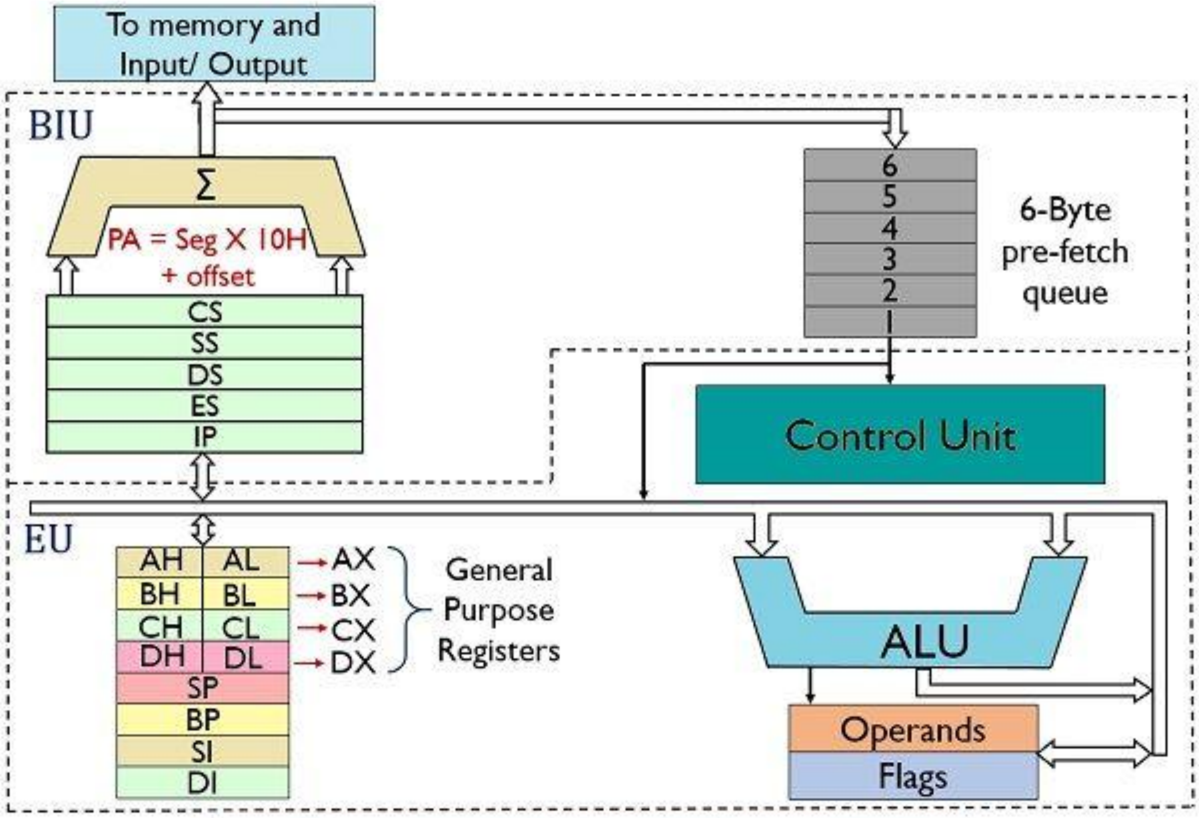
Thus the size of the data bus is 16-bit as it can carry 16-bit data at a time. The architecture of 8086 microprocessor, is very much different from that of 8085 microprocessor.

We have already discussed the introduction to the microprocessor and 8085 microprocessor. So, lets now proceed further and understand the architecture and working of 8086 microprocessor.

Block Diagram of 8086 Microprocessor

The architecture of 8086 microprocessor is composed of 2 major units, the BIU i.e., Bus Interface Unit and EU i.e., Execution Unit.

The figure below shows the block diagram of the architectural representation of the 8086 microprocessor:



Block Diagram of 8086 Microprocessor

Electronics Desk

Bus Interface Unit (BIU)

The Bus Interface Unit (BIU) manages the data, address and control buses.

The BIU functions in such a way that it:

- Fetches the sequenced instruction from the memory,
- Finds the physical address of that location in the memory where the instruction is stored and
- Manages the 6-byte pre-fetch queue where the pipelined instructions are stored.

An 8086 microprocessor exhibits a property of pipelining the instructions in a queue while performing decoding and execution of the previous instruction.

This saves the processor time of operation by a large amount. This pipelining is done in a **6-byte queue**.
Also, the BIU contains **4 segment registers**. Each segment register is of 16-bit. The segments are present in the memory and these registers hold the address of all the segments.

6-byte pre-fetch queue: This queue is used in 8086 in order to perform pipelining. As at the time of decoding and execution of the instruction in EU, the BIU fetches the sequential upcoming instructions and stores it in this queue.
The size of this queue is 6-byte. This means at maximum a 6-byte instruction can be stored in this queue. The queue exhibits **FIFO** behaviour., first in first out.

- Fetching the next instruction while the current instruction executes is called **pipelining**.
- **Segment register** − BIU has 4 segment buses, i.e. CS, DS, SS& ES. It holds the addresses of instructions and data in memory, which are used by the processor to access memory locations. It also contains 1 pointer register IP, which holds the address of the next instruction to executed by the EU.
    - **CS** − It stands for Code Segment. It is used for addressing a memory location in the code segment of the memory, where the executable program is stored.
    - **DS** − It stands for Data Segment. It consists of data used by the program and is accessed in the data segment by an offset address or the content of other register that holds the offset address.
    - **SS** − It stands for Stack Segment. It handles memory to store data and addresses during execution.
    - **ES** − It stands for Extra Segment. ES is additional data segment, which is used by the string to hold the extra destination data.
- **Instruction pointer** − It is a 16-bit register used to hold the address of the next instruction to be executed.

Execution Unit (EU)

The Execution Unit (EU) performs the decoding and execution of the instructions that are being fetched from the desired memory location.

Control Unit:
Like the timing and control unit in 8085 microprocessor, the control unit in 8086 microprocessor produces control signal after decoding the opcode to inform the general purpose register to release the value stored in it. And it also signals the ALU to perform the desired operation.

ALU:
The arithmetic and logic unit carries out the logical tasks according to the signal generated by the CU. The result of the operation is stored in the desired register.

Flag Register

It is a 16-bit register that behaves like a flip-flop, i.e. it changes its status according to the result stored in the accumulator. It has 9 flags and they are divided into 2 groups − Conditional Flags and control flags.

It represents the result of the last arithmetic or logical instruction executed. Following is the list of conditional flags −

- **Carry flag** − This flag indicates an overflow condition for arithmetic operations.
- **Auxiliary flag** − When an operation is performed at ALU, it results in a carry/barrow from lower nibble (i.e. D0 – D3) to upper nibble (i.e. D4 – D7), then this flag is set, i.e. carry given by D3 bit to D4 is AF flag. The processor uses this flag to perform binary to BCD conversion.
- **Parity flag** − This flag is used to indicate the parity of the result, i.e. when the lower order 8-bits of the result contains even number of 1's, then the Parity Flag is set. For odd number of 1's, the Parity Flag is reset.
- **Zero flag** − This flag is set to 1 when the result of arithmetic or logical operation is zero else it is set to 0.
- **Sign flag** − This flag holds the sign of the result, i.e. when the result of the operation is negative, then the sign flag is set to 1 else set to 0.
- **Overflow flag** − This flag represents the result when the system capacity is exceeded.

Control Flags

Control flags controls the operations of the execution unit. Following is the list of control flags −

- Trap flag − It is used for single step control and allows the user to execute one instruction at a time for debugging. If it is set, then the program can be run in a single step mode.
- Interrupt flag − It is an interrupt enable/disable flag, i.e. used to allow/prohibit the interruption of a program. It is set to 1 for interrupt enabled condition and set to 0 for interrupt disabled condition.
- Direction flag − It is used in string operation. As the name suggests when it is set then string bytes are accessed from the higher memory address to the lower memory address and vice-a-versa.

General purpose register

There are 8 general purpose registers, i.e., AH, AL, BH, BL, CH, CL, DH, and DL. These registers can be used individually to store 8-bit data and can be used in pairs to store 16bit data. The valid register pairs are AH and AL, BH and BL, CH and CL, and DH and DL. It is referred to the AX, BX, CX, and DX respectively.

- **AX register** − It is also known as accumulator register. It is used to store operands for arithmetic operations.
- **BX register** − It is used as a base register. It is used to store the starting base address of the memory area within the data segment.
- **CX register** − It is referred to as counter. It is used in loop instruction to store the loop counter.
- **DX register** − This register is used to hold I/O port address for I/O instruction.

Stack pointer register

It is a 16-bit register, which holds the address from the start of the segment to the memory location, where a word was most recently stored on the stack.

Calculation of Physical Address

The physical address of an instruction is given as:

PA = Segment address X 10 + Offset

For example: Suppose the segment address is 2000 H and the offset address is 4356 H. So, the generated physical address is **24356 H**.

Instruction set of 8086!—

8086 instructions are categorised into the following main types!

(i) Data Transfer Instructions!- These types of instructio are used to transfer data from source oprand to destination oprand.

(i) Mov: Move! Transfers data from one Reg./memory location to another Reg./memory location source and destination may be one of the segment reg. or other general purpose reg. or a memory location.

Example # Mov AX, BX

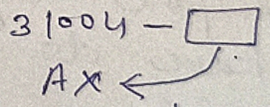
data of BX ⟶ AX

Mov AX, [SI]

[SI] indicates that the contents of SI ~~and~~ which shows memory location

i.e. SI = 1004

[SI] = [1004]



Mov AX, [SI] means move the content of memory location 1004H into AX.

SI = 1004

Mov AX, [2000H]

Mov AL, CH

Mov [0301], AL

We know in AL only 8 bit data lies. so content of AL moves at memory location [0301].

Mov [0301], CX

CX is a 16 bit Regr.

CX | 35 | 42 | H

0301 → ~~357~~ 42H

0302 → 35H

Mov [BX+Disp], CX

Suppose BX → 1005

Disp → 04

The content of CX moves at memory location 1009 H. .

Mov instruction uses for immediate data transfer

In immediate addressing mode, a segment regr can't be a destination Regr. To load the segment regr with immediate data, one will have to load any general purpose Regr with data and then it will have to moved to the particular segment regr.

# e.g. → # Mov AX, 5000H
Mov DS, AX

# Mov AX, 1593H

After Execution AX | 15 | 93H |

# Mov AL, 85H

* Mov [0301], 67H

67H moves at memory location 0301H

* Mov [0301], 1584H

0301 → 84H
0302 → 15H

* Mov [BX], 94H

* Mov [BX], 1439H

[BX] ← 1439
if BX = 2090H
then [2090H] ← 39H
[2091H] ← 14H

* Mov [BX + DISP], 58H

Suppose BX = 1090H, Disp → 05H
then 1090+05 = 1095

58H moves at memory location 1095H

* Mov [BX+SI], 1593

BX → 1030H
SI → 2040H

BX+SI = 3070H

3070H ← 93H
3071H ← 15H

* Mov [BX+SI], 57H

3070H ← 57H

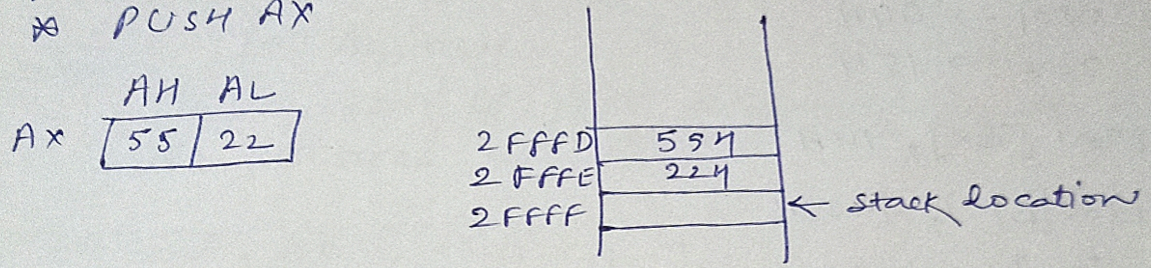
* Mov [BX+SI+DISP], 1439H

If [BX]=0301, [SI]= 1548, DISP=04
[BX+SI + DISP] = 184D
So.

184D ← 39H
184E ← 14H

PUSH : Push to stack!- This instruction pushes the content of specified reg./memory location on to the stack. After each execution stack pointer is decremented by 2.

* PUSH AX



* PUSHDS

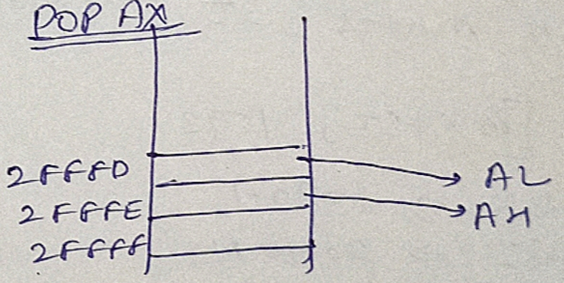
* PUSH [5000H] → The content of location 5000H and 5001H in DS are pushed on to the stack.

POP: Pop from stack:- The pop instruction serves exactly opposite to the PUSH instruction.

* POP AX

* POP DS

* POP [5000H]



XCHG!- Exchange → This instruction exchanges the content of specified source and destination operands which may be register or one of them may be memory location. However exchange of contents of two memory location not permitted.

e.g. XCHG [5000H], AX

XCHG AX, BX

IN: Input the port:- This instruction is used for reading an input port. The address of the input port may be specified in the instruction directly or indirectly.

e.g. IN AL, 03H

The data of port address 03H is read and stores in AL.

IN DX, AX

This instruction reads data from a 16 bit port whose address is in AX and stores in DX.

In IN instruction port add^ss is source and another Reg. / memory location is destination.

OUT: Output to the port:- This instruction is used for writing to an output port.

OUT 03H, AL

OUT DX, AX

In OUT instruction destination is port add^ss.

LDS/LES → Load Data/ Extra segment [LDS BX, [5000]].

XLAT → Translate:- It is used for finding out the codes in case of code conversion problems. ~~using look up it~~.

LEA Reg, Mem: Load Effective Address:- This instruction load the offset address into the specified reg^r. It can also determine the offset address of a variable memory location specified in the instruction to load it into the specified Reg^r.

Eg. LEA CX, [BX + SI + 0532H]

LAHF: Load AH Reg^r with lower bit of flag Reg^r. This instruction loads low order 8 bit of flag reg^r into AH Reg^r.

SAHF: - This instruction stores the content of AH Reg^r into low-order bits of flag Reg^r.

Arithmatic Instruction: - Instructions of this group perform addition, sub., multiplication, division, increment, decrement, comparision etc.

Eg. ADD R, M → ADD AL[3010]

ADD M, R

ADD R, R

ADD R, d8 , ADD R, d16

The content of source & destination are added and answer lie in destination.

ADD Ax, Bx

Ax [20 | 43]   Bx [10 | 39]

Ax + Bx → Ans.
↓
{Ax destination

ADD AX, 2013

Ax [20 | 43]

  2043H
+ 2013H
Ans. → AX

* ADC : Add with carry.

* SUB: Subtract

* SBB subtraction with borrow

Multiplication Instructions:

* MUL M/R (Unsigned 8 bit or 16 bit multiplication)

This is an instruction to multiply two unsigned number. If the content of a specified reg^{r} or memory location is 8 bit, it is multiplied by the content of AL. If the content of specified Reg. or memory location is 16 bit, it is multiplied by the content of AX reg. The result of multiplication is kept in the AX reg for 8 bit multiplication or in DX:AX for 16 bit multiplication. In case of 16 bit multiplication higher order 16 bits of the result are placed in DX Reg^{r} and low order 16 bit of the result is placed in AX Reg^{r}.

In either 8 bit or 16 bit multiplication, if the high order half of the result is zero, the overflow flag (OF) and carry flag (CF) are set to zero. If the high order result is non zero, OF & CF are set to 1 to indicate that significant digits of the result are in AH or DX Reg^{r}.

```
MUL CL ; AX ← (AL) X (CL)
MUL CX ; (DX) (AX) ← (AX) X (CX)
```

*

IMUL M/R (signed 8 bit or 16 bit multiplication)

This is a ~~a~~ instruction for multiplication of two signed number. The result is a signed number.

Detail is same as previous ins^n

Division Instruction:-

DIV M, R (Unsigned 8 bit or 16 bit divison)

This Ins^n is used to ~~define~~ divide a 16 bit unsigned number by an 8 bit unsigned number or 32 bit unsigned number by a 16 bit unsigned number. The 32 bit unsigned dividend is placed in DX and AX Reg^r. The 16 bit divisor is placed in a specified 16 bit Reg^r or memory location. After the execution of the instruction, the 16 bit quotient is placed in the AX Reg^r and 16 bit remainder is placed in DX Reg^r.

If a 16 bit number is divided by an 8 bit number, ~~the dividend is extended to~~ 16 ~~bit by putting Zero in MSB position~~. Divident is placed in AX and divisor is placed in 8 bit Reg^r. or memory. After the execution of the ins^n 8 bit quotient is placed in AL Reg^r and 8 bit Remainder is placed in AH Reg^r.

IDIV used for signed 8 bit or 16 bit division.

INC Reg. Increment 16 bit Reg^r

This is a 16 bit increment instruction

Segment Reg^rs are not incremented by this ins^n

DEC R/M   Decrement 16 bit Regr or memory
Not specified for segment Regr.

Decimal Adjust Instructions

(i) DAA (Decimal Adjust After BCD Addition)

When two BCD numbers are added, the DAA instruction is used after ADD or ADC insn to get correct answer in BCD.

(ii) DAS (Decimal Adjust After BCD Subtraction)

This instruction is used to get correct result in BCD, when a BCD number is subtracted from another BCD number. It is used after SUB, SBB instruction.

Ex. If AL= 53H , CL= 29H

ADD AL, CL

AL ← AL+CL

AL ← 53H+ 29H

AL ← 7C H

DAA ; AL ← 7C+ 06 (as C > 9)

CBW → Convert signed Byte to signed word →

This instruction copies the sign bit of oprand in AL Regr to all the bits of AH Reg. If signed bit is 1 ~~If high bit of~~ AL= ~~then~~

then AH = 11111111
= FFH

AL remains as before

If signed ~~byte~~ bit is 0

then AH = 00000000
= 00H

CWD (convert signed word to signed Double word)
CWD instruction extends the sign bit of a word in AX register to all the bits of DX Regr.

NEG M/R (Negate or take negative or get 2's complement)
This instruction obtains 2's complement of the content of an 8 bit or 16 bit specified register or memory location. It obtains 2's complement of a number by subtracting the given number from zero.

NEG AX , NEG BL , NEG [F001]

CMP: Compare:- This instruction compares the source oprand to destination oprand. For comparision it subtracts source oprand from destination oprand but does not store the result, only flags are affected depending on the result.

Ex. (i) CMP BX , 0100H

BX	20	13H

2013H compared with 0100H

2013H - 0100H = Answer.

(ii) CMP [5000H], 0100H

(iii) CMP BX , [SI]

(iv) CMP BX, CX

If source (S) = Destination (D)	Zeroflag = 1
S > D	CY = 1
~~B~~ S < D	CY = 0

IC 8255 Programmable Peripheral Interface(PPI) Parallel I/O device

The 8255 is a programmable peripheral interface i.e. PPI 8255.It is a general purpose programmable parallel I/O device.It can be used with almost any microprocessor.It consists of three 8-bit bidirectional I/O ports (24 I/O lines) which can be configured as per the requirement.

Features of IC 8255:-

1. It consists of 3, 8-bit IO ports i.e. Port A, Port B, and Port C.Port C can be divided into two ports of 4 bits PortC$_{\text{UPPER}}$ and Port C$_{\text{lower}}$. These ports are controlled by 2 groups: Group A and Group B.
2. Address/data bus must be externally demux'd.
3. It is TTL compatible.
4. It has improved DC driving capability.
5. Fully compatible with Intel microprocessor family.
6. Direct bit set/reset capability is available for port C.
7. It can be operate in 3 modes:-

a) Mode 0-simple I/O

b) Mode 1-strobed I/O

c) Mode 2-strobed bi-directional I/O

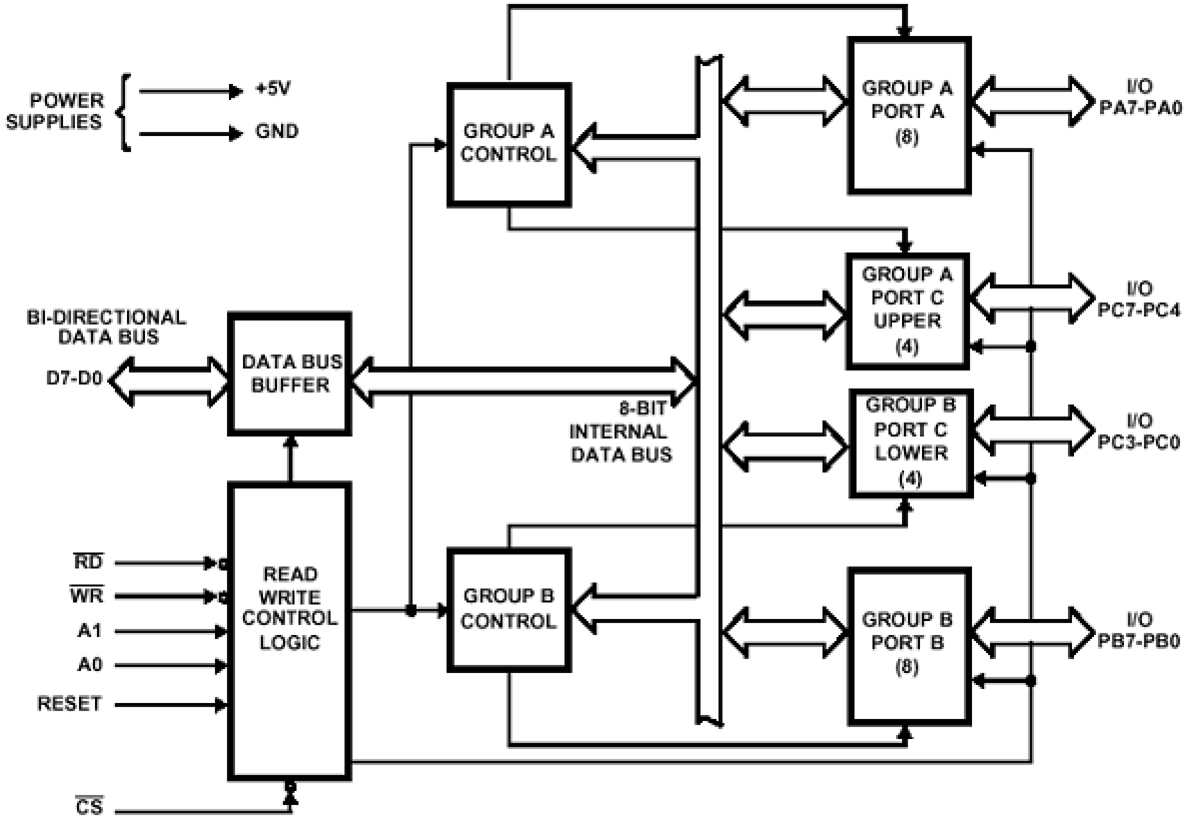
The block diagram of 8255 is as shown in fig.

It contains following blocks:

1. Data bus buffer
2. Read/write control logic
3. Group A and group B control
4. Port A and port B
5. Port C

Data bus buffer:-

It is a tri-state 8-bit buffer, which is used to interface the microprocessor to the system data bus. Data is transmitted or received by the buffer as per the instructions by the CPU. The direction of data buffer is decided by read and write control signals. When read is activated, it transmits data to the system data bus. When write is activated, it receives data from system data bus.



Read/write control logic:-

This block is responsible for controlling the internal/external transfer of data/control/status word. It accepts the input from the CPU address and control buses, and in turn issues command to both the control groups.

CS

It stands for Chip Select. A LOW on this input selects the chip and enables the communication between the 8255A and the CPU. It is connected to the decoded address, and A_0 & A_1 are connected to the microprocessor address lines.

Their result depends on the following conditions −

CS	A_1	A_0	Result
0	0	0	PORT A

0	0	1	PORT B
0	1	0	PORT C
0	1	1	Control Register
1	X	X	No Selection

WR

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

RESET

This is an active high signal. It clears the control register and sets all ports in the input mode.

RD

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

A_0 and A_1

These input signals work with RD, WR, and one of the control signal. Following is the table showing their various signals with their result.

A_1	A_0	RD	WR	CS	Result
0	0	0	1	0	**Input Operation** PORT A → Data Bus
0	1	0	1	0	PORT B → Data Bus
1	0	0	1	0	PORT C → Data Bus

0	0	1	0	0	**Output Operation** Data Bus → PORT A
0	1	1	0	0	Data Bus → PORT A
1	0	1	0	0	Data Bus → PORT B
1	1	1	0	0	Data Bus → PORT D

Group A and group B control:-

1. Group A consisting of port A and upper part of port C.
2. Group B consisting of port B and lower part of port C.

   These block receive control from the CPU and issues commands to their respective ports.

   Group A - PA and PCU (PC7 –PC 4)

   Group B - PCL (PC3 – PC 0)

   Port A: This has an 8 bit latched/buffered O/P and 8 bit input latch. It can be programmed in 3 modes – mode 0, mode 1, mode 2.

   Port B: This has an 8 bit latched / buffered O/P and 8 bit input latch. It can be programmed in mode 0, mode1.

   Port C : This has an 8 bit latched input buffer and 8 bit output latched/buffer. This port can be divided into two 4 bit ports and can be used as control signals for port A and port B. it can be programmed in mode 0.

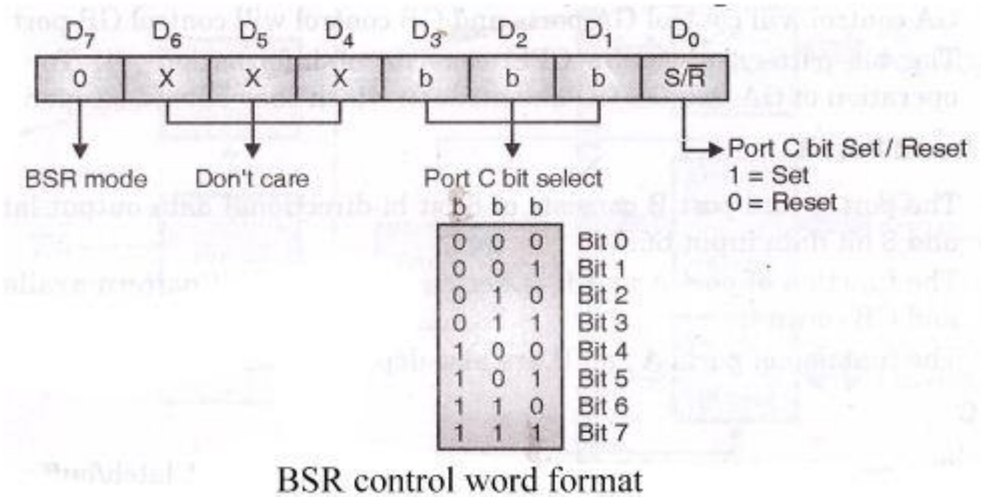
8255 operating modes:-

1. The 8255 IC provides one control word register.
2. It is selected when A0=1,A1=1,CS =0 and WR=0.The read operation is not allowed for control register.
3. The bit pattern loaded in control word register specifies an I/O function for each port and the mode of operation in which the ports are to be used.
4. There are two different control word formats which specify two basic modes:

- **BSR (Bit set reset) mode**
- **I/O mode**

1. The two basic modes are selected by D7 bit of control register. When D7=1 it is an I/O mode and when D7=0; it is a BSR mode.

BSR mode:-


D_7 D_6 D_5 D_4 D_3 D_2 D_1 D_0
0 X X X b b b $S/\overline{R}$
BSR mode
Don't care
Port C bit select
Port C bit Set / Reset
1 = Set
0 = Reset

b	b	b	
0	0	0	Bit 0
0	0	1	Bit 1
0	1	0	Bit 2
0	1	1	Bit 3
1	0	0	Bit 4
1	0	1	Bit 5
1	1	0	Bit 6
1	1	1	Bit 7

BSR control word format

1. The BSR mode is a port C bit set/reset mode.
2. The individual bit of port C can be set or reset by writing control word in the control register.
3. The control word format of BSR mode is as shown in the fig.
4. The pin of port C is selected using bit select bits [b b b] and set or reset is decided by bit $S/\overline{R}$.
5. The BSR mode affects only one bit of port C at a time. The bit set using BSR mode remains set unless and until you change the bit. So to set any bit of port C, bit pattern is loaded in control register.
6. If a BSR mode is selected it will not affect I/O mode. **I/O modes-**

There are three I/O modes of operation:

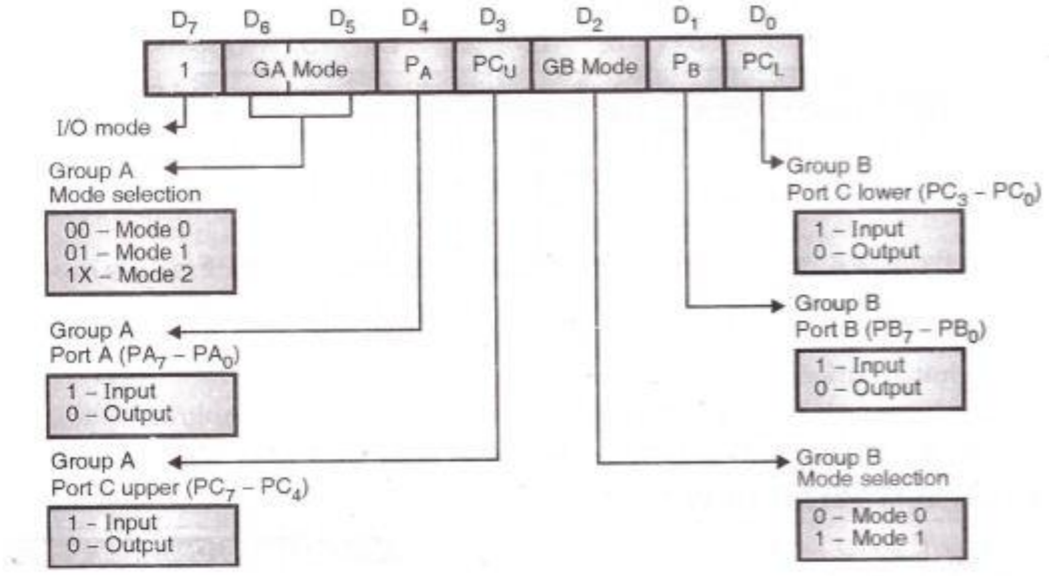
• Mode 0- Basic I/O

• Mode 1- Strobed I/O

• Mode 2- Bi-directional I/O

The I/O modes are programmed using control register.

The control word format of I/O modes is as shown in the figure below:



I/O modes control word format

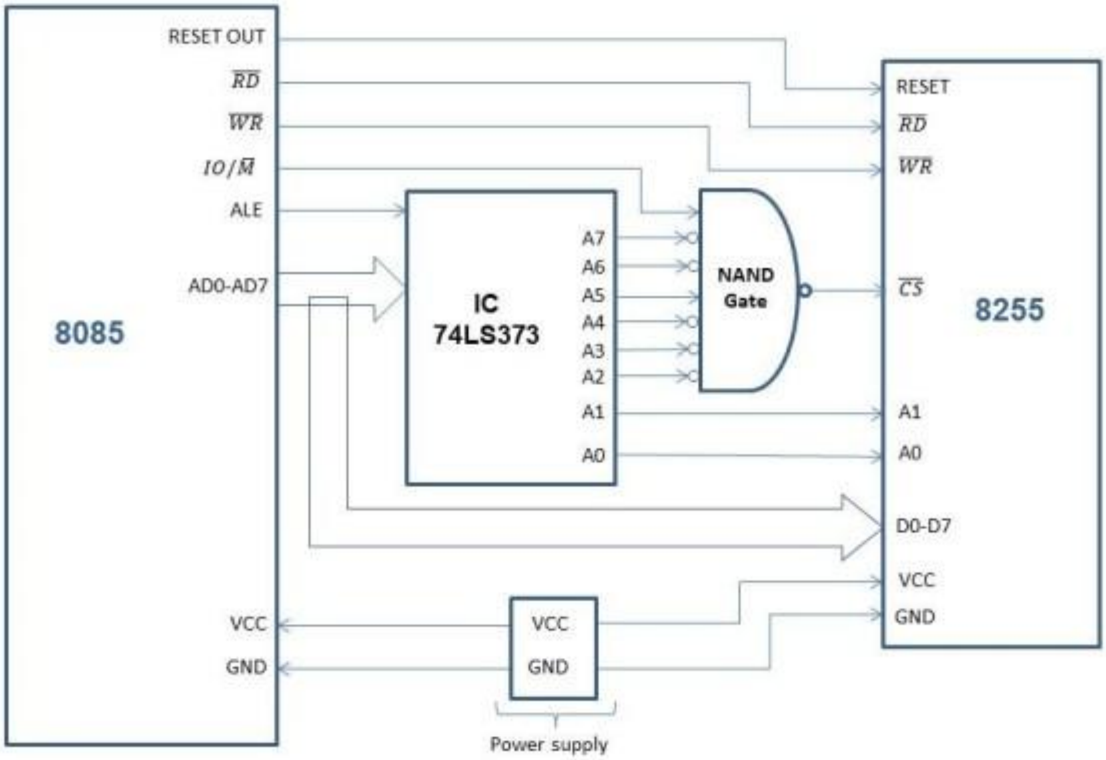
Function of each bit is as follows:

1. D7 – When the bit D7 = 1 then I/O mode is selected, if D7=0 then BSR mode is selected. The function of bits D0 to D6 is independent on mode (I/O mode or BSR mode).
2. D6 and D5 – In I/O mode the bits D6 and D5 specifies the different I/O modes for group A i.e. Mode 0, Mode 1 and Mode 2 for port A and port C upper.
3. D4 and D3 – In I/O mode the bits D4 and D3 selects the port function for group A. If these bits = 1 the respective port specified is used as input port. But if bit =0, the port is used as output port.
4. D2 – In I/O mode the bit D2 specifies the different I/O modes for group B i.e. Mode 0 and Mode 1 for port B and port C lower.
5. D1 and D0 – In I/O mode the bits D1 and D0 selects the port function for group B. If these bits = 1 the respective port specified is used as input port. But if bit = 0, the port is used as output port.

All the 3 modes i.e. Mode 0, Mode 1 and Mode 2 are only for group A ports, but for group B only 2 modes i.e. Mode 0 and Mode 1 are provided. When 8255 is reset, it will clear control word register contents and all the ports are set to input mode. The ports of 8255 can be programmed for other modes by sending appropriate bit pattern to control register.

Interfacing of 8085 with 8255 Programmable Peripheral Interface

Circuit diagram to connect the 8085 with 8255



You can understand the connections in the following steps.

- First of all, we need supply power to both 8085 and 8255 by connecting VCC and GND pins to the appropriate sources.
- RESET OUT pin of 8085 is connected to the RESET pin of 8255. When the microprocessor is reset, it also resets the 8255 via this connection.
- RD* and WD* pins of 8085 are connected to the RD* and WD* pins of 8255, respectively. 8085 conveys the type of operation (read or write) to 8255 via these connections.
- As we have come to know in this 8085 course, pins AD0-AD7 have dual functionality. They act as both; address bus and data bus. This is achieved by multiplexing. To extract address from it, we need to demultiplex it, which is achieved using the IC 74LS373. To learn in detail about the demultiplexing process, you can check out this section on **demultiplexing AD0-AD7**.

- After demultiplexing, pins A0 and A1 (from the output of IC74LS373) are connected to the pins A0 and A1 of 8255, respectively. These two pins tell 8255 about which port the microprocessor is talking about.

- The last part of making the connections is the chip select logic.

Chip select logic

For the microprocessor to be able to communicate with 8255, the CS* (active low chip select signal) should be low. So, we design a logic circuit that takes address lines AD2-AD7 as inputs and CS* signal as output. The design should be such that if the address allotted to any of the ports A, B, or C appears on AD0-AD1, output CS* should go low.

Let us allot the address to the ports A, B, and C of 8255.

- Port A: 0010 0000 (20H)
- Port B: 0010 0001 (21H)
- Port C: 0010 0010 (22H)
- Control port: 00100011 (23H)

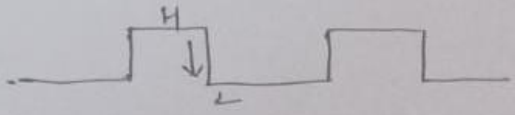
The last two bits are A0 and A1. The rest of the 6 bits are used to generate chip select signals.

Programmable Interval Timer (8253/8254):-

Its function is similar to the software designed counters and timers. It can be generate accurate time delays and can be used for applications such as real time clock.

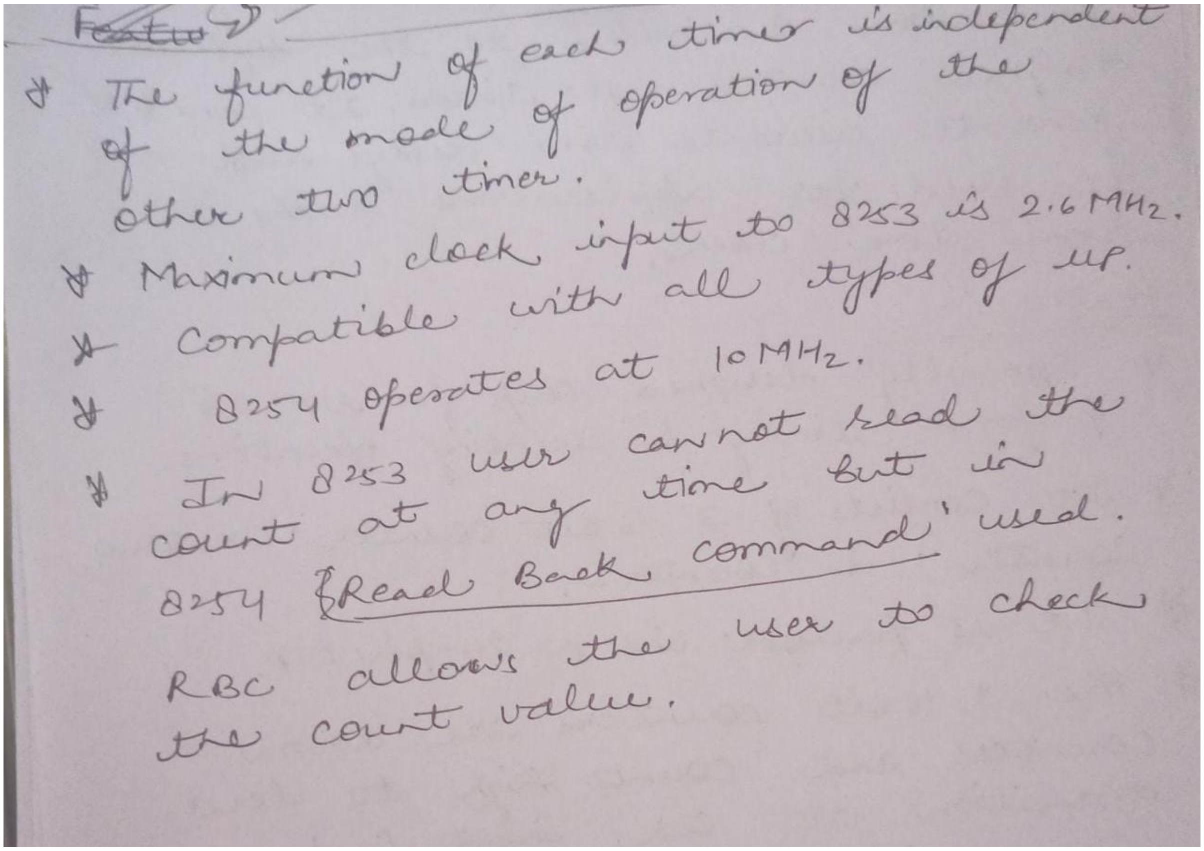
Features:-

- specially designed chip for uP to perform timing & counting operations.
- It consists of 3, 16 bit counter: Counter0 Counter 1 & Counter2.
- It is packaged in a 24-pin DIP.
- The 3, 16 bit counters are down counters and counts high to low transition at Clk input. (100, 99... 0)



- Each timer can be programmed in any one of 6 modes.
- After reaching at low that means at 0 or at the ~~count~~ end of the count, it generates a pulse that can be used to interrupt the MPU.

either in



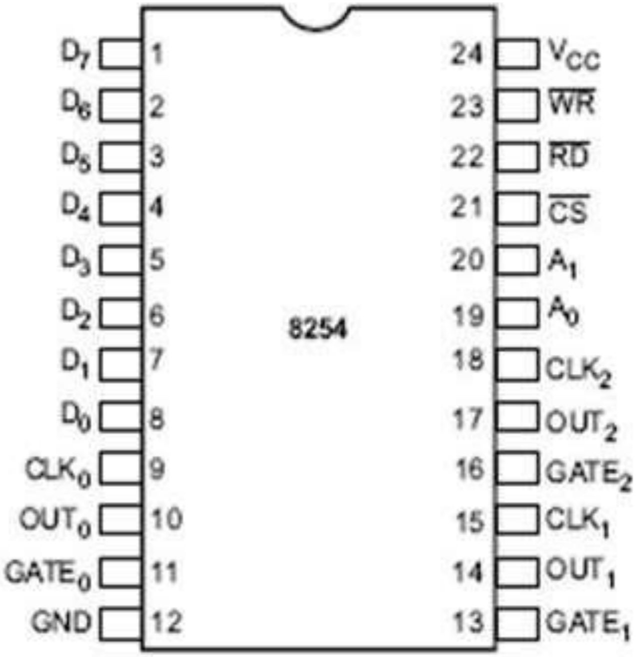
Featur ⇒

- The function of each timer is independent of the mode of operation of the other two timer.
- Maximum clock input to 8253 is 2.6 MHz.
- Compatible with all types of uP.
- 8254 operates at 10 MHz.
- In 8253 user can not read the count at any time but in 8254 {Read Back command' used.

RBC allows the user to check the count value.

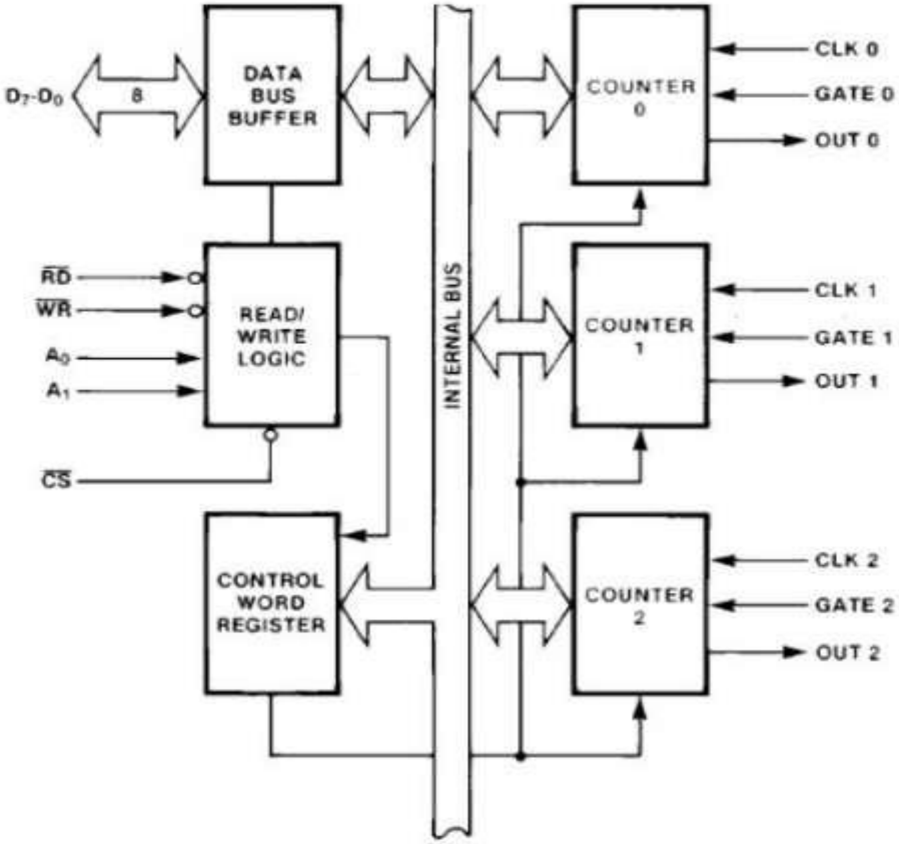
8254 Pin Description

Here is the pin diagram of 8254 −



8254 block diagram:

The architecture of 8254 looks as follows −



Data Bus Buffer

It is a tri-state, bi-directional, 8-bit buffer, which is used to interface the 8253/54 to the system data bus. It has three basic functions −

- Programming the modes of 8253/54.
- Loading the count value in timers.
- Reading the count values from timers

Data bus buffer is connected to D0-D7 pins of microprocessor.

Read/Write Logic

It includes 5 signals, i.e. RD, WR, CS, and the address lines A_0 & A_1. In the peripheral I/O mode, the RD and WR signals are connected to IOR and IOW, respectively. In the memory mapped I/O mode, these are connected to MEMR and MEMW.

RD'- At low signal CPU is reading data from 8253 in the form of count value.

WR'- At low signal CPU is WRITING DATA TO 8253 in the form of mode information or loading count values.

Address lines A_0 & A_1 of the CPU are connected to lines A_0 and A_1 of the 8253/54, and CS is tied to a decoded address. The control word register and counters are selected according to the signals on lines A_0 & A_1.

A_1	**A_0**	**Result**
0	0	Counter 0
0	1	Counter 1
1	0	Counter 2
1	1	Control Word Register
X	X	No Selection

Control Word Register

This register is accessed when lines A_0 & A_1 are at logic 1. It is used to write a command word, which specifies the counter to be used, its mode, and either a read or write operation. Following table shows the result for various control inputs.

A_1	**A_0**	**RD**	**WR**	**CS**	**Result**
0	0	1	0	0	Write Counter 0

0	1	1	0	0	Write Counter 1
1	0	1	0	0	Write Counter 2
1	1	1	0	0	Write Control Word
0	0	0	1	0	Read Counter 0
0	1	0	1	0	Read Counter 1
1	0	0	1	0	Read Counter 2
1	1	0	1	0	No operation
X	X	1	1	0	No operation
X	X	X	X	1	No operation

Counters

Each counter contains a single, 16 bit-down counter, which can perform operations in either binary or BCD. Its input and output signals are configured by the mode selection that are stored in the control word register. The programmer can have the accessibility to read the contents of any of the three counters without getting effected with the actual count in process.

These 3 counters have 3 signals:

1. CLK
2. GATE
3. OUT

CLK: Clock input pin provides 16 bit timer with the signal that causes the timer to decrement.

GATE: It used to initiate or enable counting.

The effects of gate signal depend on which of the 6 modes of operation is chosen.

OUT: It provides the output from the timer.

Operating Modes of 8253/8254:

Mode 0 ─ Interrupt on Terminal Count

- It is used to generate an interrupt to the microprocessor after a certain interval.
- Initially the output is low after the mode is set. The output remains LOW after the count value is loaded into the counter.
- The process of decrementing the counter continues till the terminal count is reached, i.e., the count become zero and the output goes HIGH and will remain high until it reloads a new count.
- The GATE signal is high for normal counting. When GATE goes low, counting is terminated and the current count is latched till the GATE goes high again.

Mode 1 – Programmable One Shot

- It can be used as a mono stable multi-vibrator.
- The gate input is used as a trigger input in this mode.
- The output remains high until the count is loaded and a trigger is applied.

Mode 2 – Rate Generator

- The output is normally high after initialization.
- Whenever the count becomes zero, another low pulse is generated at the output and the counter will be reloaded.

Mode 3 – Square Wave Generator

- This mode is similar to Mode 2 except the output remains low for half of the timer period and high for the other half of the period.

Mode 4 − Software Triggered Mode

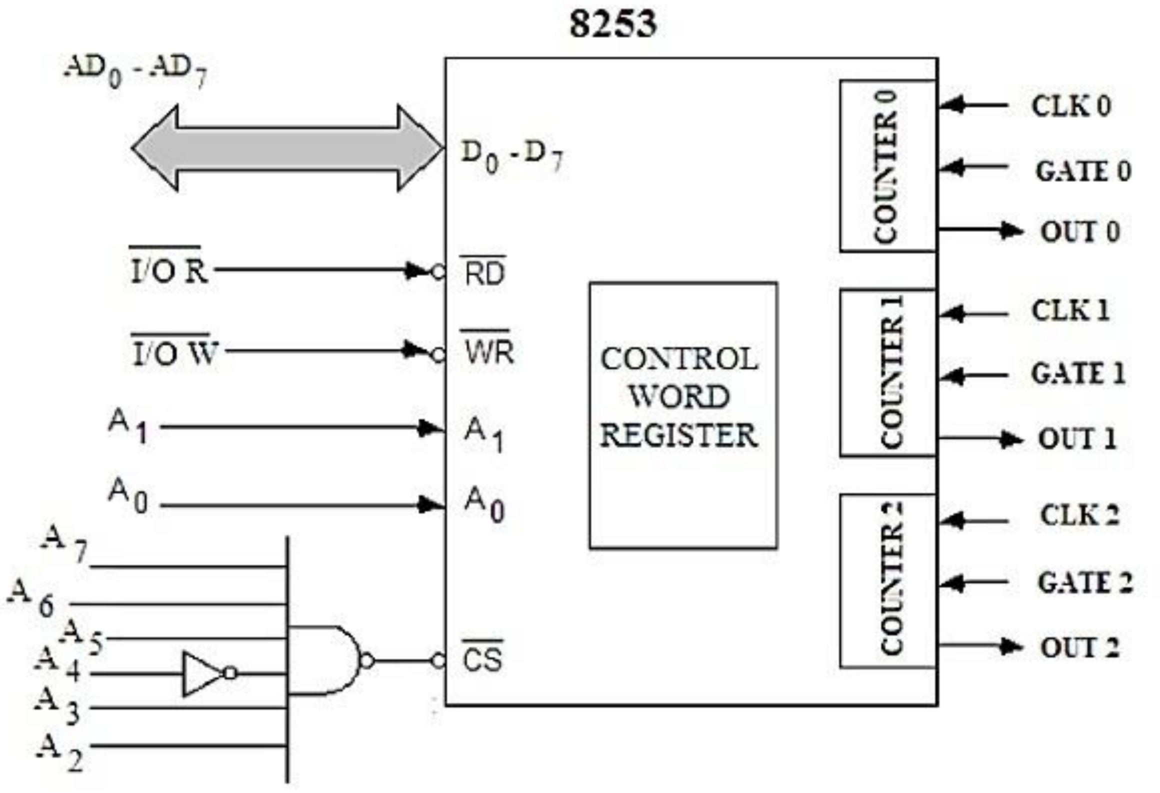
- In this mode, the output will remain high until the timer has counted to zero, at which point the output will pulse low and then go high again.
- The count is latched when the GATE signal goes LOW.
- On the terminal count, the output goes low for one clock cycle then goes HIGH. This low pulse can be used as a strobe.

Mode 5 – Hardware Triggered Mode

- This mode generates a strobe in response to an externally generated signal.
- This mode is similar to mode 4 except that the counting is initiated by a signal at the gate input, which means it is hardware triggered instead of software triggered.
- After it is initialized, the output goes high.
- When the terminal count is reached, the output goes low for one clock cycle.

Interfacing 8253 with 8085

In that diagram, we can easily find that when A_{3-2} and A_{7-5} are at logic 0 and A_4 at logic 1, then only the chip select CS pin of 8253 will be enabled.



This table is showing how the counter is being selected by using A_1 and A_0 pins of 8253.

CS								**HEX Address**	**Counter Selection**
A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0		
0	0	0	1	0	0	0	0	10H	Counter 0
0	0	0	1	0	0	0	1	11H	Counter 1
0	0	0	1	0	0	1	0	12H	Counter 2
0	0	0	1	0	0	1	1	13H	Control Word Register

8259 PIC Microprocessor

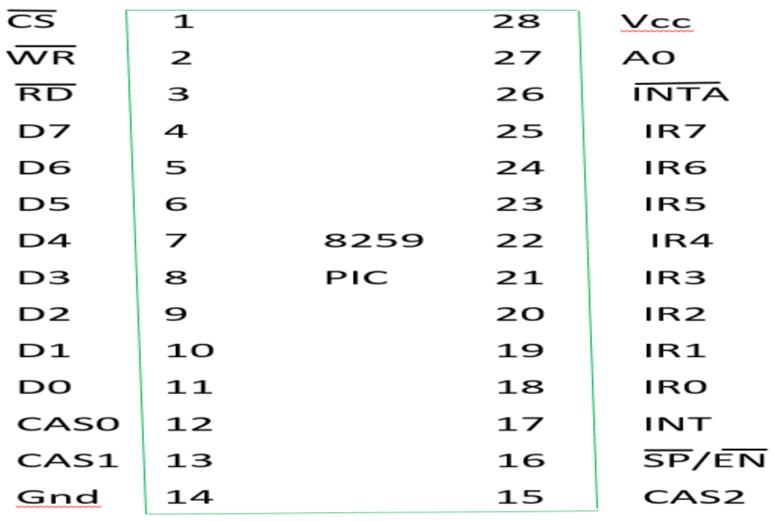
8259 microprocessor is defined as **Programmable Interrupt Controller (PIC)** microprocessor. There are 5 hardware interrupts and 2 hardware interrupts in 8085 and 8086 respectively. But by connecting 8259 with CPU, we can increase the interrupt handling capability. 8259 combines the multi interrupt input sources into a single interrupt output. Interfacing of single PIC provides 8 interrupts inputs from IR0-IR7.

For example, Interfacing of 8085 and 8259 increases the interrupt handling capability of 8085 microprocessor from 5 to 8 interrupt levels.

Features of 8259 PIC microprocessor –

1. Intel 8259 is designed for Intel 8085 and Intel 8086 microprocessor.
2. It can be programmed either in level triggered or in edge triggered interrupt level.
3. We can masked individual bits of interrupt request register.
4. We can increase interrupt handling capability upto 64 interrupt level by cascading further 8259 PIC.
5. Clock cycle is not required.

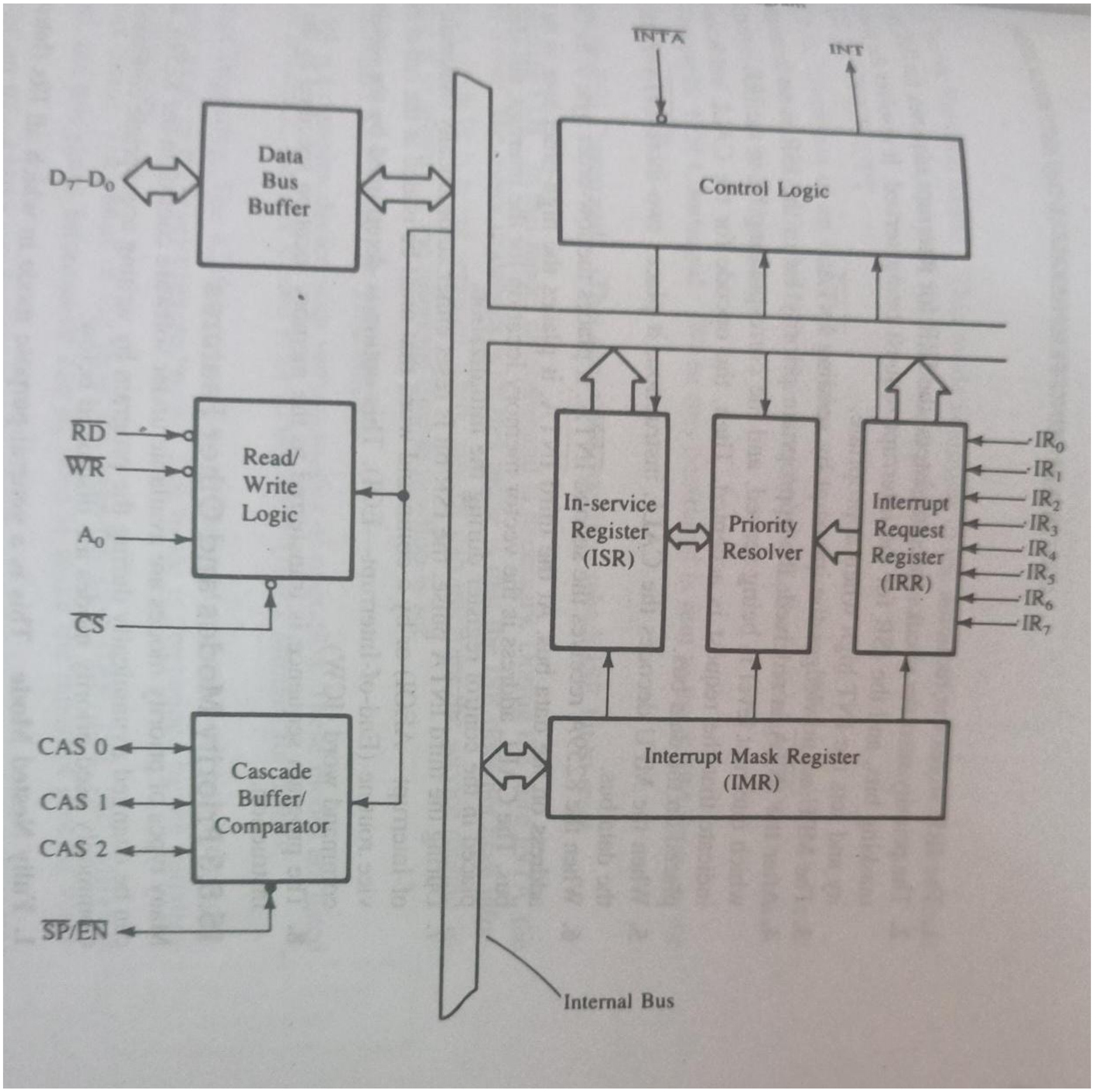
Pin Diagram of 8259 –



$\overline{CS}$	1		28	Vcc
$\overline{WR}$	2		27	A0
$\overline{RD}$	3		26	$\overline{INTA}$
D7	4		25	IR7
D6	5		24	IR6
D5	6		23	IR5
D4	7	8259	22	IR4
D3	8	PIC	21	IR3
D2	9		20	IR2
D1	10		19	IR1
D0	11		18	IR0
CAS0	12		17	INT
CAS1	13		16	$\overline{SP}/\overline{EN}$
Gnd	14		15	CAS2

We can see through above diagram that there are total 28 pins in 8259 PIC microprocessor where Vcc :5V Power supply and Gnd: ground. Other pins use are explained below.

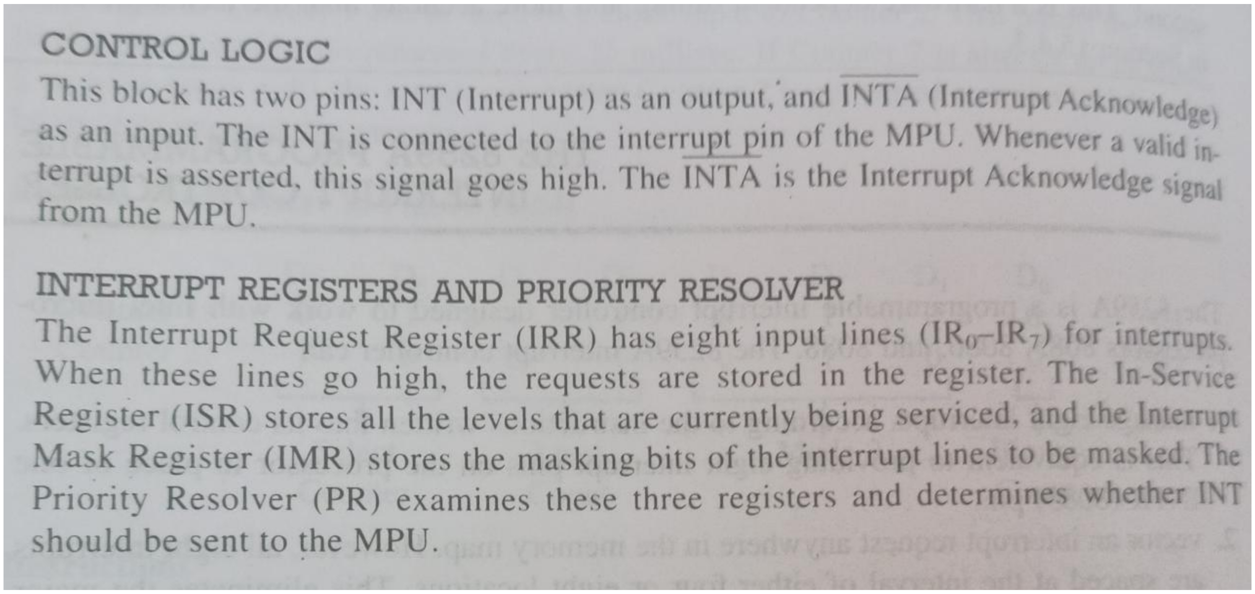
Block Diagram of 8259 PIC microprocessor –



The Block Diagram consists of 8 blocks which are – Data Bus Buffer, Read/Write Logic, Cascade Buffer Comparator, Control Logic, Priority Resolver and 3 registers- ISR, IRR, IMR.

Data bus buffer –
This Block is used as a mediator between 8259 and 8085/8086 microprocessor by acting as a buffer. It takes the control word from the 8085 (let say) microprocessor and transfer it to the control logic of 8259 microprocessor. Also, after selection of Interrupt by 8259 microprocessor, it transfer the opcode of the selected Interrupt and address of the Interrupt service sub routine to the other connected microprocessor. The data bus buffer consists of 8 bits represented as D0-D7 in the block diagram. Thus, shows that a maximum of 8 bits data can be transferred at a time.

Read/Write logic –
This block works only when the value of pin CS is low (as this pin is active low). This block is responsible for the flow of data depending upon the inputs of RD and WR. These two pins are active low pins used for read and write operations.



CONTROL LOGIC

This block has two pins: INT (Interrupt) as an output, and $\overline{INTA}$ (Interrupt Acknowledge) as an input. The INT is connected to the interrupt pin of the MPU. Whenever a valid interrupt is asserted, this signal goes high. The $\overline{INTA}$ is the Interrupt Acknowledge signal from the MPU.

INTERRUPT REGISTERS AND PRIORITY RESOLVER

The Interrupt Request Register (IRR) has eight input lines (IR_0–IR_7) for interrupts. When these lines go high, the requests are stored in the register. The In-Service Register (ISR) stores all the levels that are currently being serviced, and the Interrupt Mask Register (IMR) stores the masking bits of the interrupt lines to be masked. The Priority Resolver (PR) examines these three registers and determines whether INT should be sent to the MPU.

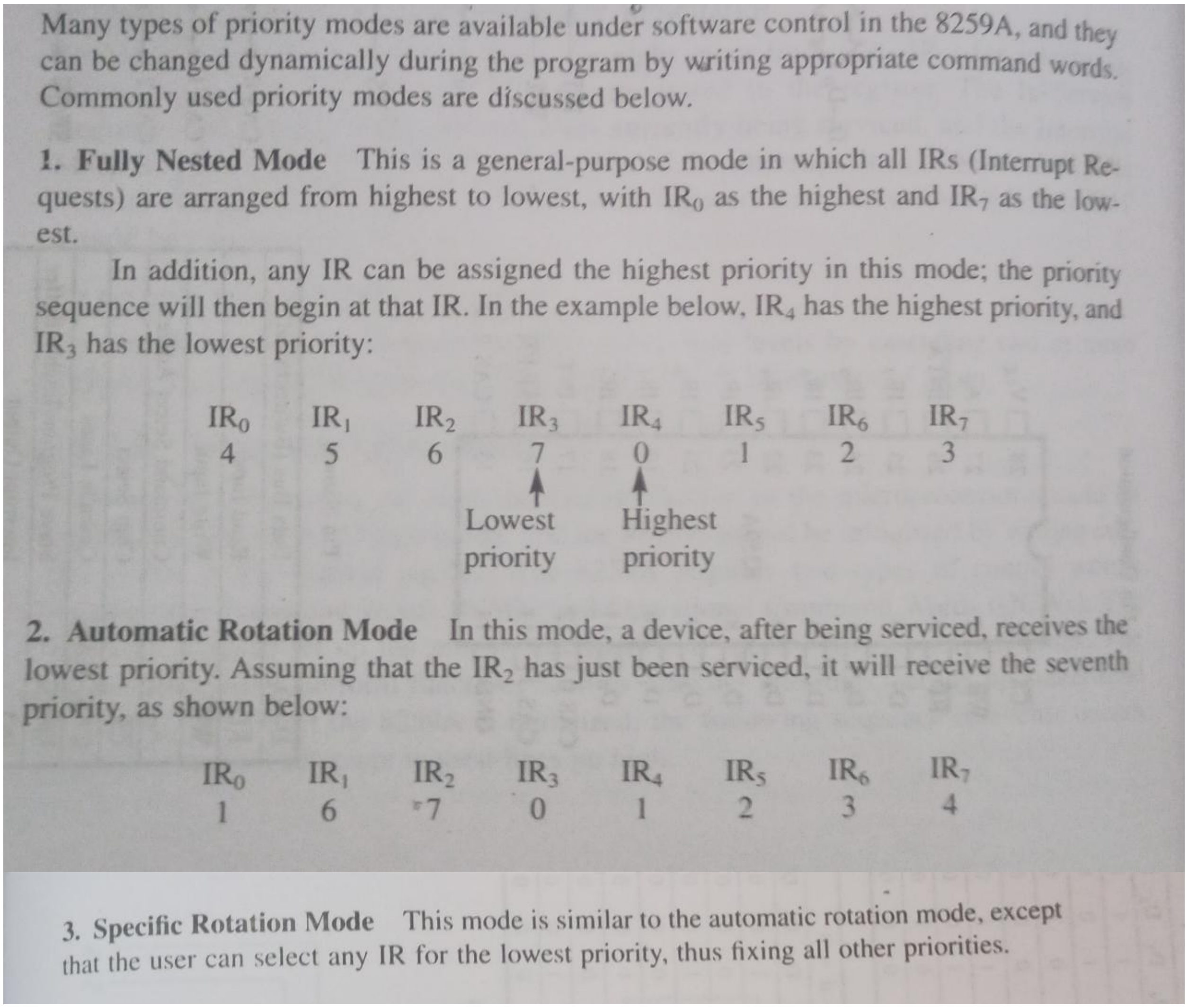
1. **Priority resolver –**
   It examines all the three registers and set the priority of interrupts and according to the priority of the interrupts, interrupt with highest priority is set in ISR register. Also, it reset the interrupt level which is already been serviced in IRR.

2. **Cascade buffer –**
   To increase the Interrupt handling capability, we can further cascade more

number of pins by using cascade buffer. So, during increment of interrupt capability, CSA lines are used to control multiple interrupt structure.

SP/EN (Slave program/Enable buffer) pin is when set to high, works in master mode else in slave mode. In Non Buffered mode, SP/EN pin is used to specify whether 8259 work as master or slave and in Buffered mode, SP/EN pin is used as an output to enable data bus.

PRIORITY MODES OF 825/8254:



Many types of priority modes are available under software control in the 8259A, and they can be changed dynamically during the program by writing appropriate command words. Commonly used priority modes are discussed below.

1. Fully Nested Mode This is a general-purpose mode in which all IRs (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 priority in this mode; the priority sequence will then begin at that IR. In the example below, IR_4 has the highest priority, and IR_3 has the lowest priority:

IR_0	IR_1	IR_2	IR_3	IR_4	IR_5	IR_6	IR_7
4	5	6	7	0	1	2	3
			↑ Lowest priority	↑ Highest priority			

2. Automatic Rotation Mode In this mode, a device, after being serviced, receives the lowest priority. Assuming that the IR_2 has just been serviced, it will receive the seventh priority, as shown below:

IR_0	IR_1	IR_2	IR_3	IR_4	IR_5	IR_6	IR_7
1	6	7	0	1	2	3	4

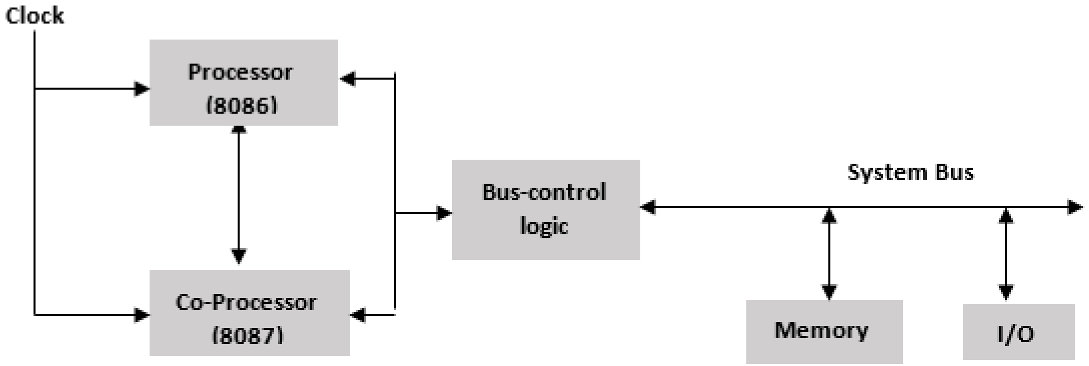
3. Specific Rotation Mode This mode is similar to the automatic rotation mode, except that the user can select any IR for the lowest priority, thus fixing all other priorities.

Coprocessor

A Coprocessor is a specially designed circuit on microprocessor chip which can perform the same task very quickly, which the microprocessor performs. It reduces the work load of the main processor. The coprocessor shares the same memory, IO system, bus, control logic and clock generator. The coprocessor handles specialized tasks like mathematical calculations, graphical display on screen, etc.

The 8086 and 8088 can perform most of the operations but their instruction set is not able to perform complex mathematical operations, so in these cases the microprocessor requires the math coprocessor like Intel 8087 math coprocessor, which can easily perform these operations very quickly.

Block Diagram of Coprocessor Configuration



8087 numeric data processor is also known as **Math co-processor, Numeric processor extension** and **Floating point unit**. It was the first math coprocessor designed by Intel to pair with 8086/8088 resulting in easier and faster calculation.

Once the instructions are identified by the 8086/8088 processor, then it is allotted to the 8087 co-processor for further execution.

The data types supported by 8087 are −

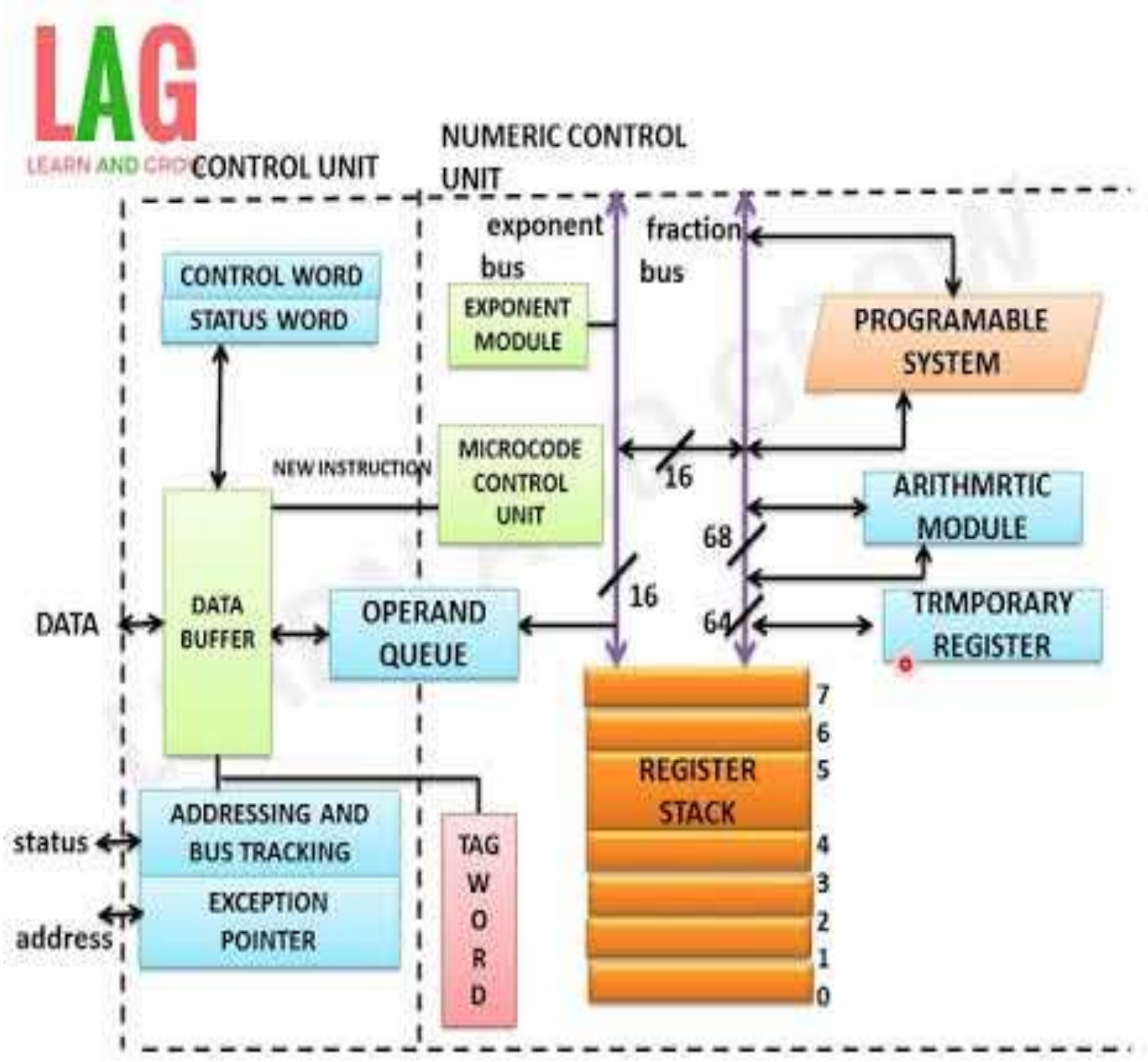
- Binary Integers
- Packed decimal numbers
- Real numbers
- Temporary real format

The most prominent features of 8087 numeric data processor are as follows −

- It supports data of type integer, float, and real types ranging from 2-10 bytes.

- The processing speed is so high that it can calculate multiplication of two 64-bits real numbers in ~27 µs and can also calculate square-root in ~35 µs.

8087 Architecture



8087 Architecture is divided into two groups, i.e., **Control Unit** (CU) and **Numeric Extension Unit** (NEU).

- The **control unit** handles all the communication between the processor and the memory such as it receives and decodes instructions, reads and writes memory operands, maintains parallel queue, etc. All the coprocessor instructions are ESC instructions, i.e., they start with ‘F’, the coprocessor only executes the ESC instructions while other instructions are executed by the microprocessor.

Instruction queue:

8087 maintains an instruction queue identical to that of the host processor (8086).

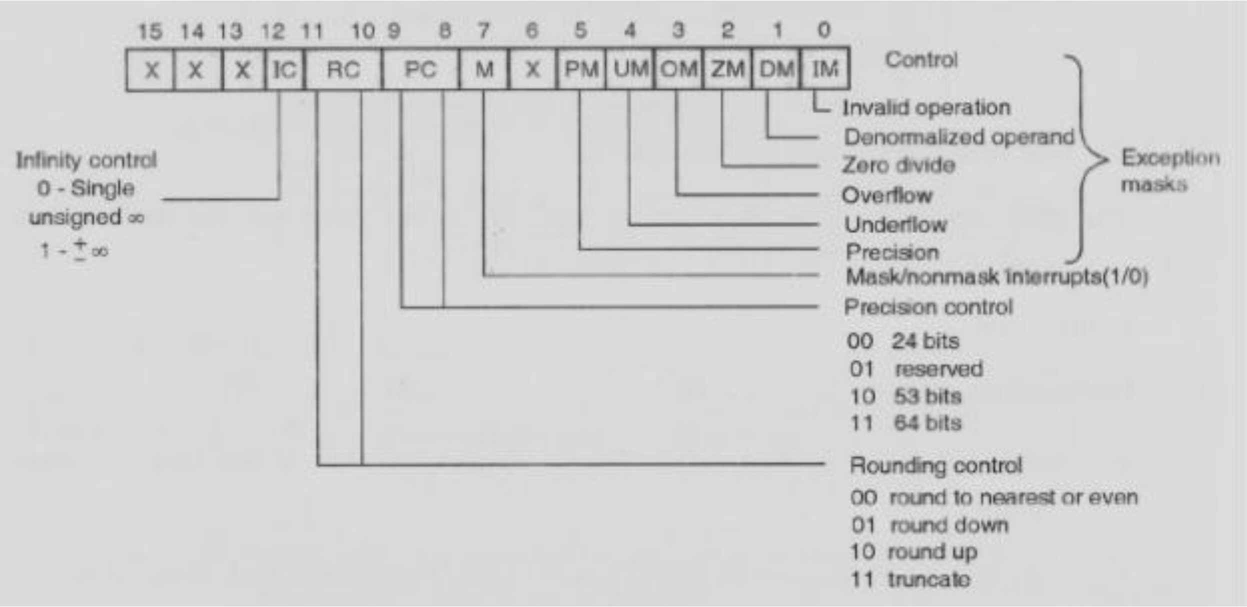
Any instruction having an ESC prefix is decoded and executed by the 8087, other instructions are ignored by it.

This queue is synchronized by monitoring the QS0 and QS1 lines.

Control word:

Control words are sent to 8087 by writing them to a memory location and having 8087 execute an instruction which reads in the control word from the memory.

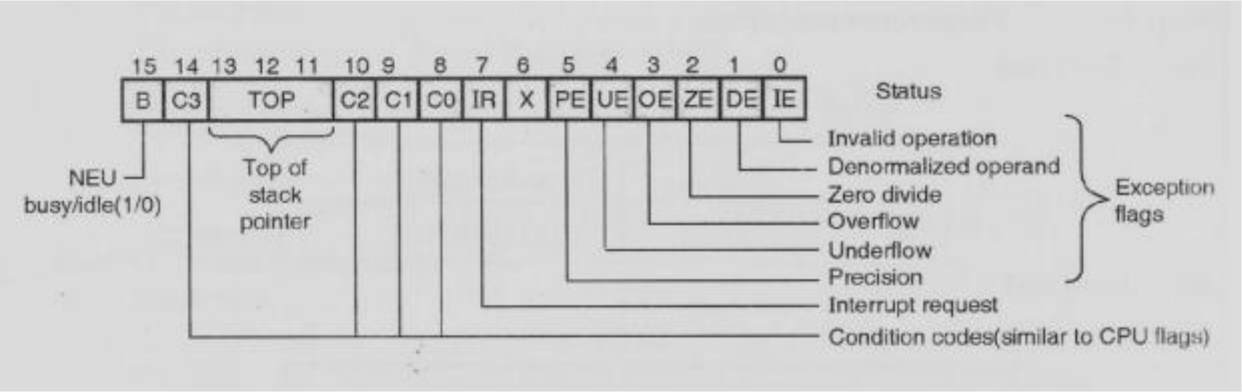
The control word is shown in the figure below:



Status word:

To read the status word from an 8087, you have it execute an instruction which writes the status word to memory where you can read or check it with an 8086 instruction.

The status word is shown in the figure below:



Instruction pointer:

It mainly contains address of the ESC instruction. Its 32 bit complete composition is

- 20 bits address of ESC instruction.
- 11 bits out of the 16 bit instruction code (the other 5 bits being 11011 for ESC).
- 1 bit is always 0.

Data pointer:

It mainly contains address of data. Its 32 bit complete composition is

- 20 bits memory address of data.
- 12 bits are always 0.

Numeric Extension Unit

The **numeric extension unit** handles all the numeric processor instructions like arithmetic, logical, transcendental, and data transfer instructions. It has 8 register stack, which holds the operands for instructions and their results.

Register stack:

8087 has eight 80 bit numeric data registers available to the programmer.

These registers operate in LIFO (Last in First Out) manner hence they are called as the register stack of 8087.

Each register is 80 bits as the data is stored internally in the temporary-real format.

The current top of the stack is called as ST (0) or simply ST and the subsequent registers are called ST (1), ST (2) etc.

8087 has a 3 bit stack pointer to hold the number of the register that is the current ST.

On initialization the stack pointer contains 000 i.e. Register (0) is the ST.

It is a circular stack i.e. after pushing 8 elements if we PUSH the 9th element it will overwrite on the 1st element itself.

After the first PUSH, SP is decremented by 1. Therefore, SP = 111 and hence the data is stored at register 7. So register 7 becomes ST (0).

During POP, data is read from top of stack and SP is incremented by 1. Therefore, SP = 000 so register 0 becomes ST (0).

Tag word:

8087 contains 2 tag bits for each stack register.

These tag bits indicate the type of number stored in the respective register.

As there are 8 registers, there is 2 x 8 = 16 tag bits called collectively as the Tag word.

When 8087 is initialized, tag word = FFFF (all 1s) as all registers are empty.

Tag bits	**Information**
00	Normal number (non-zero finite)
01	Zero
10	Special number (NaN, Infinity or De-Normal)
11	Empty

Direct Memory Access (DMA) — It is a method of allowing data to be moved from one location to another in a computer with intervention from the CPU or µP.

- It is also a fast way of transferring data with in computer.
- The DMA I/O technique provides direct access to memory while the µP is temporarily disabled.
- ~~The~~ In DMA controller ~~temp~~ technique, controller borrows the address bus, data bus and control bus from the µP and transfers the data directly from external devices.

→ **Basic DMA Operation**

3 ~~It's~~ Two control signals are used to request and acknowledge a DMA transfer µP based system.

- The HOLD signal as an i/p to processor is used to request a DMA action.
- The HLDA signal as an o/p that acknowledges the DMA action.
- When the processor recognizes the HOLD, it stop its ~~current~~ execution
- HOLD i/p has higher priority than INTR or NMI.

Basic DMA Definitions-

DMA normally occurs b/w an I/O device and memory without the use of the µP.

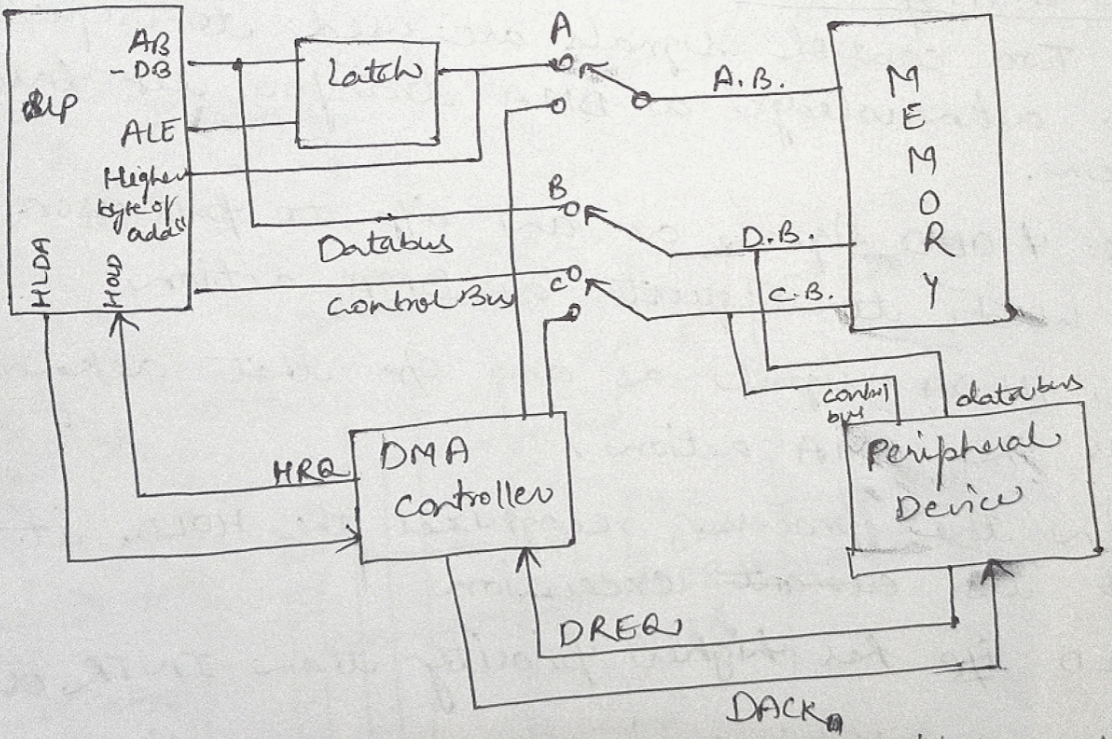
- A **DMA Read** transfers data from the memory to the I/O device.

- A DMA write transfers data from an I/O device to memory.

Data Transfer

DMA Technique:- In this technique external device is used to control data transfer. External device ~~is used~~ generates addresses and control signals required to control data transmission and allows peripheral to access directly to the memory. so this technique is called DMA and external device which controls data transfer is referred to as DMA controller.

Fig. shows how the DMA controller operates in µP system.



When the system is turned ON, switches are in position A so buses are connected from the µP to system memory and peripherals, µP executes the program until it needs to read a block

Features of 8257

- It is a 4-channel DMA.
- Each channel have 16 bit address and 14 bit counter.
- It executes 3 DMA cycles

(i) DMA Read
(ii) DMA Write
(iii) DMA Verify

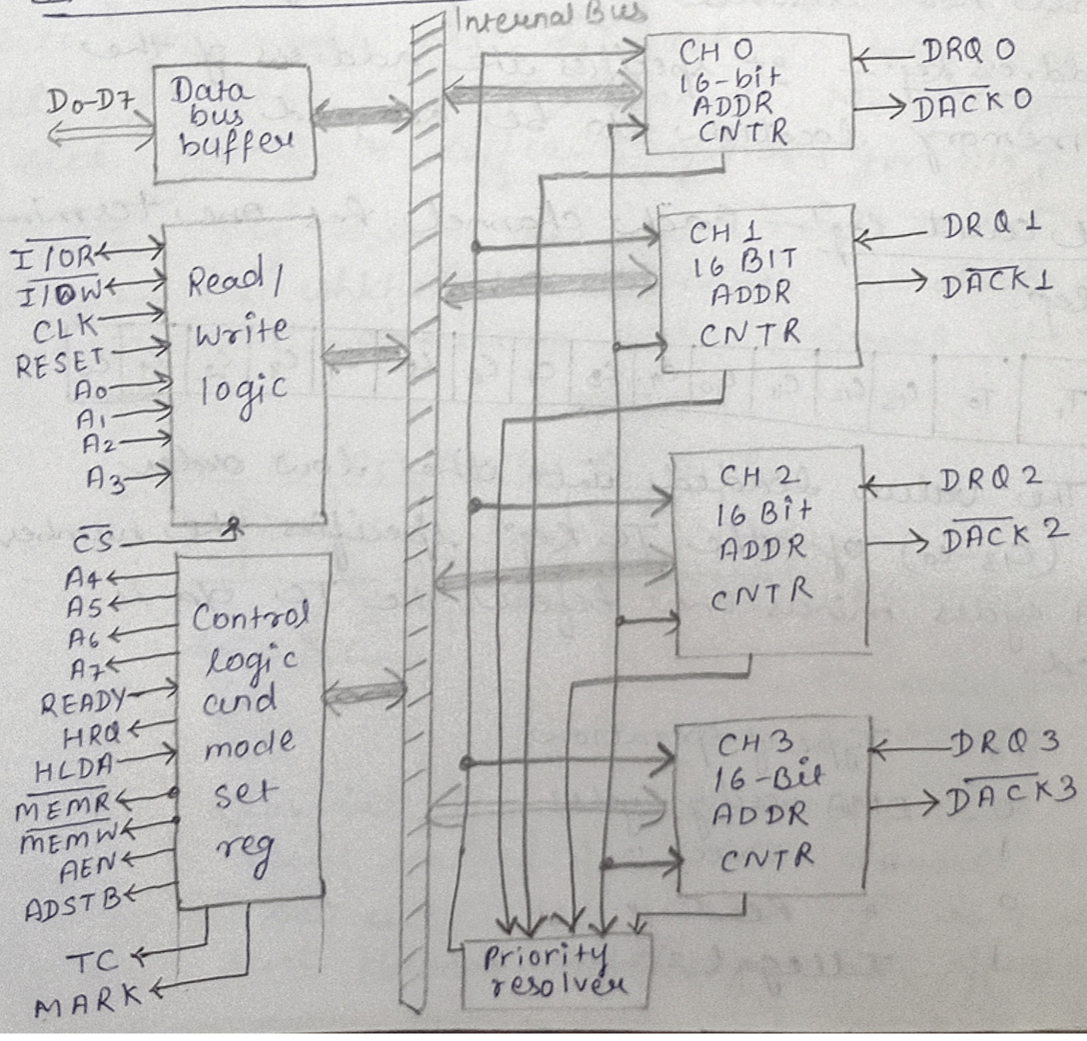
- It is operate in two modes:

(i) Master Mode
(ii) Slave Mode

In the Master mode it is unidirectional (add^n is moving)

In the slave mode it is bidirectional (data is moving)

Internal Architecture :-



The internal architecture of 8257 is shown in fig. DMA has 8 bit internal at data buffer, a read/write unit, a control unit, a priority resolving unit along with a set of registers.

Register organization of 8257 - 8257 performs DMA operation over 4, independent channels. Each of four channels of 8257 has a pair of two, 16 bit registers.

(i) DMA address reg^{r} (ii) Terminal count Reg^{r}.

Also there are 2 common registers for all channels namely (i) mode set reg^{r} (ii) status reg^{r}. Thus there are total 10 reg^{rs}. The CPU selects one of these reg^{rs} using address lines A0-A3. We will now describe each reg^{r} as follows:

DMA address Reg^{r} - It specifies the address of the first memory location to be accessed.

Terminal count Reg^{r} - Each channel has one terminal count Reg^{r}.

T_1	T_0	C_{13}	C_{12}	C_{11}	C_{10}	C_9	C_8	C_7	C_6	C_5	C_4	C_3	C_2	C_1	C_0

The value loaded into the low order 14 bits (C_{13}-C_0) of the TC Reg^{r} specifies the number of DMA cycles minus one before the TC o/p is activated.

T_1	T_0	Type of operation
0	0	DMA verify cycle
0	1	" Write "
1	0	" Read "
1	1	Illegal

The most significant 2 bits of TC Regr specifies the type of DMA operation to be performed. It is necessary to load count DMA cycles and operational code for valid DMA cycle in TC regr before channel is enabled.

Mode set Regr –

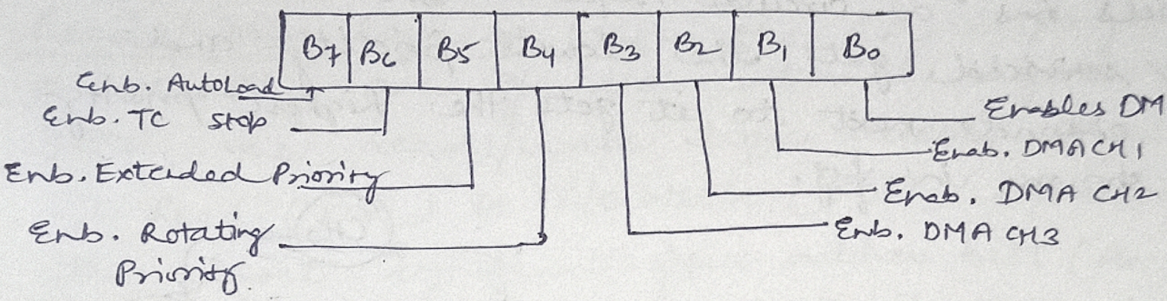
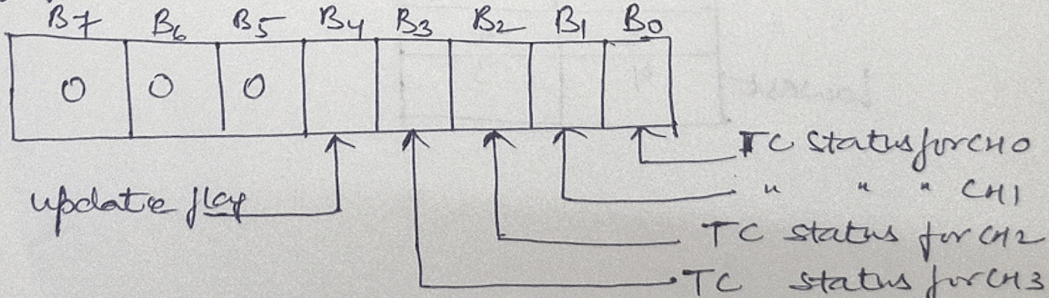


Fig. gives the format of MSR. Least significant 4 bits of mode set regr, when set, enable each of the 4 DMA channels. Most significant 4 bits allow 4 different options for 8257.

Status Regr – Fig. shows the status regr format. ~~As~~ It indicates which channels have reached a terminal count condition and includes the update flag.



The TC status bit, if one indicates terminal count has been reached for that channel. TC bit remains set until the status regr is read or 8257 is reset. However update flag is not affected.

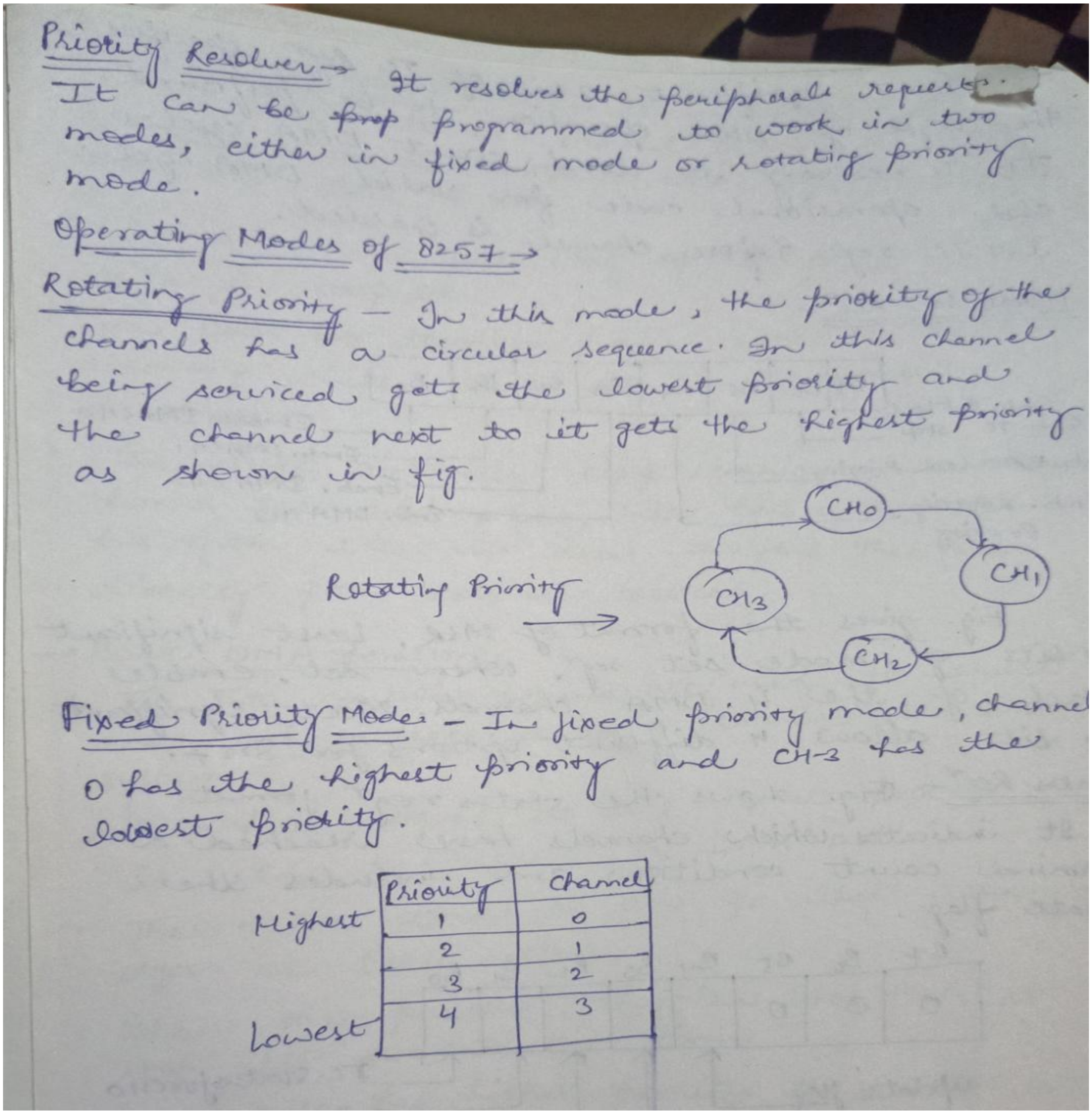
Data Bus Buffer - It is a bidirectional 8 bit buffer which interfaces the 8257 to the system data bus. In slave mode it is used to transfer data between μP and internal reg^n of 8257.

In master mode, it is used to send higher byte address (A_8-A_{15}) on the databus.

Read/Write logic → When the CPU is programming or reading one of the internal registers of 8257, the read/write logic accepts the I/O Read ($\overline{IOR}$) or I/O Write ($\overline{IOW}$) signal, decod the least significant four address bits (A_0-A_3) and either writes the contents of the data bus in to the adressed register or places the contents of the addressed register onto the data bus.

- **Control Logic**
- This block controls the sequence of operations during all DMA cycles by generating the appropriate control signals and the 16-bit address that specifies the memory location to be accessed.
- **A_4 - A_7**
- These are the higher nibble of the lower byte address generated by DMA in the master mode.
- **READY**
- It is an active-high asynchronous input signal, which makes DMA ready by inserting wait states.
- **HRQ**
- This signal is used to receive the hold request signal from the output device. In the slave mode, it is connected with a DRQ input line 8257. In Master mode, it is connected with HOLD input of the CPU.
- **HLDA**
- It is the hold acknowledgement signal which indicates the DMA controller that the bus has been granted to the requesting peripheral by the CPU when it is set to 1.
- **MEMR**
- It is the low memory read signal, which is used to read the data from the addressed memory locations during DMA read cycles.
- **MEMW**
- It is the active-low three state signal which is used to write the data to the addressed memory location during DMA write operation.
- **ADSTB**
- This signal is used to convert the higher byte of the memory address generated by the DMA controller into the latches.
- **AEN**
- It may be further used to isolate the 8257 data bus from the System Data Bus to facilitate the transfer of the most significant DMA address bits

- **TC**
- This output notifies the currently selected peripheral that the present DMA cycle should be the last cycle for this data block.
- **MARK**
- The mark will be activated after each 128 cycles or integral multiples of it from the beginning. It indicates the current DMA cycle is the 128th cycle since the previous MARK output to the selected peripheral device.



Priority Resolver → It resolves the peripherals requests. It can be ~~frop~~ programmed to work in two modes, either in fixed mode or rotating priority mode.

Operating Modes of 8257 →

Rotating Priority – In this mode, the priority of the channels has a circular sequence. In this channel being serviced gets the lowest priority and the channel next to it gets the highest priority as shown in fig.

Rotating Priority →

Fixed Priority Mode – In fixed priority mode, channel 0 has the highest priority and CH-3 has the lowest priority.

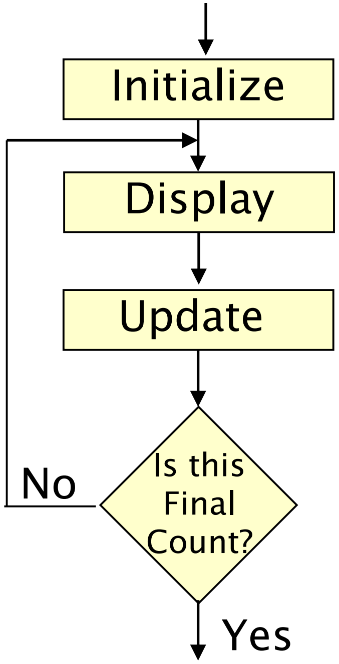
	Priority	Channel
Highest	1	0
	2	1
	3	2
Lowest	4	3

What are Counters & Time Delays ?

- ✓ COUNTERS ARE USED TO KEEP TRACK OF EVENTS.

- ✓ TIME DELAYS ARE IMPORTANT IN SETTING UP REASONABLY ACCURATE TIMING BETWEEN TWO EVENTS.

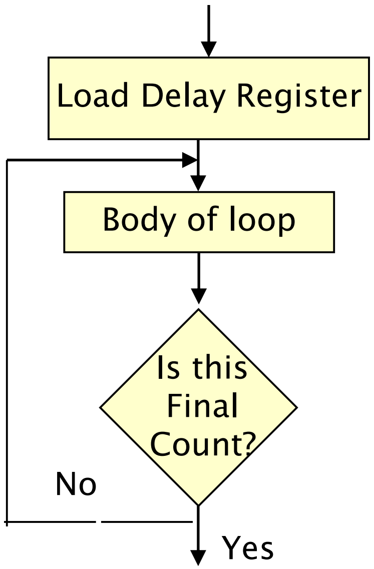
COUNTERS



Flowchart of a Counter

A Counter is designed simply by loading an appropriate number into one of the registers and using the INR (Increment by one) or the DCR (Decrement by one) instructions. A loop is established to update the count, and each count is checked to determine whether it has reached the final number; if not, the loop is repeated.

TIME DELAYS



Flowchart of a Time Delay

The procedure used to design a specific delay is similar to that used to set up a counter. A register is loaded with a number, depending on the time delay required, and then the register is decremented until it reaches zero by setting up a loop with a conditional jump instruction. The loop causes the delay, depending upon the clock period of the system.

Calculating Time Delays

- ✓ Each instruction passes through different combinations of Opcode Fetch, Memory Read, and Memory Write cycles.
- ✓ Knowing the combinations of cycles, one can calculate how long such an instruction would require to complete.
  - ✓ Number of Bytes
  - ✓ Number of Machine Cycles
  - ✓ Number of T-State.

Calculating Time Delays

- ✓ Knowing how many T-States an instruction requires, and keeping in mind that a T-State is one clock cycle long, we can calculate the time delay using the following formula:

Time Delay = No. of T-States * Clock Period

- ✓ For example,

“MVI” instruction uses 7 T-States. Therefore, if the Microprocessor is running at 2 MHz, the instruction would require 3.5 μS to complete.

We can design Time Delay using following three Techniques:

1. Using One Register.
2. Using a Register Pair.
3. Using a Loop with in a Loop

Using One Register

- A count is loaded in a register, and We can use a loop to produce a certain amount of time delay in a program.
- The following is an example of a delay using One Register:

```
        MVI C, FFH                  7 T-States
LOOP    DCR C                       4 T-States
        JNZ LOOP                   10 T-States
```

- The first instruction initializes the loop counter and is executed only once requiring only 7 T-States.
- The following two instructions form a loop that requires 14 T-States to execute and is repeated 255 times until C becomes 0.

Using One Register

- ✓ We need to keep in mind though that in the last iteration of the loop, the JNZ instruction will fail and require only 7 T-States rather than the 10.
- ✓ Therefore, we must deduct 3 T-States from the total delay to get an accurate delay calculation.
- ✓ To calculate the delay, we use the following formula:

$$T_{delay} = T_O + T_L$$

T_{delay} = total delay
T_O = delay outside the loop
T_L = delay of the loop

- ✓ T_O is the sum of all delays outside the loop.

- ✓ T_L is calculated using the formula

$$T_L = T * \text{Loop T-States} * N \text{ (no. of iterations)}$$

Using One Register

- ✓ Using these formulas, we can calculate the time delay for the previous example:
- ✓ T_O = 7 T-States

  (Delay of the MVI instruction)
- ✓ T_L = (14 X 255) - 3 = 3567 T-States

  (14 T-States for the 2 instructions repeated 255 times ($FF_{16} = 255_{10}$) reduced by the 3 T-States for the final JNZ.)
- ✓ $T_{delay} = [(T_O + T_L)/f]$

  = (7 + 3567)/2MHz

  = (3574) X 0.5 μSec

  = 1.787 mSec

  (Assuming **f** = 2 MHz)

Using a Register Pair

- ✓ Using a single register, one can repeat a loop for a maximum count of 255 times.

- ✓ It is possible to increase this count by using a register pair for the loop counter instead of the single register.

  - ✓ A minor problem arises in how to test for the final count since DCX and INX do not modify the flags.
  - ✓ However, if the loop is looking for when the count becomes zero, we can use a small trick by ORing the two registers in the pair and then checking the zero flag.

Using a Register Pair

- The following is an example of a delay loop set up with a register pair as the loop counter.

```
        LXI B, 1000H        10 T-States
LOOP    DCX B                6 T-States
        MOV A, C             4 T-States
        ORA B                4 T-States
        JNZ LOOP            10 T-States
```

Using a Register Pair

- ✓ Using the same formula from before, we can calculate:

- ✓ T_O = 10 T-States

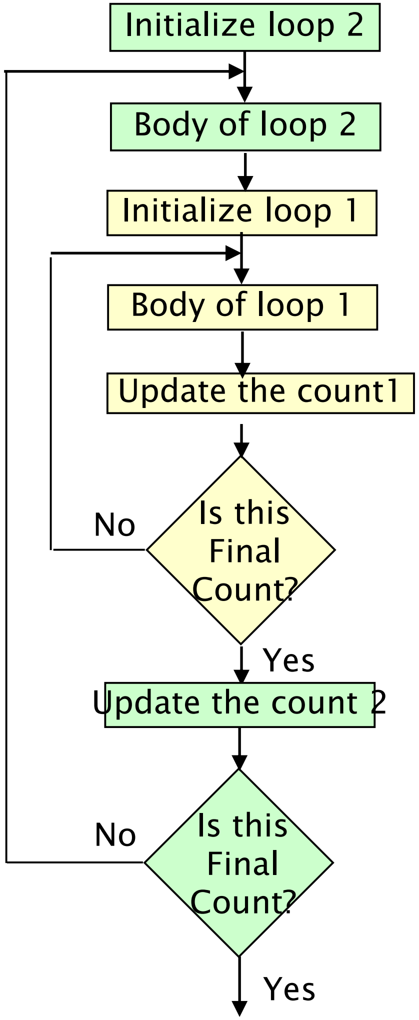
  (The delay for the LXI instruction)

- ✓ T_L = (24 X 4096) - 3 = 98301 T- States

  (24 T-States for the 4 instructions in the loop repeated 4096 times ($1000_{16} = 4096_{10}$) reduced by the 3 T-States for the JNZ in the last iteration.)

- ✓ T_{Delay} = (10 + 98301) X 0.5 mSec = 49.155 mSec

Using a Loop with in a Loop



- ✓ Nested loops can be easily setup in Assembly language by using two registers for the two loop counters and updating the right register in the right loop.

Flowchart for time delay with two loops

Using a Loop with in a Loop

- Instead (or in conjunction with) Register Pairs, a nested loop structure can be used to increase the total delay produced.

```
        MVI B, 10H            7 T-States
LOOP2   MVI C, FFH            7 T-States
LOOP1   DCR C                 4 T-States
        JNZ LOOP1            10 T-States
        DCR B                 4 T-States
        JNZ LOOP2            10 T-States
```

Using a Loop with in a Loop

- ✓ The calculation remains the same except that the formula must be applied recursively to each loop.

  Start with the inner loop, then plug that delay in the calculation of the outer loop.

- ✓ Delay of inner loop,

  T_{O1} = 7 T-States

  (MVI C, FFH instruction)

  T_{L1} = (255 X 14) - 3 = 3567 T-States

  (14 T-States for the DCR C and JNZ instructions repeated 255 times ($FF_{16} = 255_{10}$) minus 3 for the final JNZ.)

  T_{LOOP1} = 7 + 3567 = 3574 T-States

Using a Loop with in a Loop

- ✓ Delay of outer loop

  T_{O2} = 7 T-States

  (MVI B, 10H instruction)

  T_{L1} = (16 X (14 + 3574)) - 3 = 57405 T-States

  (14 T-States for the DCR B and JNZ instructions and 3574 T-States for loop1 repeated 16 times ($10_{16} = 16_{10}$) minus 3 for the final JNZ.)

  T_{Delay} = 7 + 57405 = 57412 T-States

- ✓ Total Delay

  T_{Delay} = 57412 X 0.5 µSec = 28.706 mSec

Increasing the Time Delay

- ✓The Delay can be further increased by using register pairs for each of the loop counters in the nested loops setup.

- ✓It can also be increased by adding dummy instructions (like NOP) in the body of the loop.

The 8279 Programmable Keyboard/ Display Interface

8279 is a hardware approach to interfacing a matrix keyboard and multiplexed display, the software approach have many disadvantage like microprocessor is occupied for considerable amount of time in checking the keyboard and refreshing the display. The 8279 relieves the processor from this task. The only disadvantage of using the 8279 is the cost.

8279 programmable keyboard/display controller is designed by Intel that interfaces a keyboard with the CPU. The keyboard first scans the keyboard and identifies if any key has been pressed. It then sends their relative response of the pressed key to the CPU and vice-a-versa.

Two Ways the Keyboard is Interfaced with the CPU

The Keyboard can be interfaced either in the interrupt or the polled mode.

In the **Interrupt mode**, the processor is requested service only if any key is pressed, otherwise the CPU will continue with its main task.

In the **Polled mode**, the CPU periodically reads an internal flag of 8279 to check whether any key is pressed or not with key pressure.

Working of 8279:

The keyboard consists of maximum 64 keys, which are interfaced with the CPU by using the key-codes. These key-codes are de-bounced and stored in an 8-byte FIFO RAM, which can be accessed by the CPU. If more than 8 characters are entered in the FIFO, then it means more than eight keys are pressed at a time. This is when the overrun status is set.

If a FIFO contains a valid key entry, then the CPU is interrupted in an interrupt mode else the CPU checks the status in polling to read the entry. Once the CPU reads a key entry, then FIFO is updated, and the key entry is pushed out of the FIFO to generate space for new entries.

Architecture and Description

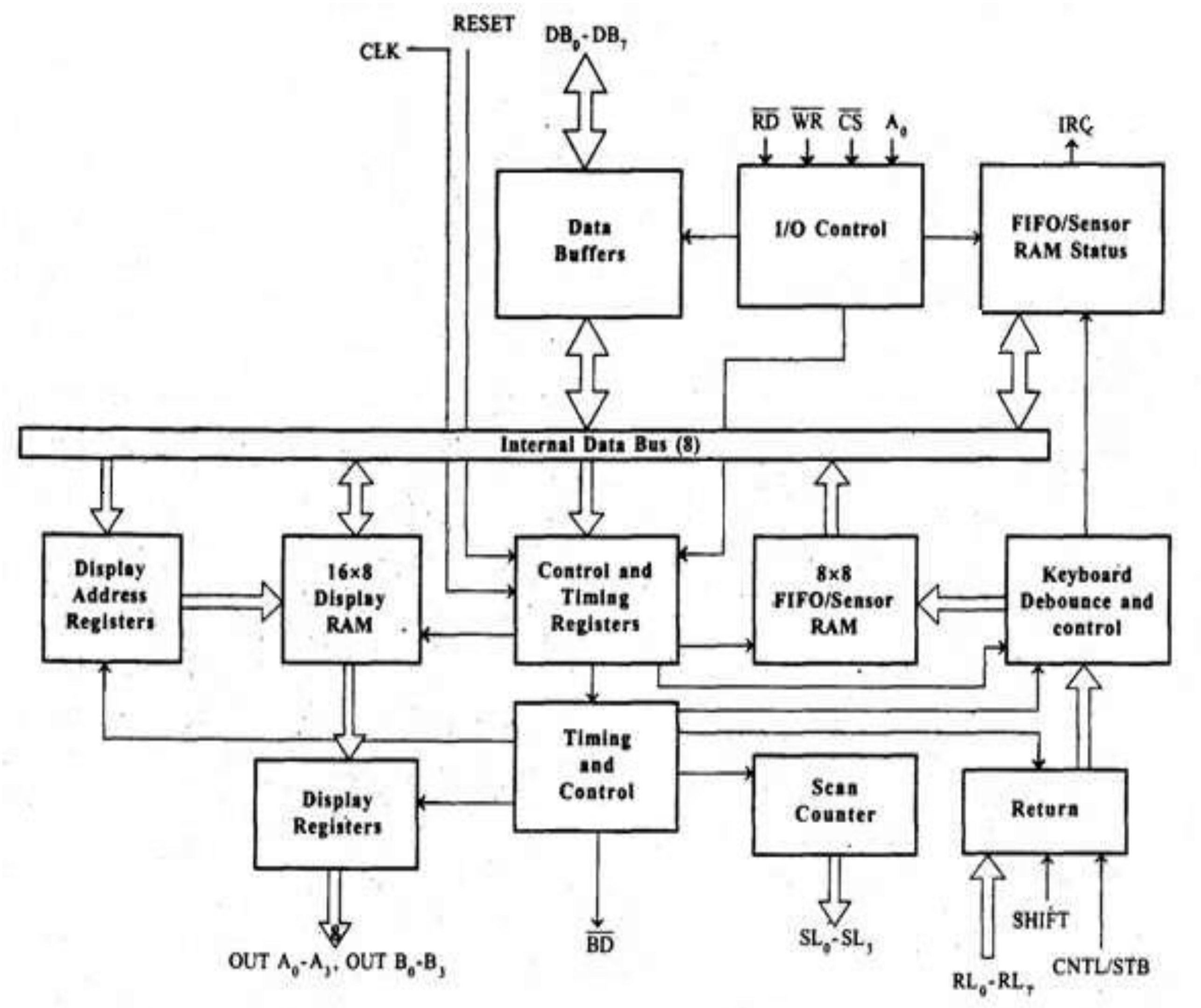
It consists of four main sections:

1. CPU interface and control section

2. Scan section

3. Keyboard section

4. Display section



1. CPU interface and control section:

I/O Control and Data Buffer

This unit controls the flow of data through the microprocessor. It is enabled only when D is low. Its data buffer interfaces the external bus of the system with the internal bus of the microprocessor. The pins A0, RD, and WR are used for command, status or data read/write operations.

Control and Timing Register and Timing Control

This unit contains registers to store the keyboard, display modes, and other operations as programmed by the CPU. The timing and control unit handles the timings for the operation of the circuit.

2. Scan section

Scan Counter

It has two modes i.e. **Encoded mode** and Decoded mode. In the encoded mode, the counter provides the binary count that is to be externally decoded to provide the scan lines for the keyboard and display.

In the **decoded scan mode**, the counter internally decodes the least significant 2 bits and provides a decoded 1 out of 4 scan on SL_0-SL_3.

3. Keyboard section

Return Buffers, Keyboard Debounce, and Control

This unit first scans the key closure row-wise, if found then the keyboard debounce unit debounces the key entry. In case, the same key is detected, then the code of that key is directly transferred to the sensor RAM along with SHIFT & CONTROL key status.

FIFO/Sensor RAM and Status Logic

This unit acts as 8-byte first-in-first-out (FIFO) RAM where the key code of every pressed key is entered into the RAM as per their sequence. The status logic generates an interrupt request after each FIFO read operation till the FIFO gets empty.

In the scanned sensor matrix mode, this unit acts as sensor RAM where its each row is loaded with the status of their corresponding row of sensors into the matrix. When the sensor changes its state, the IRQ line changes to high and interrupts the CPU.

4. Display section

Display Address Registers and Display RAM

This unit consists of display address registers which holds the addresses of the word currently read/written by the CPU to/from the display RAM.

Interfacing Analog to Digital Data Converters

- In most of the cases, the PIO 8255 is used for interfacing the analog to digital converters with microprocessor.
- We have already studied 8255 interfacing with 8086 as an I/O port, in previous section. This section we will only emphasize the interfacing techniques of analog to digital converters with 8255.
- The analog to digital converters is treaded as an input device by the microprocessor, that sends an initialising signal to the ADC to start the analogy to digital data conversation process. The start of conversation signal is a pulse of a specific duration.

Interfacing Analog to Digital Data Converters (cont..)

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

Interfacing Analog to Digital Data Converters (cont..)

- The time taken by the ADC from the active edge of SOC pulse till the active edge of EOC signal is called as the conversion delay of the ADC.
- It may range any where from a few microseconds in case of fast ADC to even a few hundred milliseconds in case of slow ADCs.
- The available ADC in the market use different conversion techniques for conversion of analog signal to digitals. Successive approximation techniques and dual slope integration techniques are the most popular techniques used in the integrated ADC chip.

Interfacing Analog to Digital Data Converters (cont..)

- General algorithm for ADC interfacing contains the following steps:

1. Ensure the stability of analog input, applied to the ADC.
2. Issue start of conversion pulse to ADC
3. Read end of conversion signal to mark the end of conversion processes.
4. Read digital data output of the ADC as equivalent digital output.

Interfacing Analog to Digital Data Converters (cont..)

- Analog input voltage must be constant at the input of the ADC right from the start of conversion till the end of the conversion to get correct results. This may be ensured by a sample and hold circuit which samples the analog signal and holds it constant for a specific time duration. The microprocessor may issue a hold signal to the sample and hold circuit.
- If the applied input changes before the complete conversion process is over, the digital equivalent of the analog input calculated by the ADC may not be correct.

Interfacing Analog to Digital Data Converters (cont..)

ADC 0808/0809 :

- The analog to digital converter chips 0808 and 0809 are 8-bit CMOS, successive approximation converters. This technique is one of the fast techniques for analog to digital conversion. The conversion delay is 100µs at a clock frequency of 640 KHz, which is quite low as compared to other converters. These converters do not need any external zero or full scale adjustments as they are already taken care of by internal circuits. These converters internally have a 3:8 analog multiplexer so that at a time eight different analog conversion by using address lines -

Interfacing Analog to Digital Data Converters (cont..)

ADD A, ADD B, ADD C. Using these address inputs, multichannel data acquisition system can be designed using a single ADC. The CPU may drive these lines using output port lines in case of multichannel applications. In case of single input applications, these may be hardwired to select the proper input.

- There are unipolar analog to digital converters, i.e. they are able to convert only positive analog input voltage to their digital equivalent. These chips do no contain any internal sample and hold circuit.

Analog I/P selected	Address lines		
	C	B	A
I / P $_0$	0	0	0
I / P $_1$	0	0	1
I / P $_2$	0	1	0
I / P $_3$	0	1	1
I / P $_4$	1	0	0
I / P $_5$	1	0	1
I / P $_6$	1	1	0
I / P $_7$	1	1	1

Fig

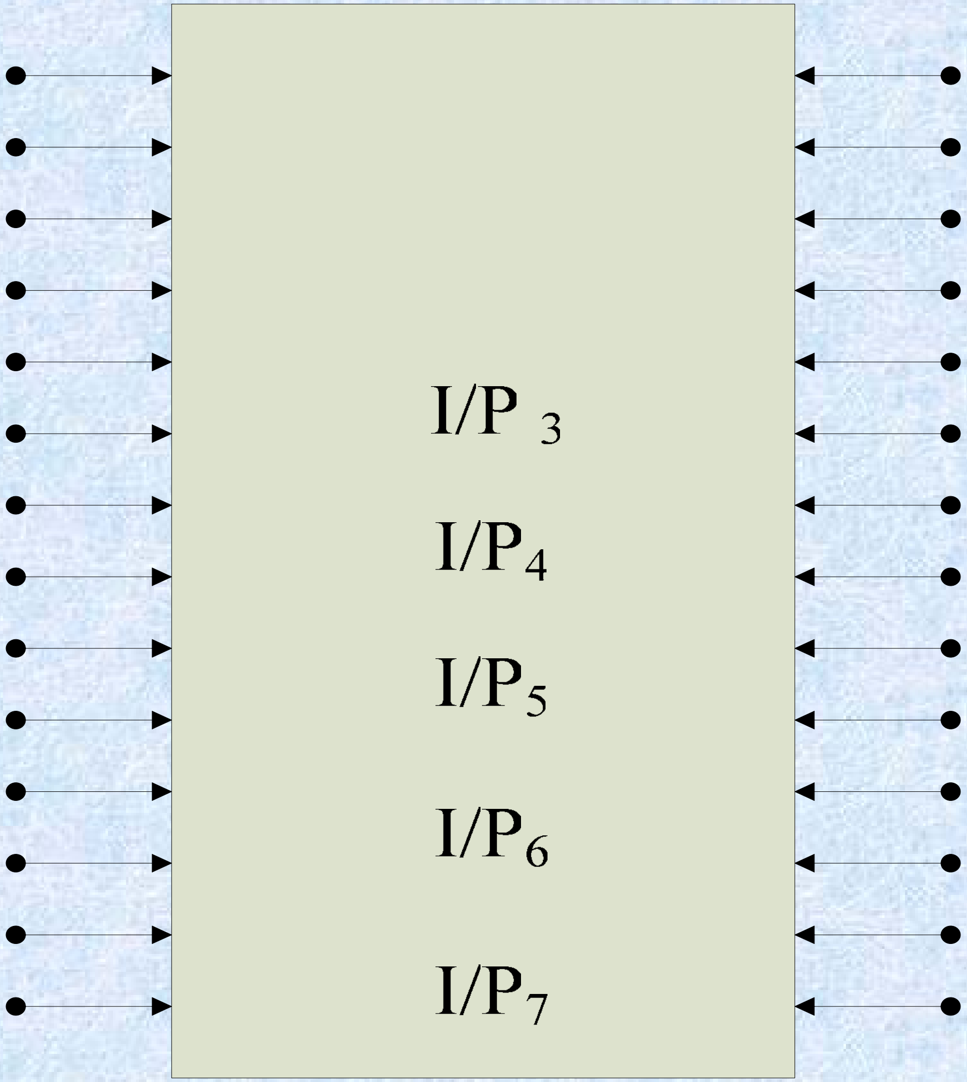
Interfacing Analog to Digital Data Converters (cont..)

- If one needs a sample and hold circuit for the conversion of fast signal into equivalent digital quantities, it has to be externally connected at each of the analog inputs.

• Vcc	Supply pins +5V
• GND	GND
• Vref +	Reference voltage positive +5 Volts maximum.
• Vref –	Reference voltage negative 0Volts minimum.

Interfacing Analog to Digital Data Converters (cont..)

• I/P_0 –I/P_7	Analog inputs
• ADD A,B,C	Address lines for selecting analog inputs.
• O_7 – O_0	Digital 8-bit output with O_7 MSB and O_0 LSB
• SOC	Start of conversion signal pin
• EOC	End of conversion signal pin
• OE	Output latch enable pin, if high enables output
• CLK	Clock input for ADC



I/P $_3$

I/P$_4$

I/P$_5$

I/P$_6$

I/P$_7$

SOC

EOC

1

2

3

4

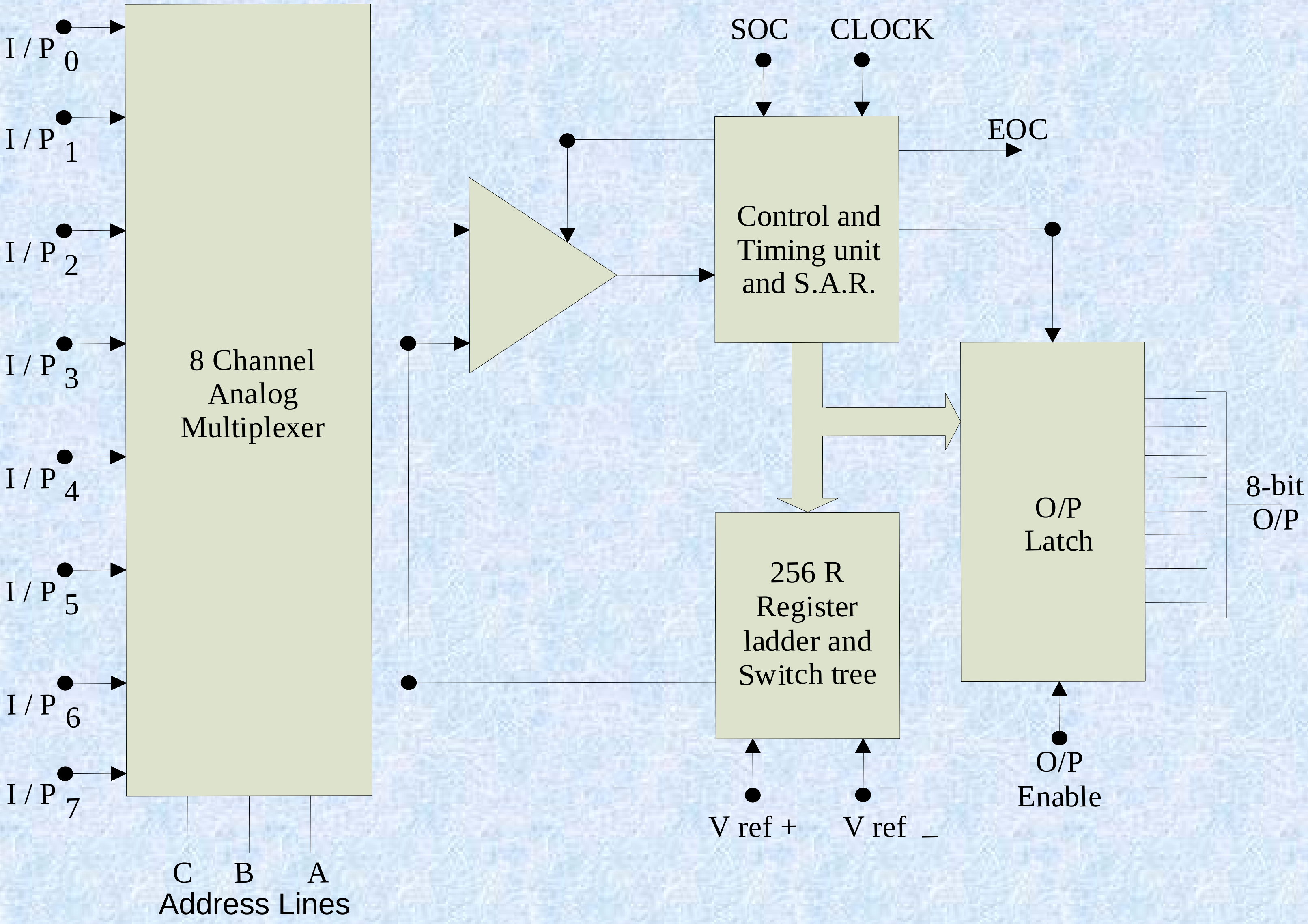
5

6

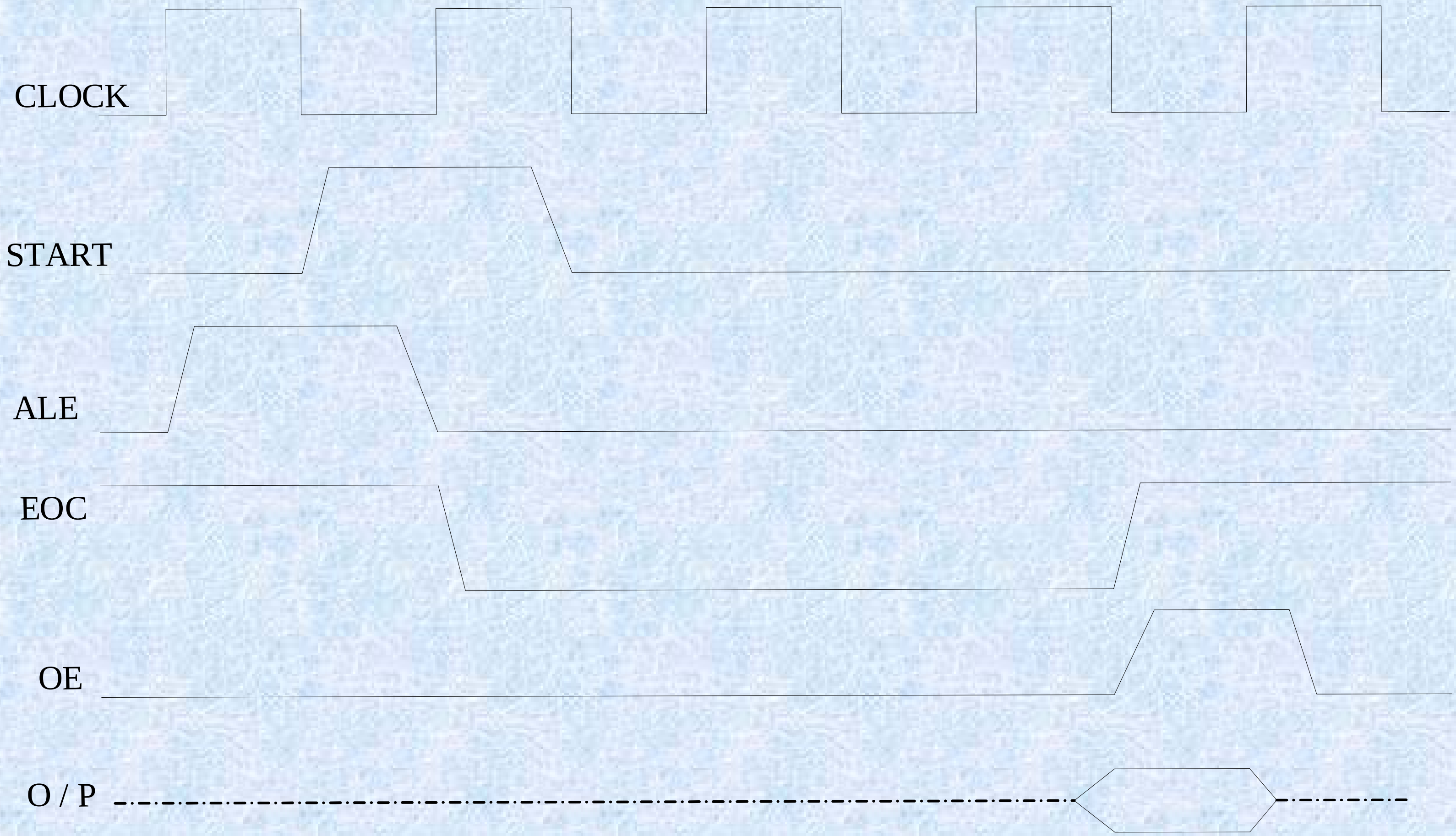
7

ADC

ADC



Block Diagram of ADC 0808 / 0809



Timing Diagram of ADC 0808

Interfacing Analog to Digital Data Converters (cont..)

- ***Example:*** Interfacing ADC 0808 with 8086 using 8255 ports. Use port A of 8255 for transferring digital data output of ADC to the CPU and port C for control signals. Assume that an analog input is present at I/P_2 of the ADC and a clock input of suitable frequency is available for ADC.
- **Solution**: The analog input I/P_2 is used and therefore address pins A,B,C should be 0,1,0 respectively to select I/P_2. The OE and ALE pins are already kept at +5V to select the ADC and enable the outputs. Port C upper acts as the input port to receive the EOC signal while port C lower acts as the output port to send SOC to the ADC.

Interfacing Analog to Digital Data Converters (cont..)

- Port A acts as a 8-bit input data port to receive the digital data output from the ADC. The 8255 control word is written as follows:

  D_7 D_6 D_5 D_4 D_3 D_2 D_1 D_0

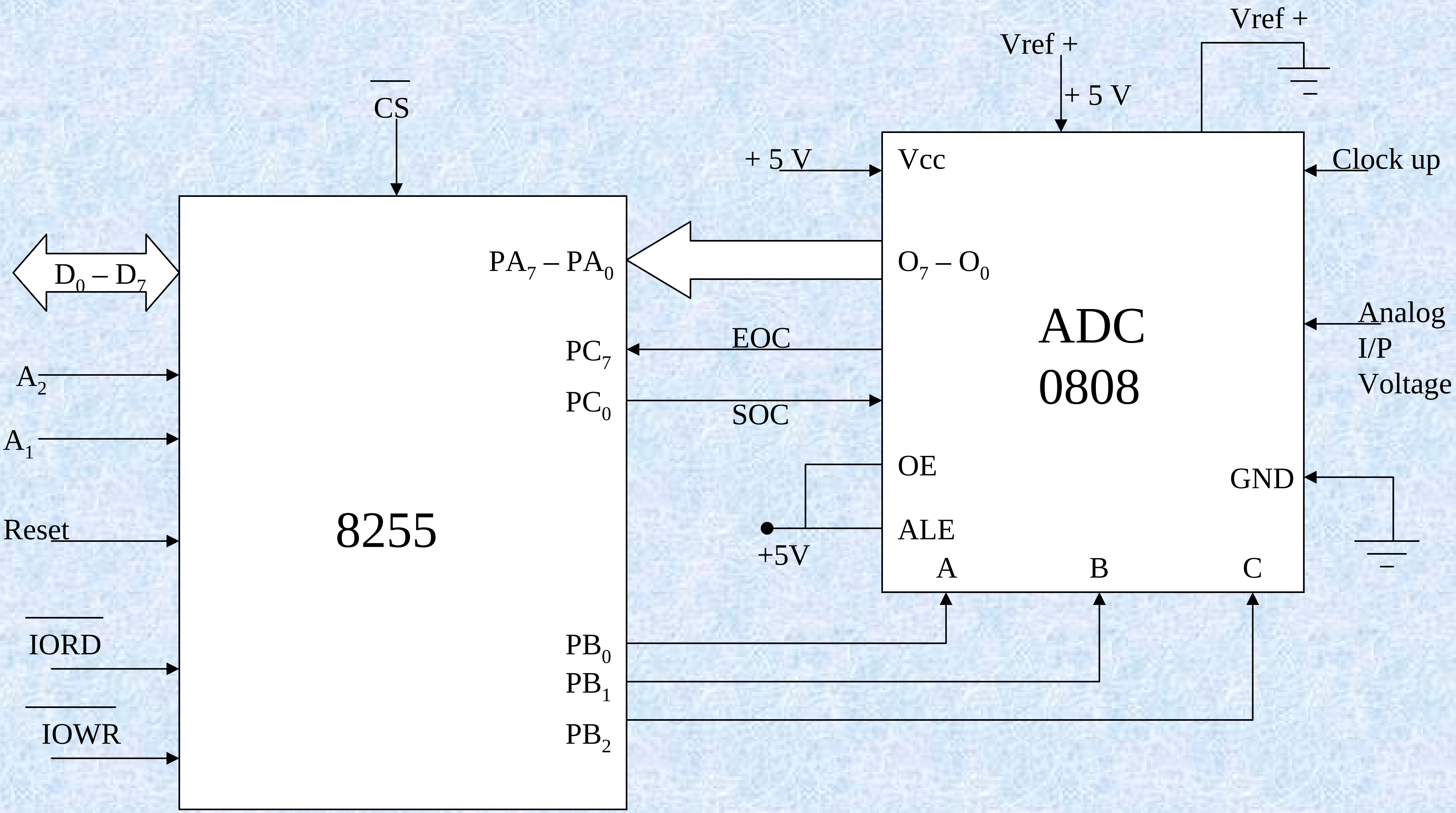
  1 0 0 1 1 0 0 0

- The required ALP is as follows:

```
MOV      AL, 98h       ;initialise 8255 as
OUT      CWR, AL       ;discussed above.
MOV      AL, 02h       ;Select I/P2 as analog
OUT      Port B, AL    ;input.
```

Interfacing Analog to Digital Data Converters (cont..)

```
        MOV   AL, 00h        ;Give start of conversion
        OUT   Port C, AL     ; pulse to the ADC
        MOV   AL, 01h
        OUT   Port C, AL
        MOV   AL, 00h
        OUT   Port C, AL
WAIT: IN      AL, Port C     ;Check for EOC by
        RCR                  ; reading port C upper and
        JNC   WAIT           ;rotating through carry.
        IN    AL, Port A     ;If EOC, read digital equivalent
                             ;in AL
        HLT                  ;Stop.
```

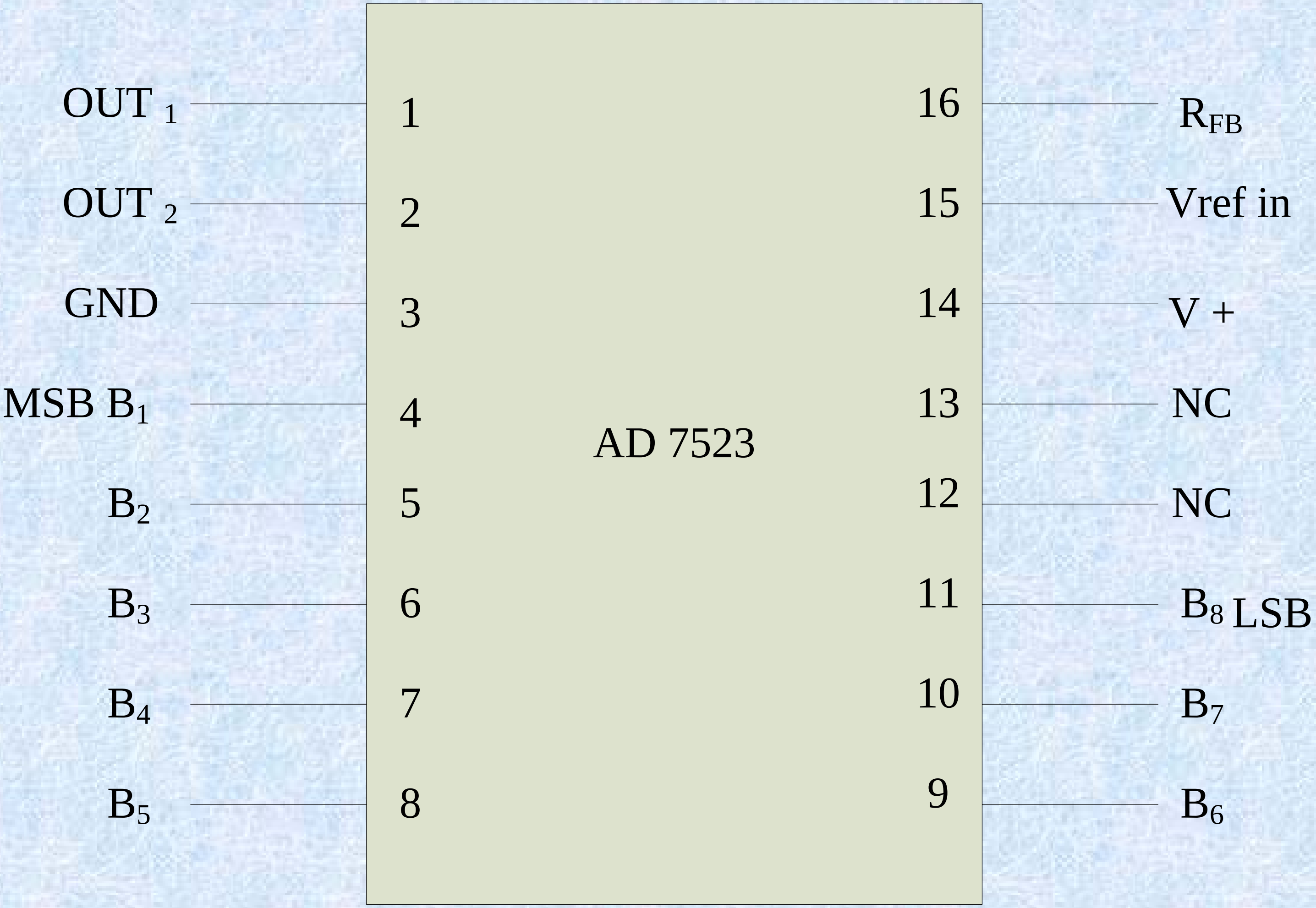


Interfacing 0808 with 8086

Interfacing Digital To Analog Converters (cont..)

INTERFACING DIGITAL TO ANALOG CONVERTERS: The digital to analog converters convert binary number into their equivalent voltages. The DAC find applications in areas like digitally controlled gains, motors speed controls, programmable gain amplifiers etc.

AD 7523 8-bit Multiplying DAC : This is a 16 pin DIP, multiplying digital to analog converter, containing R-2R ladder for D-A conversion along with single pole double thrown NMOS switches to connect the digital inputs to the ladder.



Pin Diagram of AD 7523

Interfacing Analog to Digital Data Converters (cont..)

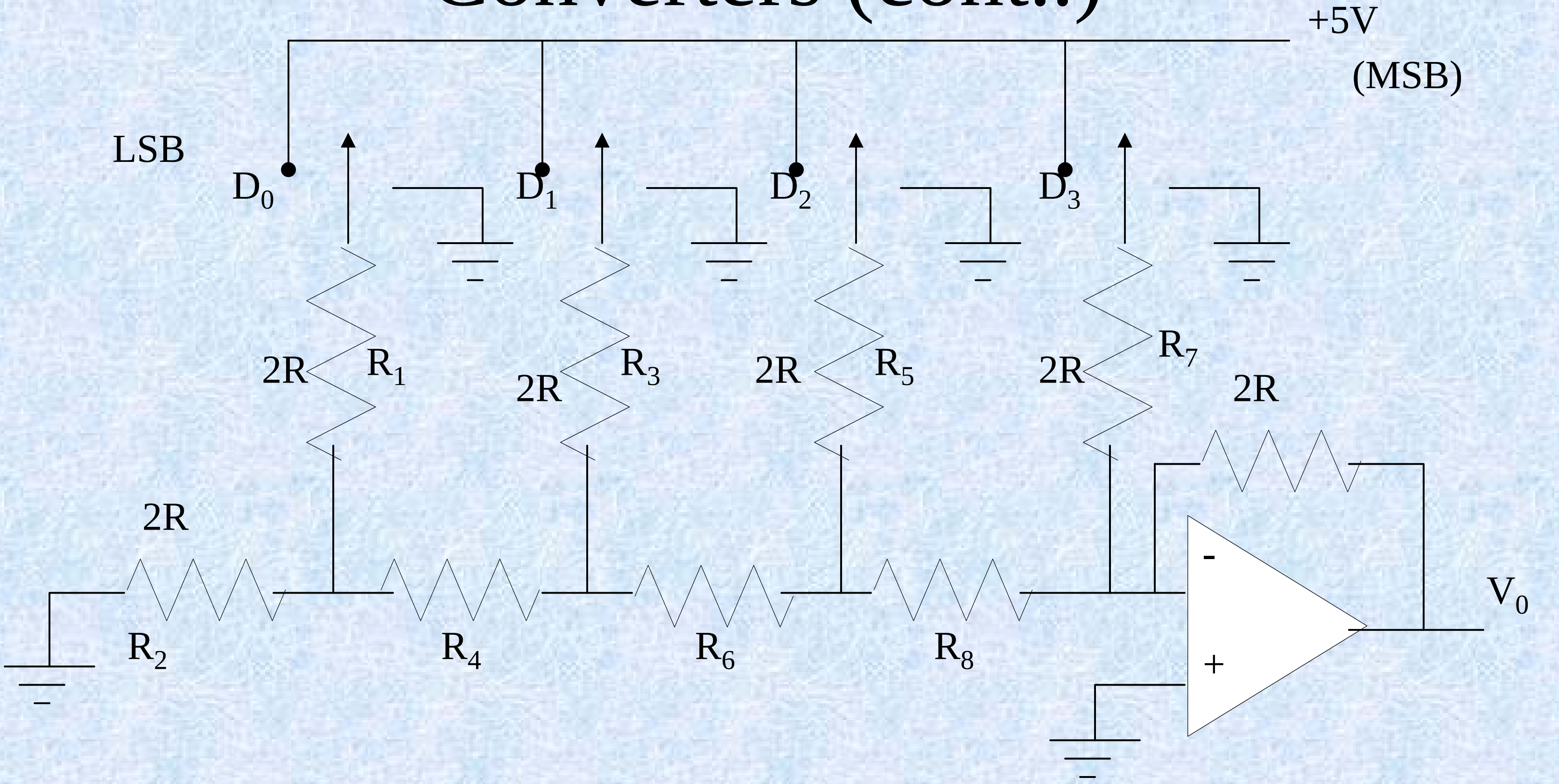


Fig:

Interfacing Digital To Analog Converters (cont..)

- The pin diagram of AD7523 is shown in fig the supply range is from +5V to +15V, while Vref may be any where between -10V to +10V. The maximum analog output voltage will be any where between -10V to +10V, when all the digital inputs are at logic high state.
- Usually a zener is connected between OUT1 and OUT2 to save the DAC from negative transients. An operational amplifier is used as a current to voltage converter at the output of AD to convert the current out put of AD to a proportional output voltage.

Interfacing Digital To Analog Converters (cont..)

- It also offers additional drive capability to the DAC output. An external feedback resistor acts to control the gain. One may not connect any external feedback resistor, if no gain control is required.
- ***EXAMPLE***: Interfacing DAC AD7523 with an 8086 CPU running at 8MH_Z and write an assembly language program to generate a sawtooth waveform of period 1ms with Vmax 5V.
- Solution: Fig shows the interfacing circuit of AD 74523 with 8086 using 8255. program gives an ALP to generate a sawtooth waveform using circuit.

Example (cont..)

```
ASSUME      CS:CODE
CODE        SEGMENT
START:      MOV AL,80h          ;make all ports output
            OUT   CW, AL
AGAIN:      MOV   AL,00h        ;start voltage for ramp
BACK :      OUT   PA, AL
            INC   AL
            CMP   AL, 0FFh
            JB    BACK
            JMP   AGAIN
            CODE  ENDS
            END   START
```

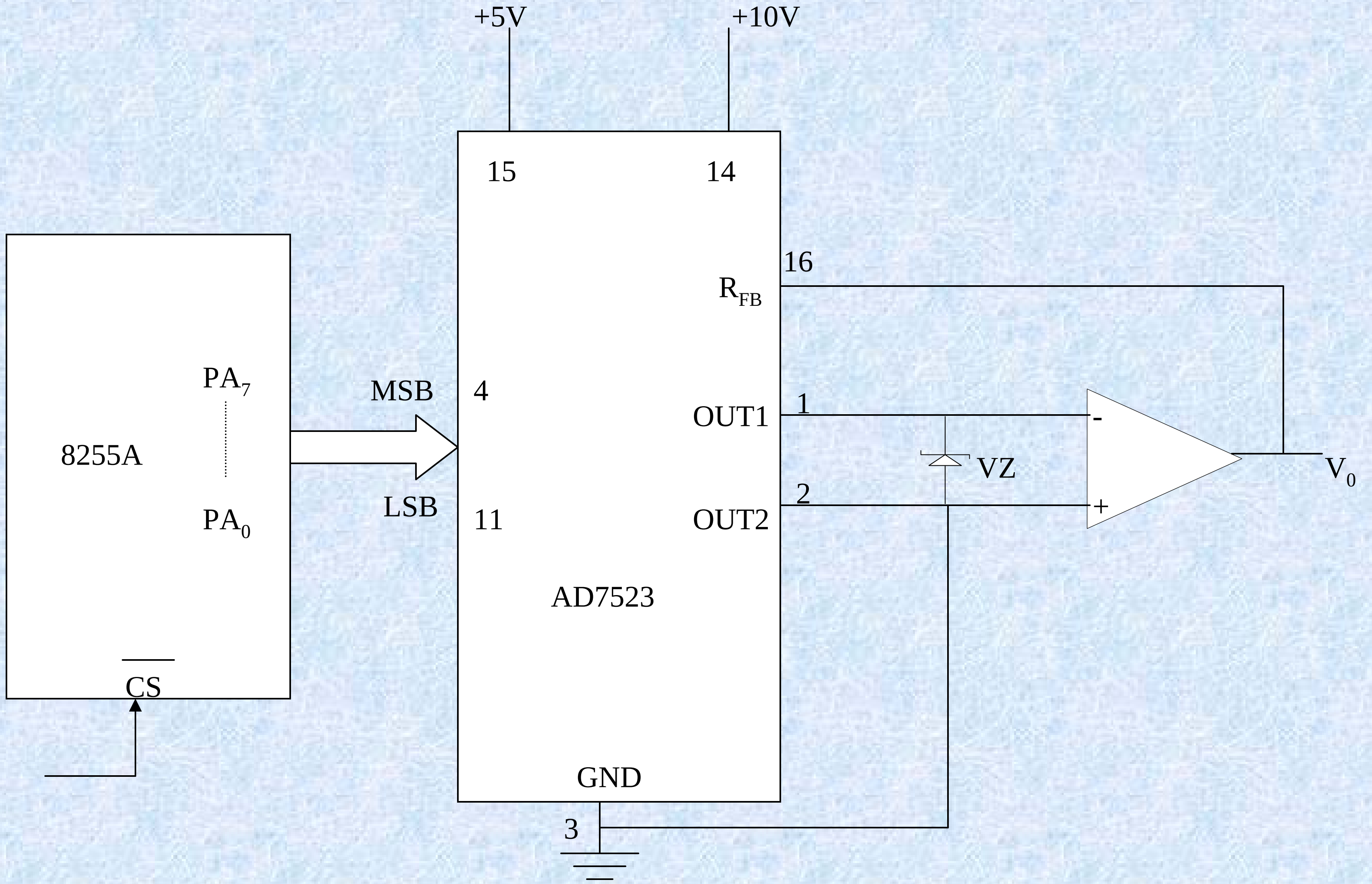


Fig: Interfacing of AD7523

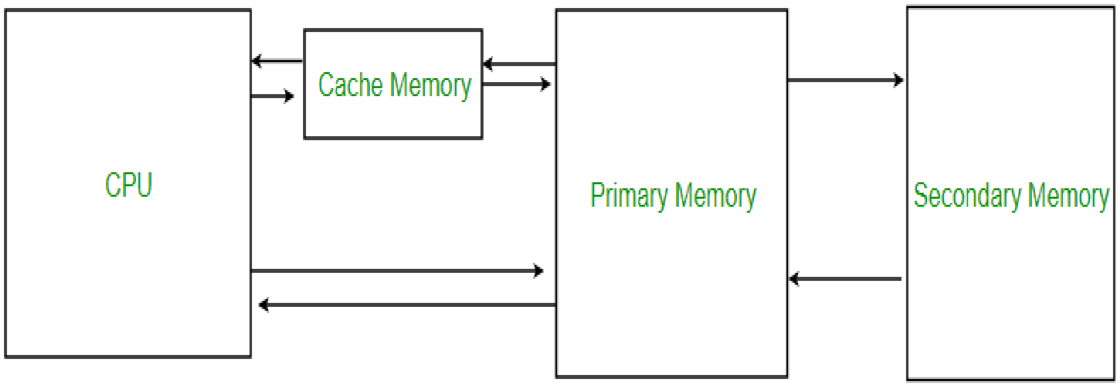
Interfacing Analog to Digital Data Converters (cont..)

- In the above program, port A is initialized as the output port for sending the digital data as input to DAC. The ramp starts from the 0V (analog), hence AL starts with 00H. To increment the ramp, the content of AL is increased during each execution of loop till it reaches F2H.
- After that the saw tooth wave again starts from 00H, i.e. 0V(analog) and the procedure is repeated. The ramp period given by this program is precisely 1.000625 ms. Here the count F2H has been calculated by dividing the required delay of 1ms by the time required for the execution of the loop once. The ramp slope can be controlled by calling a controllable delay after the OUT instruction.

Cache Memory

Cache Memory is a special very high-speed memory. It is used to speed up and synchronizing with high-speed CPU. Cache memory is costlier than main memory or disk memory but economical than CPU registers. Cache memory is an extremely fast memory type that acts as a buffer between RAM and the CPU. It holds frequently requested data and instructions so that they are immediately available to the CPU when needed.

Cache memory is used to reduce the average time to access data from the Main memory. The cache is a smaller and faster memory which stores copies of the data from frequently used main memory locations. There are various different independent caches in a CPU, which store instructions and data.



Levels of memory:

- **Level 1 or Register –**
  It is a type of memory in which data is stored and accepted that are immediately stored in CPU. Most commonly used register is accumulator, Program counter, address register etc.
- **Level 2 or Cache memory –**
  It is the fastest memory which has faster access time where data is temporarily stored for faster access.
- **Level 3 or Main Memory –**
  It is memory on which computer works currently. It is small in size and once power is off data no longer stays in this memory.
- **Level 4 or Secondary Memory –**
  It is external memory which is not as fast as main memory but data stays permanently in this memory.

Cache Performance:

When the processor needs to read or write a location in main memory, it first checks for a corresponding entry in the cache.

- If the processor finds that the memory location is in the cache, a **cache hit** has occurred and data is read from cache

- If the processor **does not** find the memory location in the cache, a **cache miss** has occurred. For a cache miss, the cache allocates a new entry and copies in data from main memory, then the request is fulfilled from the contents of the cache.

The performance of cache memory is frequently measured in terms of a quantity called **Hit ratio.**

Hit ratio = hit / (hit + miss) = no. of hits/total accesses

Virtual Memory

All of us are aware of the fact that our program needs to be available in main memory for the processor to execute it. Assume that your computer has something like 32 or 64 MB RAM available for the CPU to use. Unfortunately, that amount of RAM is not enough to run all of the programs that most users expect to run at once. For example, if you load the operating system, an e-mail program, a Web browser and word processor into RAM simultaneously, 32 MB is not enough to hold all of them. If there were no such thing as virtual memory, then you will not be able to run your programs, unless some program is closed. With virtual memory, we do not view the program as one single piece. We divide it into pieces, and only the one part that is currently being referenced by the processor need to be available in main memory. The entire program is available in the hard disk. As the copying between the hard disk and main memory happens automatically, you don't even know it is happening, and it makes your computer feel like is has unlimited RAM space even though it only has 32 MB installed. Because hard disk space is so much cheaper than RAM chips, it also has a n economic benefit.

Techniques that automatically move program and data blocks into the physical main memory when they are required for execution are called *virtual-memory* techniques.

80286 Microprocessor

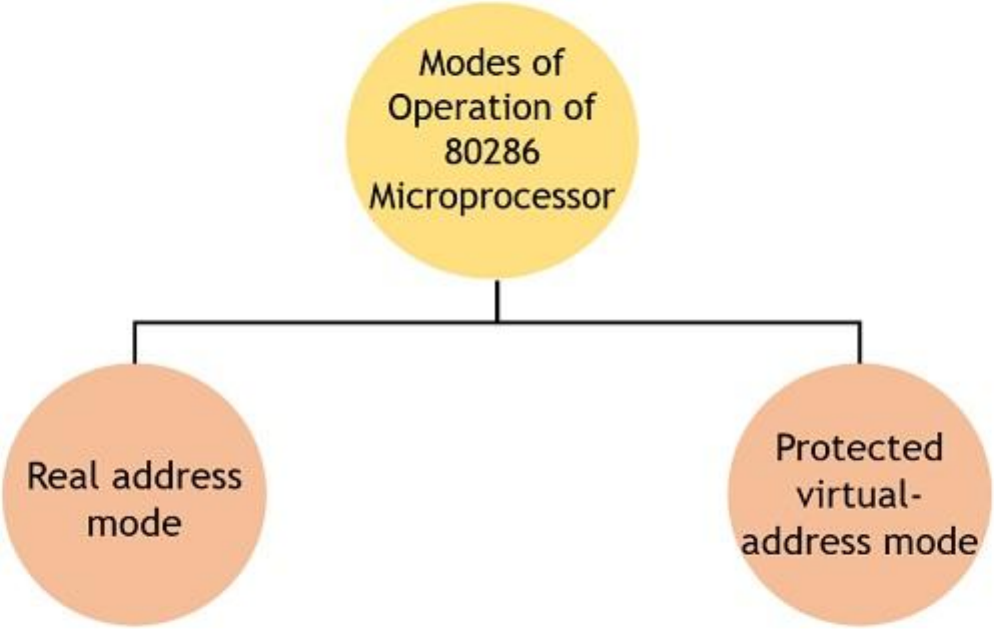
- 80286 Microprocessor is a 16-bit microprocessor that has the ability to execute 16-bit instruction at a time.
- It has non-multiplexed data and address bus.
- The size of data bus is 16-bit whereas the size of address bus is 24-bit.
- It was invented in February 1982 by Intel.
- 80286 Microprocessor was basically an advancement of 8086 microprocessor. Further in 1985, Intel produced upgraded version of 80286 which was a 32-bit microprocessor.

Factors that make 80286 more advantageous than 8086 microprocessor:

- It has non-multiplexed address and data bus that reduces operational speed.
- The addressable memory in case of 80286 is 16 MB.
- It offers an additional adder for address calculation.
- 80286 has faster multipliers that lead to quick operation.
- The performance per clock cycle of 80286 is almost twice when compared with 8086 or 8088.

Operating modes of 80286 microprocessor

80286 operates in two modes:


Modes of Operation of 80286 Microprocessor
Real address mode
Protected virtual-address mode

In real address mode, this microprocessor acts as a version of 8086 which is quite faster. Also without any special modification, the instruction programmed for 8086 can be executed in 80286. It offers memory addressability of 1 MB of physical memory.
The protected virtual-address mode of 80286 supports multitasking because multiple programs can be executed using virtual memory. This mode of 80286 offers memory addressability of 16 MB of physical memory along with 1 GB of virtual memory.
As using virtual memory, space for other programs can be saved. Sometimes bulky programs also do exist that cannot be stored in physical memory, so virtual memory is utilized in order to execute large programs.

This mode is used in 80286, so that in case of memory failure in real address mode, it can stay in protected manner.

What is virtual memory?
Virtual memory is that part of hard disk which can be utilized for storing large instructions inside the system. This extra memory can be addressed by the computer other than the physical memory.

When there exists an instruction that is to be loaded in the memory but whose size is greater than the provided physical memory. Then some part of hard disk is used in order to store that instruction, which is known as virtual memory.

Architecture of 80286 Microprocessor

The figure below shows the architectural representation of 80286 Microprocessor:

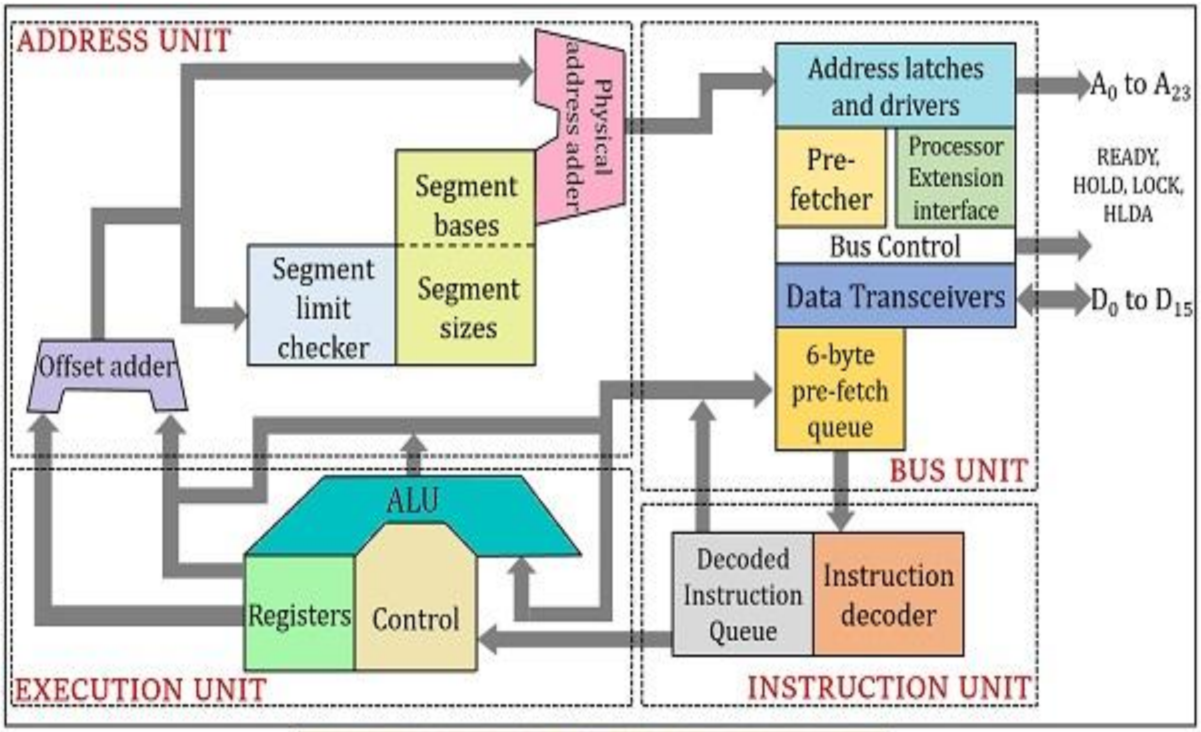
We have already mentioned earlier that it is a 16-bit microprocessor thus holds a 16-bit data bus and 24-bit address bus. Also, unlike the 8086 microprocessor, it offers non-multiplexed address and data bus, which increases the operating speed of the system.

80286 is composed of nearly around 125K transistors and the pin configuration has a total of 68 pins.

The CPU, central processing unit of 80286 microprocessor, consists of 4 functional block:

- Address Unit
- Bus Unit

- Instruction Unit
- Execution Unit



Block Diagram of 80286 Microprocessor

Electronics Desk

Firstly, the physical address from where the data or instruction is to be fetched is calculated, by the **address unit**. Once the physical address is calculated then the calculated address is handed over to the bus unit. More specifically we can say, that the calculated address is loaded on the address bus of the bus unit.
This address specifies the memory location from where the data or instruction is to be fetched. The fetching of data through the memory is done through the data bus. For faster execution of instruction, the BU fetches the instructions in advanced from the memory and stores them in the queue.

This is done through the bus control module. As we have discussed that the prefetched instructions are stored in a **6-byte instruction queue**. This instruction queue then further sends the instruction to the **instruction unit**.
The instruction unit on receiving the instructions now starts decoding the instruction. As instructions are stored in prefetched queue thus the decoder continuously decodes the fetched instructions and stores them into decoded instruction queue.

Now after the instructions gets decoded then further these are needed to be executed. So, the instructions from decoded instruction queue are fed to the **execution unit**. The main component of EU is ALU i.e., arithmetic and logic unit that performs the arithmetic and logic operations over the operand according to the decoded instruction.
Once the execution of the instruction is performed then the result of the operation i.e., the desired data is send to the register bank through the data bus.

As we have already discussed that 80286 is just a modified version of 8086. The register set in 80286 is same as that of 8086 microprocessor.

- It holds 8 general purpose registers of **16 bit** each.
- It contains 4 segment register each of **16-bit**.
- Also has status and control register and instruction pointer.

Interrupt of 80286 Microprocessor

We know that whenever an interrupt gets generated in a system, then the execution of the current program is stopped and the execution gets transferred to the new program location where the interrupt is generated.

But once the interrupt gets executed then in order to get back to the original program, its address as well as machine state must be stored in the stack. Basically there exist 3 categories of interrupt in 80286 microprocessor:

- External interrupt (Hardware interrupt)
- INT instruction interrupt (Software interrupt)
- Internally generated interrupt due to some exceptions

External or hardware initiate interrupt are those interrupts that gets generated due to an external input. And are basically of two types:
1. Maskable interrupt
2. Non-maskable interrupt

Sometimes when multiple programs are allowed to be executed in a system, then this leads to generation of INT instruction, and such an interrupt is known as **software interrupt**.

Another interrupt in 80286 exist due to some unusual conditions or situations generated in the system that leads to prevention of further execution of the current instruction.

So, this is all about the modes of operation, architecture and interrupts of 80286 microprocessor.

80486 Microprocessor

Features:

- The 32-bit 80486 is the next evolutionary step up from the 80386. It is also known as i486 or 486.
- It was introduced in 1989.
- One of the most obvious feature included in a 80486 is a built in math coprocessor. This coprocessor is essentially the same as the 80387 processor used with a 80386, but being integrated on the chip allows it to execute math instructions about three times as fast as a 80386/387 combination.
- 80486 has an 8Kbyte code and data cache.
- To make room for the additional signals, the 80486 is packaged in a 168 pin, pin grid array package instead of the 132 pin PGA used for the 80386.
- A 50 MHz 80486 executes around 40 million instructions per second on average.
- 32 bit address and data bus.
- The 80486 microprocessor is an improved version of the 80386 microprocessor that contains an 8K-byte cache and an 80387 arithmetic co processor. it executes many instructions in one clocking period.
- It contains 4GB RAM.
- The 80486 introduced the concept of instruction pipelining. Instruction pipelining partitions the execution process into multiple independent steps capable of occurring in parallel.
- The 80486 achieved instruction-level parallelism (ILP) through instruction pipelining. Prior to the 80486, the predecessor to the Pentium, each instruction was executed serially. In other words, each instruction began and finished execution before the execution of the next instruction could begin. This resulted in inefficient utilization of the processor's resources, as instruction execution did not require all of those resources simultaneously.
- The execution pipeline of the 80486 is partitioned into five stages, meaning that ideally five instructions are executing simultaneously (5 pipelining feature).

80486 overcome problem of Floating Point Operations:

Prior to the 80486 microprocessor the x86 processors had a companion processor like math coprocessor 80387 for floating point operations. These numeric extra processor required transactions between the main processor and coprocessor. It takes more time. But the 80486 has the FPU integrated into the processor which eliminates the additional bus cycles.

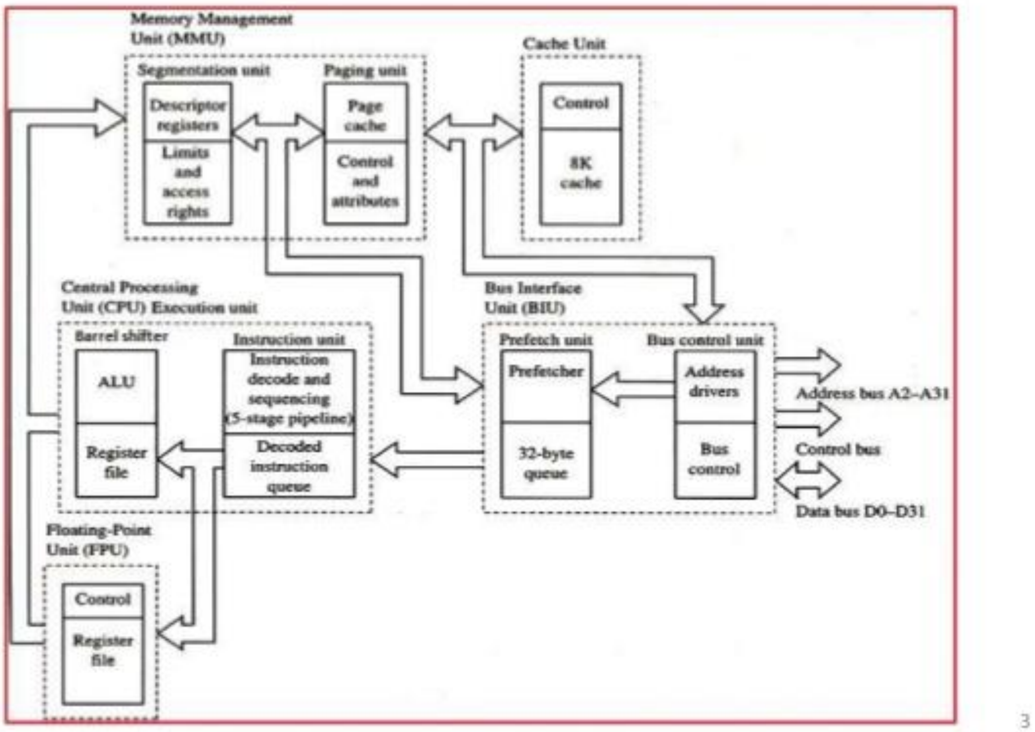
80486 microprocessor Architecture:

The 80486 microprocessor consists of the functional units illustrated in figure:

- Bus Interface Unit
- Cache Unit
- Instruction Pipeline/Decode Unit (consists of instruction prefetch and instruction decode units)

- Control Unit
- Floating-Point Unit
- Data Path Unit

Block diagram of 80486



The Bus Unit

The bus unit provides the physical interface between the 80486 and external devices. The bus unit consists of the following functional entities:

Address drivers/receivers:

When the 80486 is executing a bus cycle, the address drivers are used to drive the address out onto the processor's local address bus.

Bus control:

Senses when the microprocessor is communicating with 8- or 16-bit devices. Used to control the buses during the execution of a burst transfer.

Instruction Prefetch

The Prefetcher reads instructions in 16-byte blocks (lines). The line of code is read into both the internal cache and the 32-byte prefetch queue.

The 80486 Cache Unit

The 80486 microprocessor incorporates a cache controller and 8KB of fast access static RAM cache memory.

Two-Stage Instruction Decode

During the stage 1 decode, the opcode byte is decoded. During the stage 2 decode, the displacement is added to the address and any immediate operands are taken into account.

Execution

The instruction is executed.

Register Write-Back

Instruction execution is completed and the result written back to a target register.

The Floating-Point Unit

The floating-point unit executes the same instruction set as the 80387 Numeric Co-Processor extension.

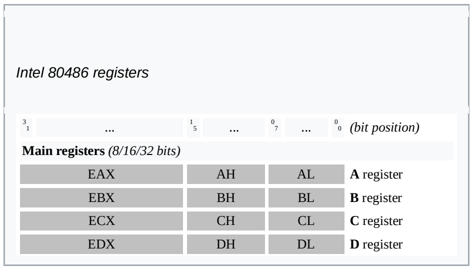
The Memory Management Unit (MMU)

The MMU consists of two sub-units:

The segmentation unit: Calculates effective and linear addresses from the segment and offset. It has been redesigned to generate one address per clock.

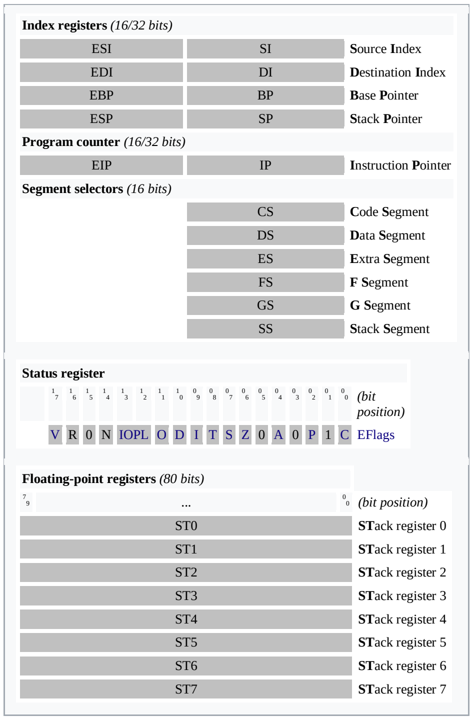
The paging unit: The paging unit translates the linear address to a physical address.

General-purpose registers



Intel 80486 registers

31	...	15	...	07	...	00	*(bit position)*
Main registers *(8/16/32 bits)*							
EAX		AH		AL			**A** register
EBX		BH		BL			**B** register
ECX		CH		CL			**C** register
EDX		DH		DL			**D** register



Index registers *(16/32 bits)*		
ESI	SI	**S**ource **I**ndex
EDI	DI	**D**estination **I**ndex
EBP	BP	**B**ase **P**ointer
ESP	SP	**S**tack **P**ointer
Program counter *(16/32 bits)*		
EIP	IP	**I**nstruction **P**ointer
Segment selectors *(16 bits)*		
	CS	**C**ode **S**egment
	DS	**D**ata **S**egment
	ES	**E**xtra **S**egment
	FS	**F** **S**egment
	GS	**G** **S**egment
	SS	**S**tack **S**egment

Status register																			
	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00	*(bit position)*
	V	R	0	N	IOPL		O	D	I	T	S	Z	0	A	0	P	1	C	EFlags

Floating-point registers *(80 bits)*			
79	...	00	*(bit position)*
	ST0		**ST**ack register 0
	ST1		**ST**ack register 1
	ST2		**ST**ack register 2
	ST3		**ST**ack register 3
	ST4		**ST**ack register 4
	ST5		**ST**ack register 5
	ST6		**ST**ack register 6
	ST7		**ST**ack register 7

Arithmetic logic unit (ALU). This unit handles all integer and bit-oriented math functions.

Flags. The flag register basically consists of two bit fields:

The flag status bits reflect the results of the previously executed instruction.

The flag control bits allow the programmer to alter certain operational characteristics of the microprocessor.

EFlags: EFLAGS indicate the condition of the microprocessor and control its operation. Figure 2-2 shows the flag registers of all versions of the microprocessor. The 8086-80286 contain a FLAG register (16 bits) and the 80386 and above contain an EFLAG register (32-bit extended flag register).

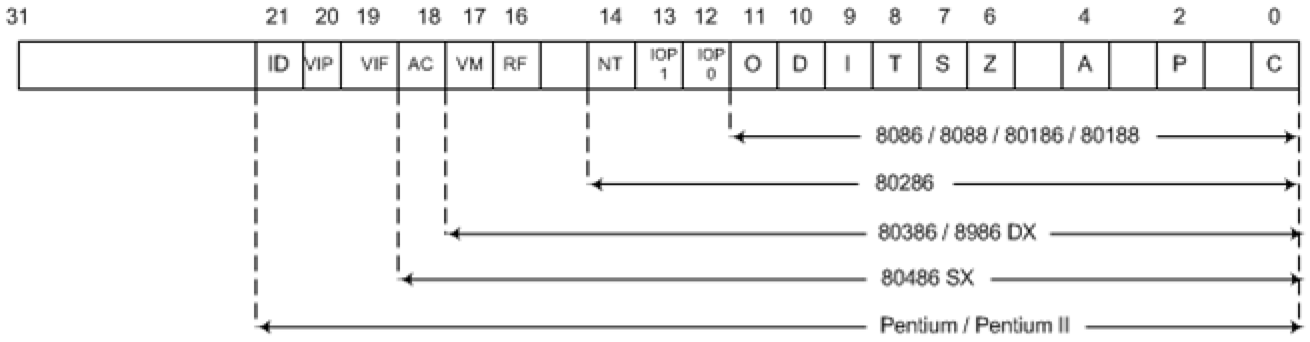


Figure 2-2 : The EFLAGS register.

The rightmost five flag bits and the overflow flag change after many arithmetic and logic instructions execute. The flags never change for any data transfer or program control operation. Some of the flags are also used to control features found in the microprocessor. Following is a list of each flag bit, with a brief description of their function.

C (carry) Carry holds the carry after addition or the borrow after subtraction. The carry flag also indicates error conditions, as dictated by some programs and procedures. This is especially true of the DOS function calls.

P (parity) Parity is a logic 0 for odd parity and a logic 1 for even parity. Parity is a count of ones in a number expressed as even or odd.

If a number contains zero one bits, it has even parity.

A(auxiliary carry) The auxiliary carry holds the carry (half-carry) after addition or the borrow after subtraction between bits positions 3 and 4 of the result.

Z (zero) The zero flag shows that the result of an arithmetic or logic operation is zero. If Z=1, the result is zero; if Z= 0, the result is not zero.

S (sign) The sign flag holds the arithmetic sign of the result after an arithmetic or logic instruction executes. If S=1, the sign bit (leftmost hit of a number) is set or negative; if S=0, the sign bit is cleared or positive.

T (trap) The trap flag enables trapping through an on-chip debugging feature. (A program is debugged to find an error or bug.) If the T flag is enabled (1), the microprocessor interrupts the flow of the program on conditions as indicated by the debug registers and control registers. lf the T flag is a logic 0, the trapping (debugging) feature is disabled.

I (interrupt) The interrupt flag controls the operation of the INTR (interrupt request) input pin. If I=1. the INTR pin is enabled: if I= 0, the INTR pin is disabled. The state of the I flag bit is controlled by the STI (set I flag) and CLI (clear I flag) instructions.

D (direction) The direction flag selects either the increment or decrement mode for the DI and/or SI registers during string instructions. If D=1, the registers are automatically decremented: if D=1, the registers are automatically incremented.

0 (overflow) Overflows occurs when signed numbers are added or subtracted. An overflow indicates that the result has exceeded the capacity of the machine. For unsigned operations, the overflow flag is ignored.

IOPL (I/0 privilege level) IOPL is used in protected mode operation to select the privilege level for I/O devices. If the current privilege level is higher or more trusted than the IOPL, I/O executes without hindrance. If the IOPL is lower than the current privilege level, an interrupt occurs, causing execution to suspend. Note that an IOPL of 00 is the highest or most trusted: if IOPL is 11, it is the lowest or least trusted.

NT (nested task) The nested task flag indicates that the current task is nested within another task in protected mode operation. This line is set when the task is nested by software.

RF (resume) The resume flag is used with debugging to control the resumption of execution after the next instruction.

VM (virtual mode) The VM flag bit selects virtual mode operation in a protected mode system.

AC (alignment check) The alignment check flag bit activates if a word or doubleword is addressed on a non-word or non-doubleword boundary. Only the 80486SX microprocessor contains the alignment check bit that is primarily used by its companion numeric coprocessor.

16-bit Processors

- The 8086 has 16-bit registers and a 16-bit external data bus, with 20-bit addressing giving a 1-MByte address space.
- The 8088 is similar to the 8086 except it has an 8-bit external data bus.
- The Intel 286,386,486 Processor:

80286 Microprocessor is a 16-bit microprocessor that has the ability to execute 16-bit instruction at a time. It has non-multiplexed data and address bus.

The size of data bus is 16-bit whereas the size of address bus is 24-bit. Further in 1985, Intel produced upgraded version of 80286 which was a 32-bit microprocessor. The addressable memory in case of 80286 is 16 MB.

- The Intel386 processor was the first 32-bit processor invented by Intel. It introduced 32-bit registers.

- The 32-bit 80486 is the next evolutionary step up from the 80386. It is also known as i486 or 486. It was introduced in 1989.

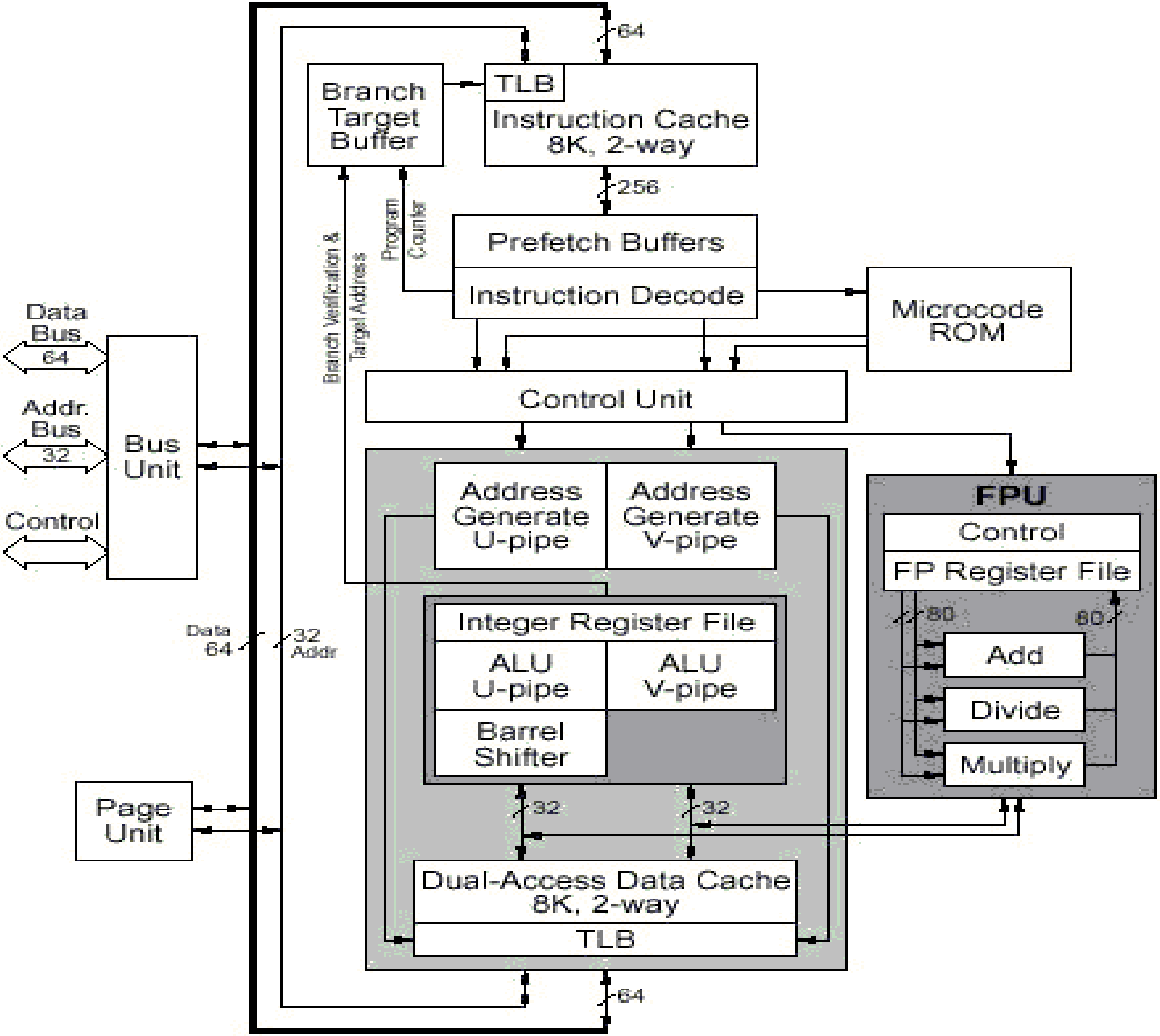
Pentium Processor

Features of Pentium

- After the 80486, Intel introduced the Pentium family of microprocessors in 1993.
- Pentium processors had a 32-bit internal architecture (registers).
- The external address and data buses were each 32 bits wide.
- Pentiums were based on superscalar architecture, which used two pipelines for parallel processing.
- They also had better cache memory than 80486 processors. The 80486 had a 8-KB cache for storing both code and data, the Pentium had a 16-KB cache, with 8-KB reserved for caching data and 8-KB reserved for caching instructions.

Pentium Architecture

It is not a load/store architecture. -- The instruction set is huge. We go over only a fraction of the instruction set. The text only presents a fraction. -- There are lots of restrictions on how instructions/operands are put together, but there is also an amazing amount of flexibility.

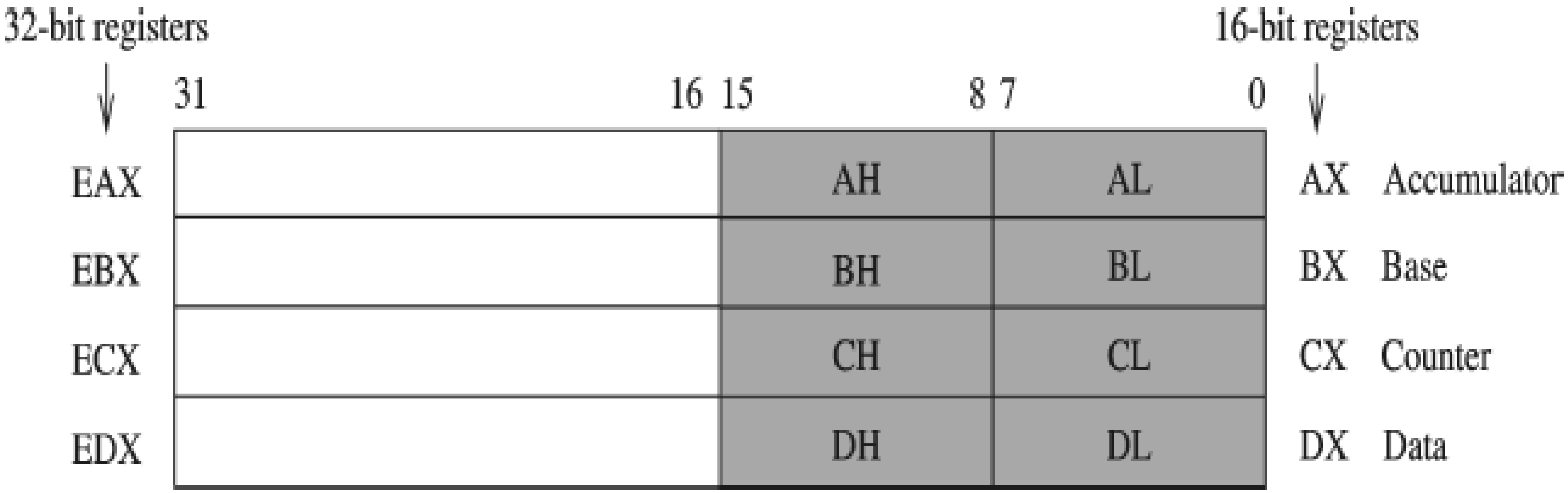

64
Branch Target Buffer
TLB
Instruction Cache 8K, 2-way
256
Program Counter
Branch Verification & Target Address
Prefetch Buffers
Instruction Decode
Microcode ROM
Data Bus 64
Addr. Bus 32
Control
Bus Unit
Control Unit
Address Generate U-pipe
Address Generate V-pipe
FPU
Control
FP Register File
80
80
Add
Divide
Multiply
Data 64
32 Addr
Integer Register File
ALU U-pipe
ALU V-pipe
Barrel Shifter
32
32
Page Unit
Dual-Access Data Cache 8K, 2-way
TLB
64

- **Code Cache:**
  - 2 way set associative cache
  - 256 lines b/w code cache and **prefetch buffer**, permitting prefetching of 32 bytes (256/8) of instructions.
- **Prefetch Buffers**: When instructions are prefetched from cache, they are placed into one set of prefetch buffers. Four prefetch buffers within the processor works as two independent pairs. The other set is used as when a branch operation is predicted. Prefetch buffer sends a pair of instructions to instruction decode.

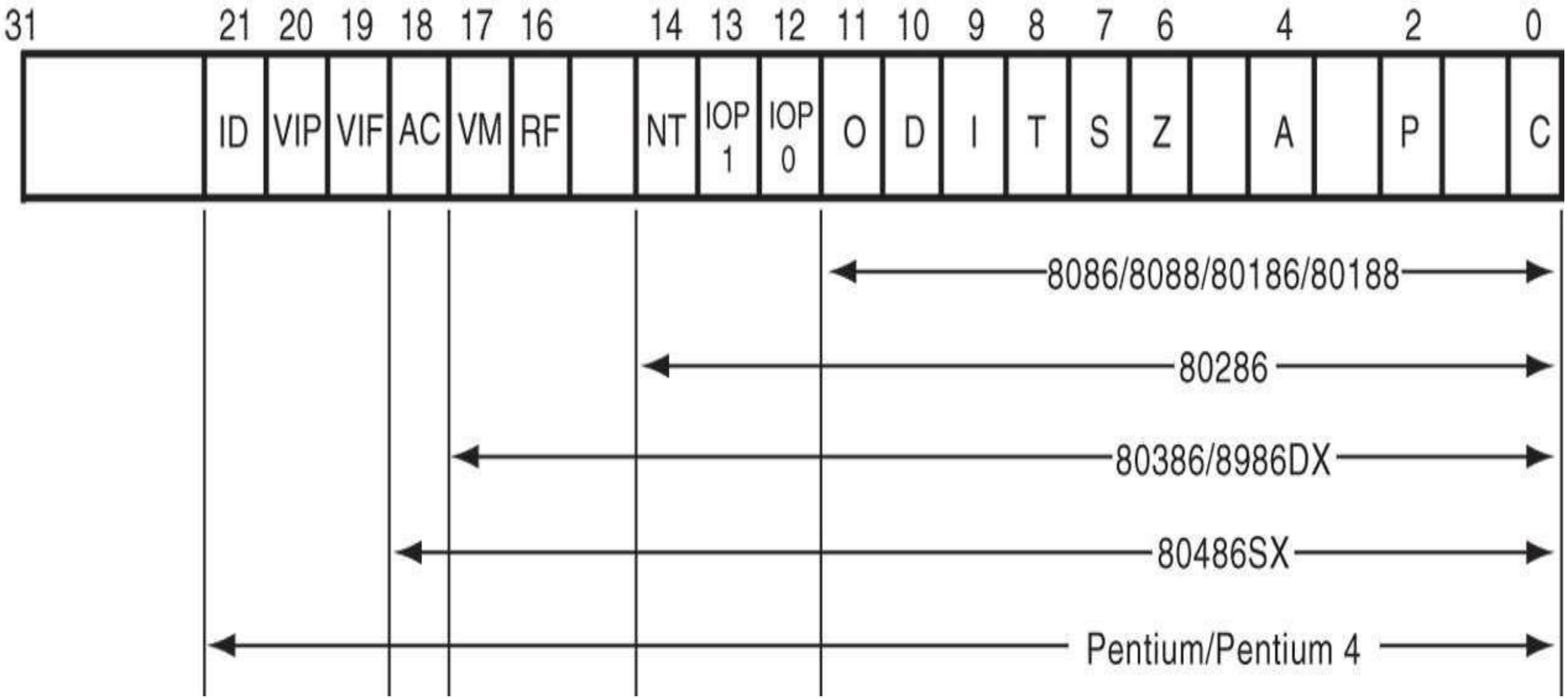
- **Instruction Decode Unit:** It occurs in two stages – Decode1 (D1) and Decode2(D2).
    - D1 checks whether instructions can be paired.
    - D2 calculates the address of memory resident operands.
- **Control Unit** : This unit interprets the instruction word and microcode entry point fed to it by Instruction Decode Unit.
- It handles exceptions, breakpoints and interrupts. It controls the integer pipelines and floating point sequences.

Pentium Registers

- Four 32-bit registers can be used as
- ∗ Four 32-bit register (EAX, EBX, ECX, EDX)
- ∗ Four 16-bit register (AX, BX, CX, DX)
- ∗ Eight 8-bit register (AH, AL, BH, BL, CH, CL, DH, DL)
- Some registers have special use
- ∗ ECX for count in loop instructions



Pentium Registers (Eflags)



31	21	20	19	18	17	16		14	13	12	11	10	9	8	7	6		4		2		0
	ID	VIP	VIF	AC	VM	RF		NT	IOP 1	IOP 0	O	D	I	T	S	Z		A		P		C

8086/8088/80186/80188

80286

80386/8986DX

80486SX

Pentium/Pentium 4

Flag bits, with a brief description of function.

- **C (carry)** holds the carry after addition or borrow after subtraction.
  - also indicates error conditions

P (parity) is the count of ones in a number expressed as even or odd. Logic 0 for odd parity; logic 1 for even parity.

- if a number contains three binary one bits, it has odd parity
- if a number contains no one bits, it has even parity

- **A (auxiliary carry)** holds the carry (half- carry) after addition or the borrow after subtraction between bit positions 3 and 4 of the result.
- **Z (zero)**shows that the result of an arithmetic or logic operation is zero.
- **S (sign)** flag holds the arithmetic sign of the result after an arithmetic or logic instruction executes.
- **T (trap)**The trap flag enables trapping through an on-chip debugging feature.
- **I (interrupt)** controls operation of the INTR (interrupt request) input pin.
- **D (direction)**selects increment or decrement mode for the DI and/or SI registers.
- **O (overflow)**occurs when signed numbers are added or subtracted, an overflow indicates the result has exceeded the capacity of the machine.

- **IOPL** used in protected mode operation to select the privilege level for I/O devices.
- **NT (nested task)** flag indicates the current task is nested within another task in protected mode operation.
- **RF (resume)** used with debugging to control resumption of execution after the next instruction.
- **VM (virtual mode)** flag bit selects virtual mode operation in a protected mode system.
- **AC, (alignment check)** flag bit activates if a word or doubleword is addressed on a non-word or non-doubleword boundary.

- **VIF** is a copy of the interrupt flag bit available to the Pentium 4–**(virtual interrupt)**
- **VIP (virtual)** provides information about a virtual mode interrupt for **(interrupt pending)** Pentium.
  - ▫ used in multitasking environments to provide virtual interrupt flags
- **ID (identification)** flag indicates that the Pentium microprocessors support the CPUID instruction.
  - ▫ CPUID instruction provides the system with information about the Pentium microprocessor

Difference between Microprocessor and Microcontroller

- Microprocessor consists of only a Central Processing Unit, whereas Micro Controller contains a CPU, Memory, I/O all integrated into one chip.
- Microprocessor is used in Personal Computers whereas Micro Controller is used in an embedded system.

Von Neumann and Harvard Architecture

- Von Neumann Architecture is a digital computer architecture whose design is based on the concept of stored program computers where program data and instruction data are stored in the same memory.
- Harvard Architecture is the digital computer architecture whose design is based on the concept where there are separate storage and separate buses (signal path) for instruction and data.

8051 Microcontroller

8051 microcontroller is designed by Intel in 1981. It is an 8-bit microcontroller based on Harvard architecture and primarily developed for use in embedded systems.

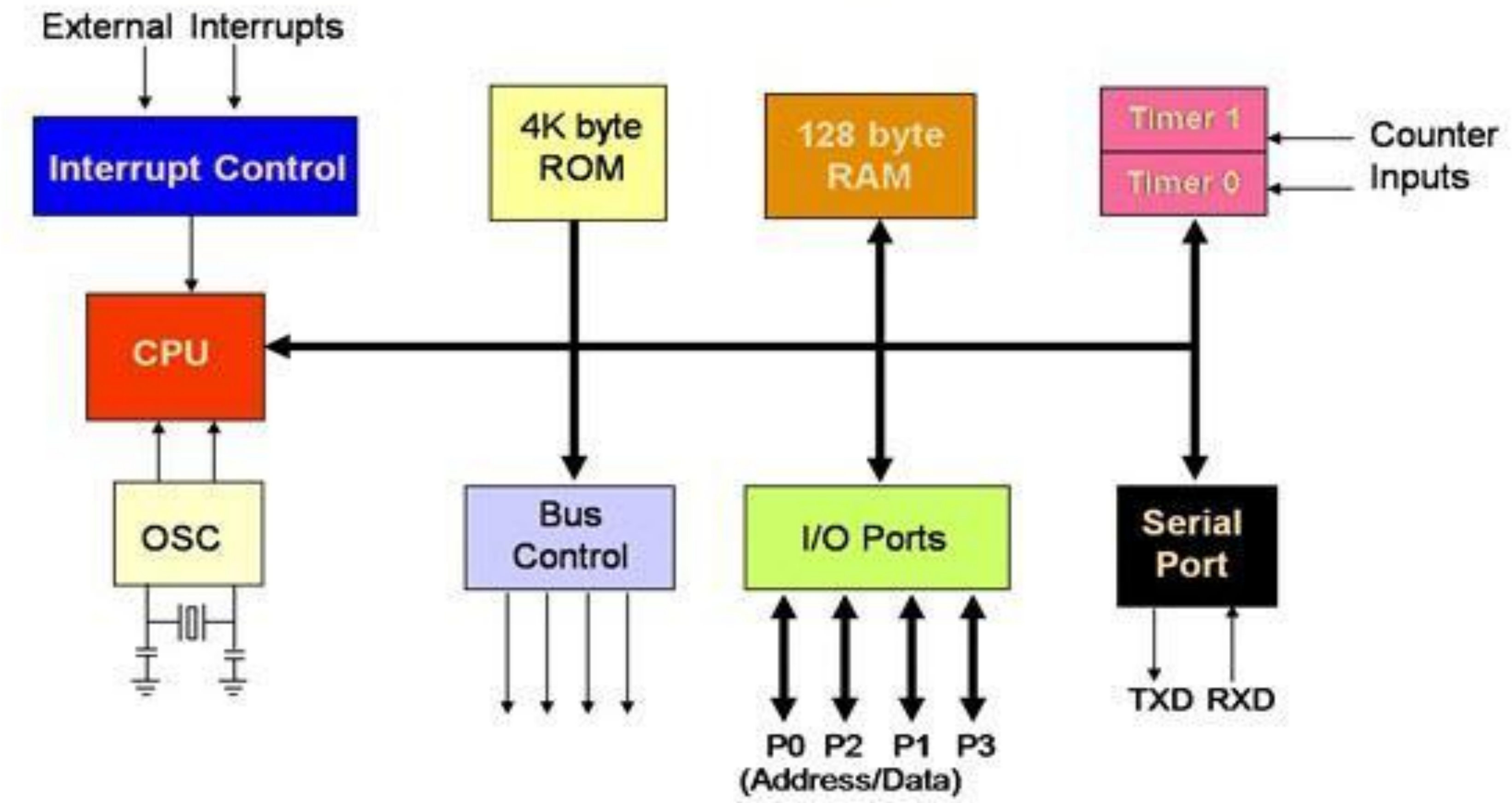
When it became widely popular, Intel allowed other manufacturers to make and market different flavors of 8051 with its code compatible with 8051. It means that if you write your program for one flavor of 8051, it will run on other flavors too, regardless of the manufacturer. This has led to several versions with different speeds and amounts of on-chip RAM.

At first, it was created using NMOS technology but as NMOS technology needs more power to function therefore Intel re-intended Microcontroller 8051 employing CMOS technology and a new edition came into existence with a letter ‘C’ in the title name, for illustration: 80C51.

Features:

- 4KB bytes on-chip program memory (ROM)
- 128 bytes on-chip data memory (RAM)
- Four register banks
- 8-bit bidirectional data bus
- 16-bit unidirectional address bus
- 32 general purpose registers each of 8-bit
- 16 bit Timers (usually 2, but may have more or less)
- Three internal and two external Interrupts
- 16-bit program counter and data pointer

Block Diagram of 8051 Microcontroller



- **CPU (Central Processor Unit):**

  As you may be familiar that the Central Processor Unit or CPU is the mind of any processing machine. It scrutinizes and manages all processes that are carried out in the Microcontroller.

- **Interrupts:**

  Interrupt is a subroutine call that reads the Microcontroller's key function or job and helps it to perform some other program which is extra important then. Interrupts provide us a method to postpone or delay the current process, carry out a sub-routine task and then all over again restart standard program implementation.

There are 5 interrupt supplies in the 8051 Microcontroller, two out of five are peripheral interrupts, two are timer interrupts and one is serial port interrupt.

- The interrupts of the 8051 microcontrollers have the following sources
- TF0 (Timer 0 Overflow Interrupt)
- TF1 (Timer 1 Overflow Interrupt)
- INT0 (External Hardware Interrupt)
- INT1 (External Hardware Interrupt)
- RI/TI (Serial Communication Interrupt)

- **Memory:**
- The micro-controller needs a program that is a set of commands. These programs need a storage space. The memory which is brought into play to accumulate the program of the Microcontroller is recognized as Program memory or code memory. In common language, it's also known as Read-Only Memory or ROM.
- The storage space which is employed to momentarily data storage for functioning is acknowledged as Data Memory and we employ Random Access Memory or RAM for this principle reason. Microcontroller 8051 contains code memory or program memory 4K so which has 4KB Rom and it also comprises data memory (RAM) of 128 bytes.

- **Bus**
- Fundamentally Bus is a group of wires which function as a communication canal or means for the transfer of Data. There are two types of buses:
- **Address Bus:** Microcontroller 8051 consists of a 16-bit address bus.
- **Data Bus:** Microcontroller 8051 comprise of 8 bits data bus.

- **Oscillator**
- As we all make out the Microcontroller is a digital circuit piece of equipment, thus it needs a timer for its function. For this function, Microcontroller 8051 consists of an on-chip oscillator.
- **Timer and Control Unit**
- The main function of a timer is to make a delay otherwise time gap among two events. This microcontroller includes two timers where each timer is 16-bit.

Registers

- Registers are used in the CPU to store information on temporarily basis which could be data to be processed, or an address pointing to the data which is to be fetched. In 8051.
- The most widely used registers of the 8051 are A (accumulator), B, R0-R7, DPTR (data pointer), and PC (program counter). All these registers are of 8-bits, except DPTR and PC.

- **Storage Registers in 8051**

  We will discuss the following types of storage registers here −

- Accumulator
- R register
- B register
- Data Pointer (DPTR)
- Program Counter (PC)
- Stack Pointer (SP)
- **Accumulator**

  The accumulator, register A, is used for all arithmetic and logic operations. If the accumulator is not present, then every result of each calculation (addition, multiplication, shift, etc.) is to be stored into the main memory. Access to main memory is slower than access to a register like the accumulator

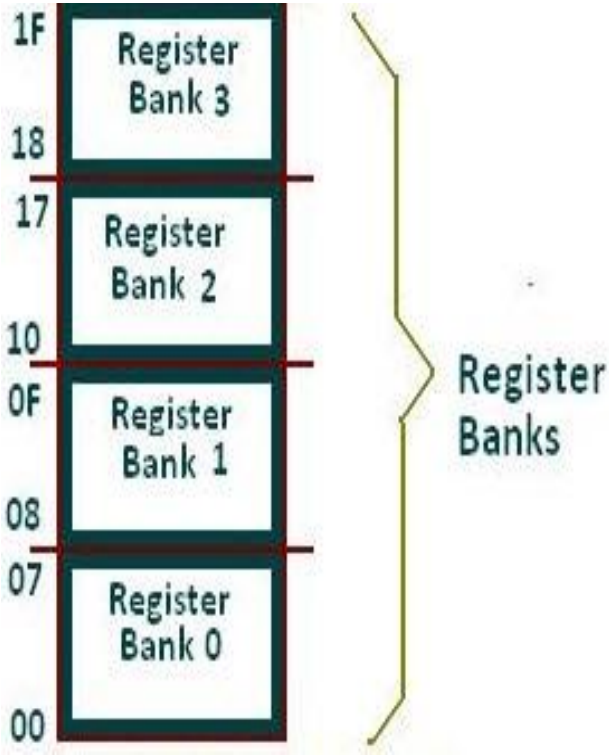
- The "R" Registers
- The "R" registers are a set of eight registers, namely, R0, R1 to R7. These registers function as auxiliary or temporary storage registers in many operations.
- The "B" Register
- The "B" register is very similar to the Accumulator in the sense that it may hold an 8-bit (1-byte) value. The "B" register is used only by two 8051 instructions: **MUL AB** and **DIV AB**.

- The Data Pointer
- The Data Pointer (DPTR) is the 8051’s only user-accessible 16-bit (2-byte) register. It is used by the 8051 to access external memory using the address indicated by DPTR.
- The Program Counter
- The Program Counter (PC) is a 2-byte address which tells the 8051 where the next instruction to execute can be found in the memory.

- **Stack Pointer**
- The stack is a section of a RAM used by the CPU to store information such as data or memory address on temporary basis. The register used to access the stack is known as the stack pointer register. The stack pointer in the 8051 is 8-bits wide.

Register Bank of 8051

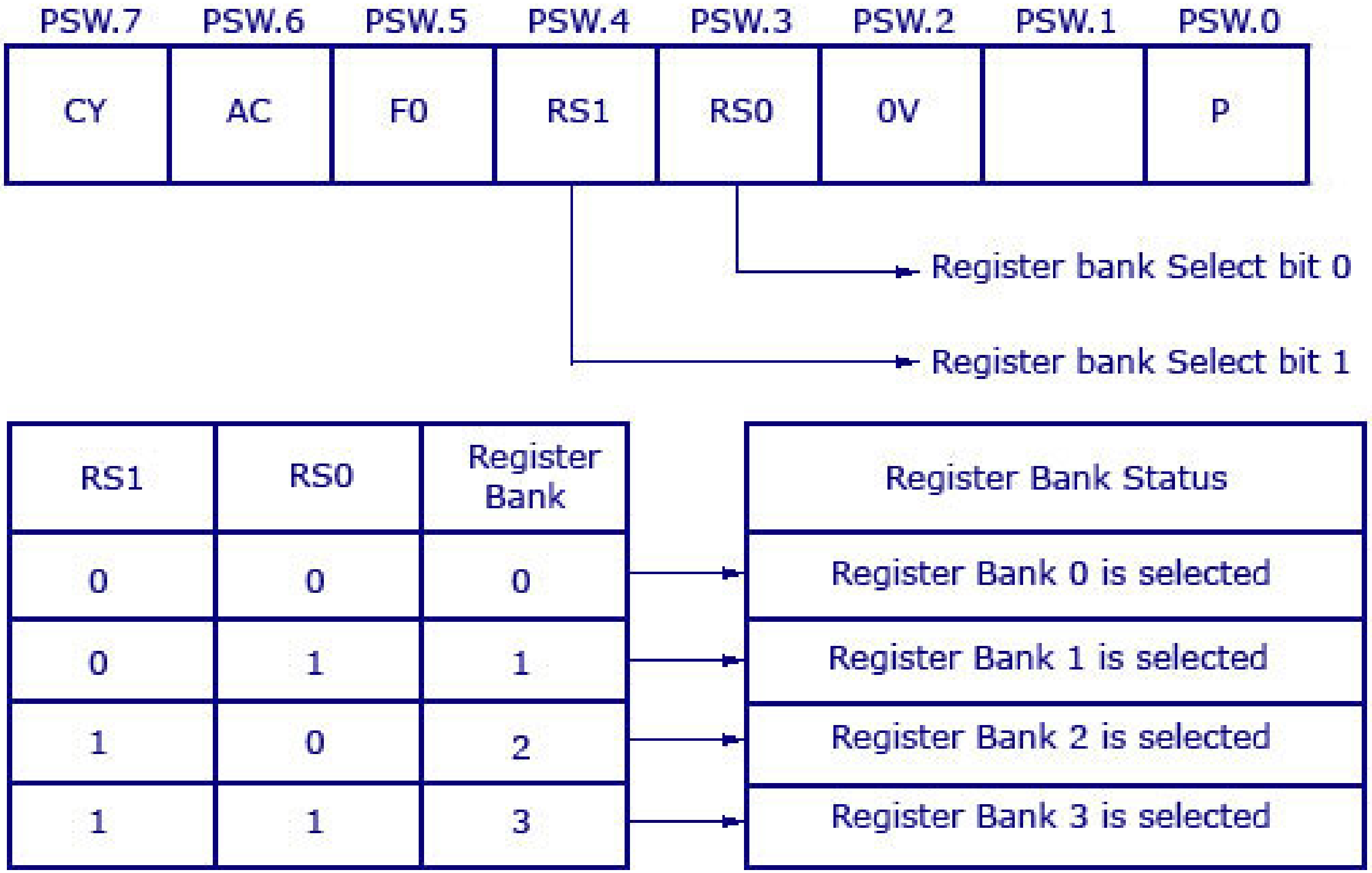
The 8051 microcontroller consists of four register banks, such as Bank0, Bank1, Bank2, Bank3 which are selected by the PSW (Program Status Word) register. These register banks are present in the internal RAM memory of the 8051 microcontroller.



Program Status Word (PSW)

- The program status word (PSW) register is an 8-bit register.I t is also referred to as the *flag register.* Although the PSW register is 8 bits wide, only *6* bits of it are used by the 8051.The two unused bits are user-definable flags.
- Four of the flags are called *conditional flags,* meaning that they indicate some conditions that result after an instruction is executed.
- These four are CY (carry), AC (auxiliary carry), P (parity), and OV (overflow).As seen from below figure, the bits PSW.3 and PSW.4 are designated as RS0 and RS1 as register selection bit, respectively, and are used to change the bank registers.

Processor Status Word



PSW.7	PSW.6	PSW.5	PSW.4	PSW.3	PSW.2	PSW.1	PSW.0
CY	AC	F0	RS1	RS0	0V		P

RS0 → Register bank Select bit 0

RS1 → Register bank Select bit 1

RS1	RS0	Register Bank
0	0	0
0	1	1
1	0	2
1	1	3

Register Bank Status
Register Bank 0 is selected
Register Bank 1 is selected
Register Bank 2 is selected
Register Bank 3 is selected

Applications of 8051 Microcontroller

- Light sensing and controlling devices.
- Temperature sensing and controlling devices.
- Fire detections and safety devices.
- Automobile applications.
- Defense applications.

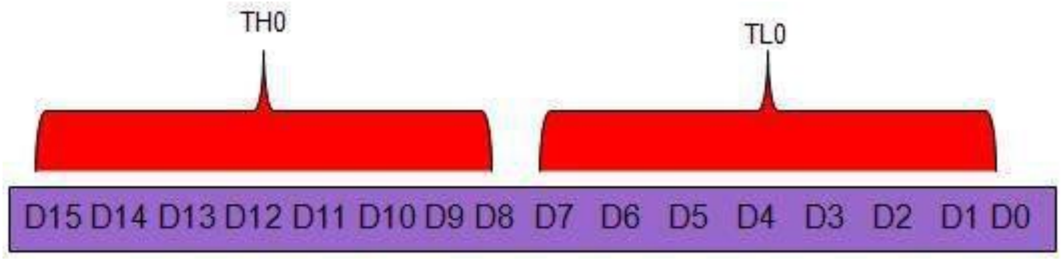
Thank You

Timers of 8051 and their Associated Registers

The 8051 has two timers, Timer 0 and Timer 1. They can be used as timers or as event counters. Both Timer 0 and Timer 1 are 16-bit wide. Since the 8051 follows an 8-bit architecture, each 16 bit is accessed as two separate registers of low-byte and high-byte.

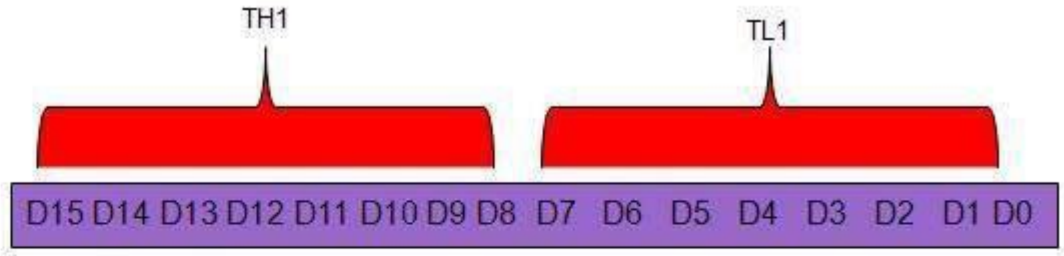
Timer 0 Register

The 16-bit register of Timer 0 is accessed as low- and high-byte. The low-byte register is called TL0 (Timer 0 low byte) and the high-byte register is called TH0 (Timer 0 high byte). These registers can be accessed like any other register. For example, the instruction **MOV TL0, #4H** moves the value into the low-byte of Timer #0.



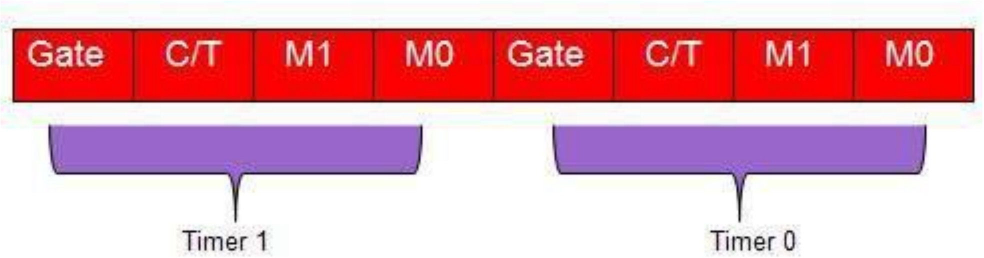
Timer 1 Register

The 16-bit register of Timer 1 is accessed as low- and high-byte. The low-byte register is called TL1 (Timer 1 low byte) and the high-byte register is called TH1 (Timer 1 high byte). These registers can be accessed like any other register. For example, the instruction **MOV TL1, #4H** moves the value into the low-byte of Timer 1.



TMOD (Timer Mode) Register

Both Timer 0 and Timer 1 use the same register to set the various timer operation modes. It is an 8-bit register in which the lower 4 bits are set aside for Timer 0 and the upper four bits for Timers. In each case, the lower 2 bits are used to set the timer mode in advance and the upper 2 bits are used to specify the location.



Gate − When set, the timer only runs while INT(0,1) is high.

C/T − Counter/Timer select bit.

M1 − Mode bit 1.

M0 − Mode bit 0.

GATE

Every timer has a means of starting and stopping. Some timers do this by software, some by hardware, and some have both software and hardware controls. 8051 timers have both software and hardware controls. The start and stop of a timer is controlled by software using the instruction **SETB TR1** and **CLR TR1** for timer 1, and **SETB TR0** and **CLR TR0** for timer 0.

The SETB instruction is used to start it and it is stopped by the CLR instruction. These instructions start and stop the timers as long as GATE = 0 in the TMOD register. Timers can be started and stopped by an external source by making GATE = 1 in the TMOD register.

C/T (CLOCK / TIMER)

This bit in the TMOD register is used to decide whether a timer is used as a **delay generator** or an **event manager**. If C/T = 0, it is used as a timer for timer delay generation. The clock source to create the time delay is the crystal frequency of the 8051. If C/T = 0, the crystal frequency attached to the 8051 also decides the speed at which the 8051 timer ticks at a regular interval.

Timer frequency is always 1/12th of the frequency of the crystal attached to the 8051. Although various 8051 based systems have an XTAL frequency of 10 MHz to 40 MHz, we normally work with the XTAL frequency of 11.0592 MHz. It is because the baud rate for serial communication of the 8051.XTAL = 11.0592 allows the 8051 system to communicate with the PC with no errors.

M1 / M2

M1	M2	Mode
0	0	13-bit timer mode.

0	1	16-bit timer mode.
1	0	8-bit auto reload mode.
1	1	Spilt mode.

Different Modes of Timers

Mode 0 (13-Bit Timer Mode)

Both Timer 1 and Timer 0 in Mode 0 operate as 8-bit counters. Timer register is configured as a 13-bit register consisting of all the 8 bits of TH1 and the lower 5 bits of TL1. The upper 3 bits of TL1 are indeterminate and should be ignored. The timer interrupt flag TF1 is set when the count rolls over from all 1s to all 0s. Mode 0 operation is the same for Timer 0 as it is for Timer 1.

Mode 1 (16-Bit Timer Mode)

Timer mode "1" is a 16-bit timer and is a commonly used mode. It functions in the same way as 13-bit mode except that all 16 bits are used. TLx is incremented starting from 0 to a maximum 255. Once the value 255 is reached, TLx resets to 0 and then THx is incremented by 1.

Mode 2 (8 Bit Auto Reload)

Both the timer registers are configured as 8-bit counters (TL1 and TL0) with automatic reload. Overflow from TL1 (TL0) sets TF1 (TF0) and also reloads TL1 (TL0) with the contents of Th1 (TH0), which is preset by software. The reload leaves TH1 (TH0) unchanged.

The benefit of auto-reload mode is that you can have the timer to always contain a value from 200 to 255. If you use mode 0 or 1, you would have to check in the code to see the overflow and, in that case, reset the timer to 200. In this case, precious instructions check the value and/or get reloaded. In mode 2, the microcontroller takes care of this. Once you have configured a timer in mode 2, you don't have to worry about checking to see if the timer has overflowed, nor do you have to worry about resetting the value because the microcontroller hardware will do it all for you. The auto-reload mode is used for establishing a common baud rate.

Mode 3 (Split Timer Mode)

Timer mode "3" is known as **split-timer mode**. When Timer 0 is placed in mode 3, it becomes two separate 8-bit timers. Timer 0 is TL0 and Timer 1 is TH0. Both the timers count from 0 to 255 and in case of overflow, reset back to 0. All the bits that are of Timer 1 will now be tied to TH0.

When Timer 0 is in split mode, the real Timer 1 (i.e. TH1 and TL1) can be set in modes 0, 1 or 2, but it cannot be started/stopped as the bits that do that are now linked to TH0. The real timer 1 will be incremented with every machine cycle.

Initializing a Timer

Decide the timer mode. Consider a 16-bit timer that runs continuously, and is independent of any external pins.

Initialize the TMOD SFR. Use the lowest 4 bits of TMOD and consider Timer 0. Keep the two bits, GATE 0 and C/T 0, as 0, since we want the timer to be independent of the external pins. As 16-bit mode is timer mode 1, clear T0M1 and set T0M0. Effectively, the only bit to turn on is bit 0 of TMOD. Now execute the following instruction −

```
MOV TMOD,#01h
```

Now, Timer 0 is in 16-bit timer mode, but the timer is not running. To start the timer in running mode, set the TR0 bit by executing the following instruction −

```
SETB TR0
```

Now, Timer 0 will immediately start counting, being incremented once every machine cycle.

Reading a Timer

A 16-bit timer can be read in two ways. Either read the actual value of the timer as a 16-bit number, or you detect when the timer has overflowed.

Detecting Timer Overflow

When a timer overflows from its highest value to 0, the microcontroller automatically sets the TFx bit in the TCON register. So instead of checking the exact value of the timer, the TFx bit can be checked. If TF0 is set, then Timer 0 has overflowed; if TF1 is set, then Timer 1 has overflowed.

RISC and CISC Processor

A **microprocessor** is a processing unit on a single chip. It is an integrated circuit which performs the core functions of a computer CPU. It is a multipurpose programmable silicon chip constructed using Metal Oxide Semiconductor (MOS) technology which is clock driven and register based. It accepts binary data as input and provides output after processing it as per the specification of instructions stored in the memory. These microprocessors are capable of processing 128 bits at a time at the speed of one billion instructions per second.

Characterstics of a micro processor:

- **Instruction Set –**
  Set of complete instructions that the microprocessor executes is termed as the instruction set.
- **Word Length –**
  The number of bits processed in a single instruction is called word length or word size. Greater the word size, larger the processing power of the CPU.
- **System Clock Speed –**
  Clock speed determines how fast a single instruction can be executed in a processor. The microprocessor's pace is controlled by the System Clock. Clock speeds are generally measured in million of cycles per second (MHz) and thousand million of cycles per second (GHz). Clock speed is considered to be a very important aspect of predicting the performance of a processor.

Classification of Microprocessors:

Besides the classification based on the word length, the classification is also based on the architecture i.e. Instruction Set of the microprocessor. These are categorised into RISC and CISC.

1. **RISC:**
   It stands for Reduced Instruction Set Computer. It is a type of microprocessor architecture that uses a small set of instructions of uniform length. These are simple instructions which are generally executed in one clock cycle. RISC chips are relatively simple to design and inexpensive.The setback of this design is that the computer has to repeatedly perform simple operations to execute a larger program having a large number of processing operations.
   Examples: SPARC, POWER PC etc.
2. **CISC:**
   It stands for Complex Instruction Set Computer. These processors offer the users, hundreds of instructions of variable sizes. CISC architecture includes a complete set of special purpose circuits that carry out these instructions at a very high speed. These instructions interact with memory by using complex addressing modes. CISC processors reduce the program size and hence lesser number of memory cycles are required to execute the programs. This increases the overall speed of execution.
   Examples: Intel architecture, AMD
3. **EPIC:**
   It stands for Explicitly Parallel Instruction Computing. The best features of RISC and CISC processors are combined in the architecture. It implements parallel processing of instructions rather than using fixed length instructions. The working of EPIC processors are supported by using a set of complex instructions that contain both basic instructions as well

as the information of execution of parallel instructions. It substantially increases the efficiency of these processors.

Below are few differences between RISC and CISC:

CISC	RISC
A large number of instructions are present in the architecture.	Very fewer instructions are present. The number of instructions are generally less than 100.
Some instructions with long execution times. These include instructions that copy an entire block from one part of memory to another and others that copy multiple registers to and from memory.	No instruction with a long execution time due to very simple instruction set. Some early RISC machines did not even have an integer multiply instruction, requiring compilers to implement multiplication as a sequence of additions.
Variable-length encodings of the instructions. **Example:** IA32 instruction size can range from 1 to 15 bytes.	Fixed-length encodings of the instructions are used. **Example:** In IA32, generally all instructions are encoded as 4 bytes.
Multiple formats are supported for specifying operands. A memory operand specifier can have many different combinations of displacement, base and index	Simple addressing formats are supported. Only base and displacement addressing is allowed.

CISC	RISC
registers.	
CISC supports array.	RISC does not supports array.
Arithmetic and logical operations can be applied to both memory and register operands.	Arithmetic and logical operations only use register operands. Memory referencing is only allowed by load and store instructions, i.e. reading from memory into a register and writing from a register to memory respectively.
Implementation programs are hidden from machine level programs. The ISA provides a clean abstraction between programs and how they get executed.	Implementation programs exposed to machine level programs. Few RISC machines do not allow specific instruction sequences.
Condition codes are used.	No condition codes are used.
The stack is being used for procedure arguments and return addresses.	Registers are being used for procedure arguments and return addresses. Memory references can be avoided by some procedures.

ARM processor and its features.

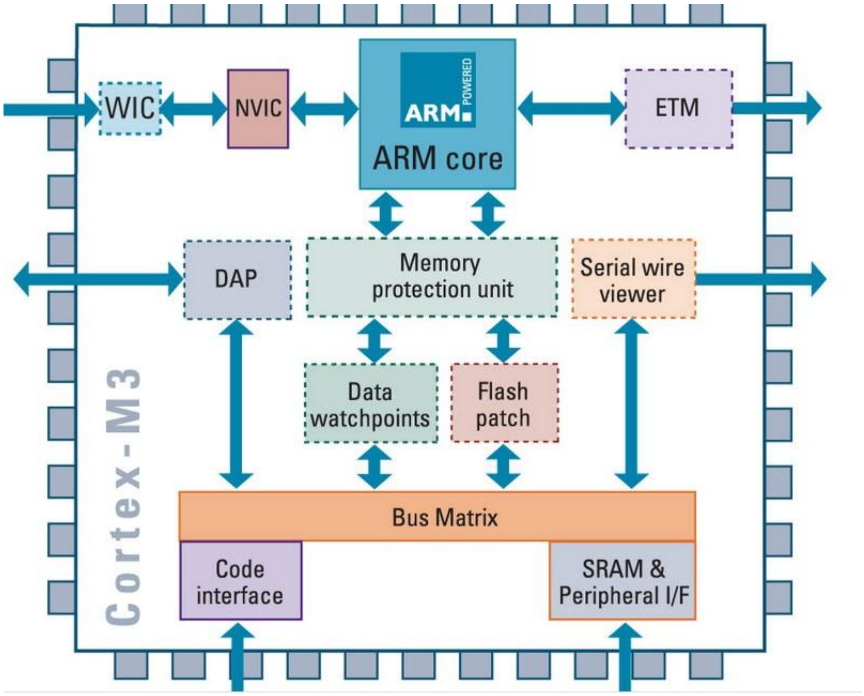
Advanced RISC Machine (ARM) Processor is considered to be family of Central Processing Units that is used in music players, smartphones, wearables, tablets and other consumer electronic devices.
The architecture of **ARM processor** is created by **Advanced RISC Machines**, hence name ARM. This needs very few instruction sets and transistors. It has very small size. This is reason that it is perfect fit for small size devices. It has less power consumption along with reduced complexity in its circuits.
They can be applied to various designs such as 32-bit devices and embedded systems. They can even be upgraded according to user needs.

The main features of ARM Processor are mentioned below :

1. **Multiprocessing Systems –**
   ARM processors are designed so that they can be used in cases of multiprocessing systems where more than one processors are used to process information. First AMP processor introduced by name of ARMv6K had ability to support 4 CPUs along with its hardware.
2. **Tightly Coupled Memory –**
   Memory of ARM processors is tightly coupled. This has very fast response time. It has low latency (quick response) that can also be used in cases of cache memory being unpredictable.
3. **Memory Management –**
   ARM processor has management section. This includes Memory Management Unit and Memory Protection Unit. These management systems become very important in managing memory efficiently.
4. **Thumb-2 Technology –**
   Thumb-2 Technology was introduced in 2003 and was used to create variable length instruction set. It extends 16-bit instructions of initial Thumb technology to 32-bit instructions. It has better performance than previously used Thumb technology.
5. **One cycle execution time –**
   ARM processor is optimised for each instruction on CPU. Each instruction is of fixed length that allows time for fetching future instructions before executing present instruction. ARM has CPI (Clock Per Insttuction) of one cycle.
6. **Pipelining –**
   Processing of instructions is done in parallel using pipelines. Instructions are broken down and decoded in one pipeline stage. The pipeline advances one step at a time to increase throughput (rate of processing).
7. **Large number of registers –**
   Large number of registers are used in ARM processor to prevent large amount of memory interactions. Registers contain data and addresses. These act as local memory store for all operations.

ARM Architecture.

The ARM architecture processor is an advanced reduced instruction set computing [RISC] machine and it’s a 32bit reduced instruction set computer (RISC) microcontroller. It was introduced by the Acron computer organization in 1987. This ARM is a family of microcontroller developed by makers like ST Microelectronics,Motorola, and so on. The ARM architecture comes with totally different versions like ARMv1, ARMv2, etc., and, each one has its own advantage and disadvantages.



The ARM processor conjointly has other components like the Program status register, which contains the processor flags (Z, S, V and C). The modes bits conjointly exist within the program standing register, in addition to the interrupt and quick interrupt disable bits; Some special registers: Some registers are used like the instruction, memory data read and write registers and memory address register.

Priority encoder: The encoder is used in the multiple load and store instruction to point which register within the register file to be loaded or kept .

Arithmetic Logic Unit (ALU)

The ALU has two 32-bits inputs. The primary comes from the register file, whereas the other comes from the shifter. Status registers flags modified by the ALU outputs. The V-bit output goes to the V flag as well as the Count goes to the C flag. Whereas the foremost significant bit really represents the S flag, the ALU output operation is done by NORed to get the Z flag. The ALU has a 4-bit function bus that permits up to 16 opcode to be implemented.

Booth Multiplier Factor

The multiplier factor has 3 32-bit inputs and the inputs return from the register file. The multiplier output is barely 32-Least Significant Bits of the merchandise. The entity representation of the multiplier factor is shown in the above block diagram. The multiplication starts whenever the beginning 04 input goes active. Fin of the output goes high when finishing.

Booth Algorithm

Booth algorithm is a noteworthy multiplication algorithmic rule for 2’s complement numbers. This treats positive and negative numbers uniformly. Moreover, the runs of 0’s or 1’s within the multiplier factor are skipped over without any addition or subtraction being performed, thereby creating possible quicker multiplication. The figure shows the simulation results for the multiplier test bench. It’s clear that the multiplication finishes only in16 clock cycle.

Barrel Shifter

The barrel shifter features a 32-bit input to be shifted. This input is coming back from the register file or it might be immediate data. The shifter has different control inputs coming back from the instruction register. The Shift field within the instruction controls the operation of the barrel shifter. This field indicates the kind of shift to be performed (logical left or right, arithmetic right or rotate right). The quantity by which the register ought to be shifted is contained in an immediate field within the instruction or it might be the lower 6 bits of a register within the register file.

Control Unit

For any microprocessor, control unit is the heart of the whole process and it is responsible for the system operation,so the control unit design is the most important part within the whole design. The control unit is sometimes a pure combinational circuit design. Here, the control unit is implemented by easy state machine. The processor timing is additionally included within the control unit. Signals from the control unit are connected to each component within the processor to supervise its operation.