Syllabus

29 March 2022 04:31 PM

UNIT 1	**Basics of Digital Electronics :** Binary ,Octal & Hexadecimal number System, Parity Code,7-bit hamming code, Logic gates ,K-Map, Multiplexers &De multiplexers , Flip-flops, Registers, Counters, Introduction to D/A,A/D convertor.
Unit 2	**8086 Microprocessor:** 8086 internal Architecture, memory Organization, Addressing modes , Accessing immediate & Register data ,memory accessing. **Instruction set of 8086, Programming with 8086:** 8086 data transfer instruction, Arithmetic instruction, Bit manipulation instruction, String instruction, Conditional & unconditional branch instruction ,Process control instruction. Use of Assembler Debug, Development cycle, debugging software Modular Programming, Procedures Develop programs in assembly language
Unit 3	**Designing 8086 CPU** Basic 8086 CPU hardware design, Generating CPU clock and reset signals, Bus types and buffering techniques, 8086 minimum mode CPU module, 8086

	maximum mode CPU module Design minimum mode CPU module using appropriate tool such as ORCAD
Unit 4	Main memory design-SRAM,DRAM,ROM & interfacing Basic input-output-Parallel, serial programmed and interrupt driven I/O,DMA
Unit 5	**Peripheral Controllers** 8255, 8259, 8251

Registers ; It is a group of flip flops

Its, basic function is to hold information

Application of registers include, serial addition, error-control coding, pseudo-random sequence generator

In typical 4-bit register, common clock input triggers all the flip flops & the binary data available at four inputs are transfered to registers

Counters ; It is essentially a register that goes through a predetermined sequence of states

The gates in counter are connected in such a way to produce a prescribed sequence of binary states

Generally used as pattern generators

Available in two catagories ; Ripple counters
Synchronous counters

The counting sequence, often depicted by graph ; State diagram

flip-flops

Basic difference b/w flip-flops & latch is

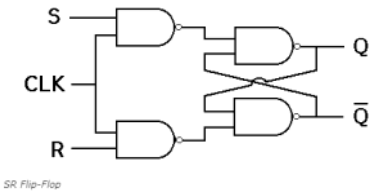
flip flop ; edge triggered

latch ; level triggered

Types of flip-flops

1. SR flip-flop
2. JK flip-flop (Race around Condition)
3. D flip-flop
4. T flip-flop

1. SR flip-flop



SR Flip-Flop

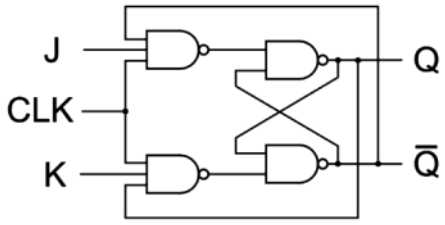
S ; Set Input
R ; Reset Input

If S is active then Q will be high

{S active '1'} for understanding purpose only

S	R	Q	Q'
0	0	0	1
0	1	0	1
1	0	1	0
1	1	∞	∞

2. JK flip-flop



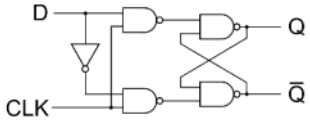
JK Flip-Flop

- Improved version of SR

J	K	Q	Q'
0	0	0	0
0	1	0	0
1	0	0	1
1	1	0	1
0	0	1	1
0	1	1	0
1	0	1	1
1	1	1	0

③ D flip-flop

D flip-flop is a better alternative that is very popular with digital electronics. They are commonly used for counters and shift-registers and input synchronisation.



D Flip-Flop

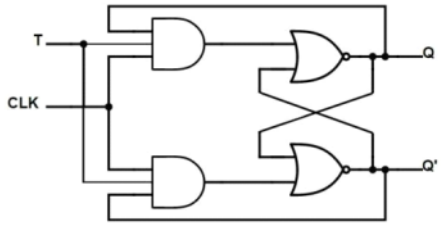
In this, the output can be only changed at the clock edge, and if the input changes at other times, the output will be unaffected.

Clock	D	Q	Q'
↓ » 0	0	0	1
↑ » 1	0	0	1
↓ » 0	1	0	1
↑ » 1	1	1	0

The change of state of the output is dependent on the rising edge of the clock. The output (Q) is same as the input and can only change at the rising edge of the clock.

④ T flip-flop

A T flip-flop is like a JK flip-flop. These are basically a single input version of JK flip-flops. This modified form of JK flip-flop is obtained by connecting both inputs J and K together. It has only one input along with the clock input.



These flip-flops are called T flip-flops because of their ability to complement its state (i.e.) Toggle, hence the name Toggle flip-flop.

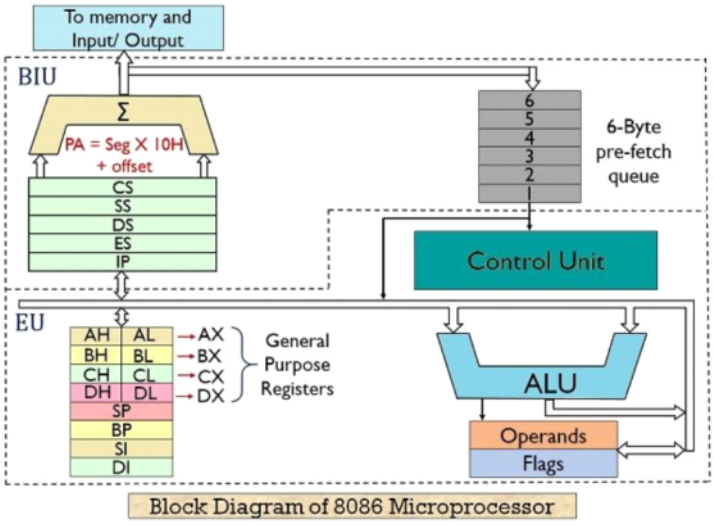
T	Q	Q (t+1)
0	0	0
1	0	1
0	1	1
1	1	0

Race Around Condition in JK Flip-flop

- For J-K flip-flop, if J=K=1, and if clk=1 for a long period of time, then output Q will toggle as long as CLK remains high which makes the output unstable or uncertain.
- This is called a race around condition in J-K flip-flop.
- We can overcome this problem by making the clock =1 for very less duration.
- The circuit used to overcome race around conditions is called the Master **Slave JK** flip flop.

8086 Internal Architecture

17 June 2022 08:52 AM



Block Diagram of 8086 Microprocessor

Electronics Desk

8086

BIU

Bus Interface Unit

→ fetches sequenced instruction from the memory

→ Find physical address of the location

→ Manage 6-byte pre-fetch queue where pipelined instructions are stored

- Contains Segment Register
- And 6byte pre fetch queue
- BIU manages data, addresses, & Control buses

EU

Execution Unit

- Performs decoding & execution of instructions.
- Contains

→ Control Unit; Produces control signal to general purpose registers & ALU

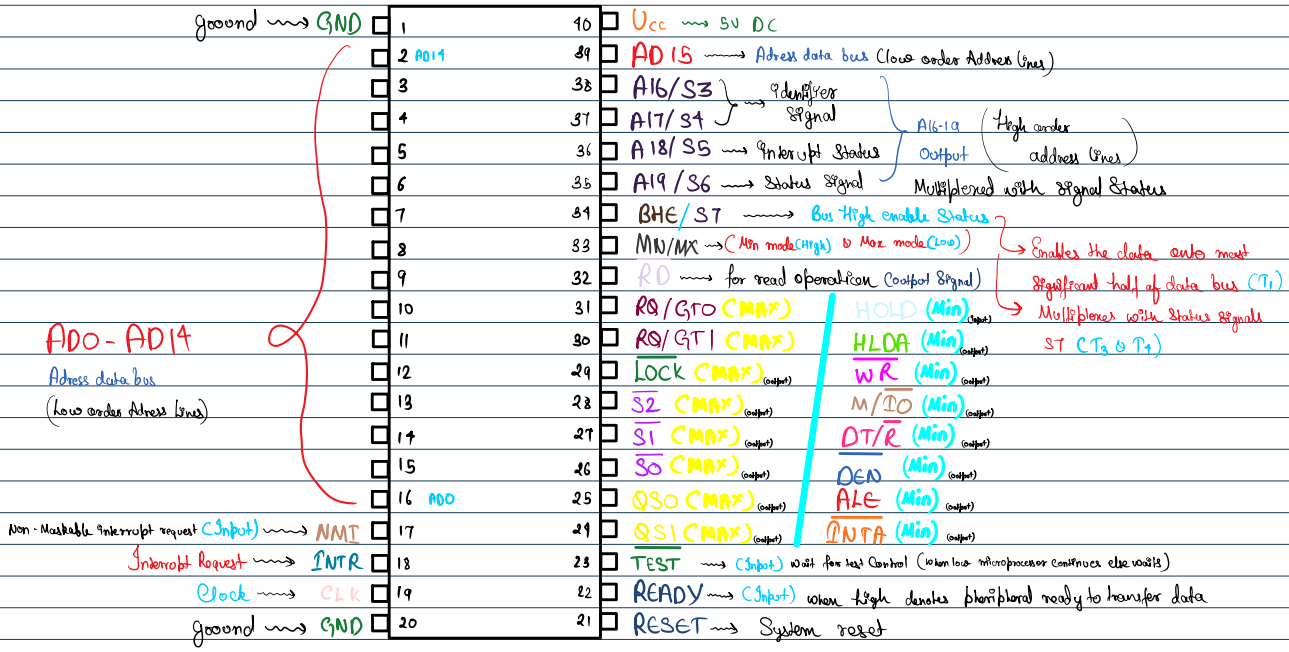
→ ALU; Arithmatic logic Unit carries out logical task

flags; these hold states generated by ALU

Operands; Temp store values at

time of operation

8086 Microprocessor



Operating modes of 8086

Minimum Mode
MN/MX = High

INTA : (Output) Interrupt acknowledge

ALE : (Output) Address Latch enable (high during T_1)

DEN : (Output) Data Enable

DT/R : (Output) Data Transmit / Recives (high : Data sent out, low : Data recived)

M/IO : (Output) Memory or I/O access

WR : (Output) Write

HLDA : (Output) Hold acknowledgement

HOLD : (Input) Hold, when other microcomp. System wants to use the data bus

Maximum Mode
MN/MX = Low

QS1, QS0 : (Output) Instruction queue Statues

S0, S1, S2 : (Output) Connected to bus controller, which generates memory & I/O access control signals

LOCK : (Output) when this signal is low, all interrupt are masked & no HOLD request is granted

RQ/GT1, RQ/GT0 : (Bidirectional) Local Bus priority control other processor ask CPU by these ones to release local bus.

In maximum mode operation signals WR, ALE, DEN, DT/R etc aren't available directly from the processor

↓

These signal are available from the Controller 8288

$\overline{S_2}$	$\overline{S_1}$	S_0	Operation
0	0	0	Interrupt acknowledgement
0	0	1	Read data from I/O
0	1	0	write data from I/O
0	1	1	Halt
1	0	0	Opcode fetch
1	0	1	Memory read
1	1	0	Memory write
1	1	1	Passive state

QS1	QS0	Operation
0	0	No operation
0	1	1st Byte of opcode from queue
1	0	Empty the queue
1	1	Subsequent byte from queue

Register Organization

Register

General Purpose

- AX Accumulator
- BX Base
- CX Code
- DX data

↓

Used to Store temporary data while instructions are executed

(16 bit length)

divided into two subparts of 8-bit length (high & low)

Segment

- CS Code Seg.
- DS data seg.
- SS Stack seg.
- ES Extra seg.

↓

Code & Instruction are Stored inside different segments

(20 bit length)

↓

16 bit for logical address

Pointer & Index

- SP Stack pointer
- BP Base ptr
- SI Source Index
- DI destination Index
- IP (Instruction ptr)

↓

These Store some address or memory location

Flag

9 flags

(States & Control)

↓

(16 bit length)

tells the states of the process / processor

General purpose Registers

	D15 D8	D7 D0
AX	AH	AL
BX	BH	BL
CX	CH	CL
DX	DH	DL

General Purpose Registers

AX ; Accumulator ; 16 bit Register devided into two parts High & Low

AH (8-bit) , AL (8-bit)

generally used for arithematic & logical instruction

ADD AX, AX (AX = AX + AX)

BX ; Base Register ; 16-bit register , also devided into BH, BL

Used to store offset values MOV BL, 500H (BL = 500H)

CX ; Counter Register ;

Used in looping & Rotation

```
MOV CX, 0005
LOOP
```

DX ; Data Register ;

Used in multiplication , input/output port addressing

MUL BX (DX, AX = AX * BX)

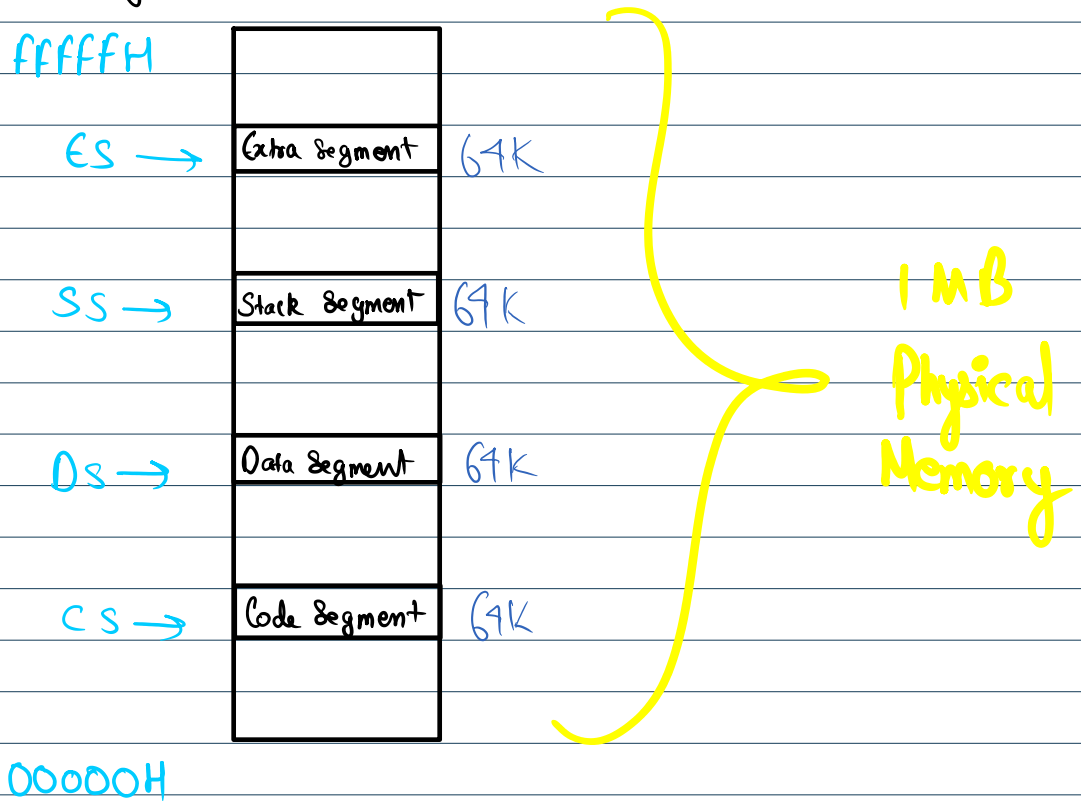
Segment Registers

17 June 2022 12:17 AM

Segment Register

It is a 20-bit wide physical address to access 1MB memory location.

As the microprocessor's register has only 16-bit registers, thus the memory is segmented



1. Data Segment (DS) ; Used to point to base address of data segment
2. Code Segment ; Used to point to base address of code segment
3. Stack Segment ; Stores the base address of stack memory segment
4. Extra Segment ; Stores the base address of Extra Memory segment

Pointer & Index Registers

These are 16-bit register, each of them is associated with with each segment register, in order to generate 20 bit physical address

Stack pointer	SP
Base pointer	BP
Source Index	SI
Destination Index	DI
Program Counter / Instruction Pointer	IP

(1) Stack pointer; holds the address of stack top (uppermost stack location)
There is no direct access to the register, modifications are done depending contents of stack

(2) Base pointer; primarly used in accessing parameters passed by the stack

(3) Source Index; Used as pointer addressing data as a source in some string related operation

(4) Destination Index; Used as pointer addressing data as a destination in some string related operation

(5) Instruction pointer; Gives the offset address of next instruction to be fetched from the code segment

When reset, it fetches the first instruction to

be performed.

8086 has 20bit memory which cannot be accessed via 16 bit segment Registers alone.

In order to access memory, 16-bit address is offset with respect to segment

So, physical address + offset

Suppose data segment holds the base address as 1000h & the data you need is present on 0020H memory location (offset) of data segment, the calculations are

- Left Shift 16 bit address present in segment by 4-bits

0001 0000 0000 0000 (0000)

- Add 16-bit offset address to shifted base address

```
  0001 0000 0000 0000 0000
+       0000 0000 0010 0000
-------------------------------
  0001 0000 0000 0010 0000
```

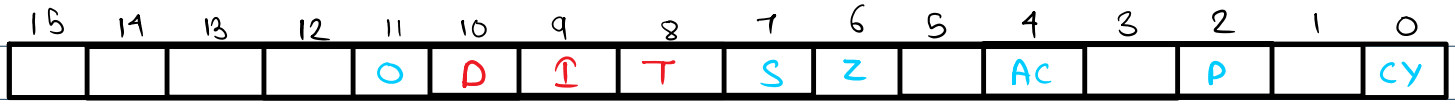
So the actual address turns out to be 10020h

Flags in 8086

Flags are special purpose Registers, which depending upon value of result after arithematic or logical operation the flag bit becomes either (0) or (1)

- Flags
  - Status flag
    1. Sign Flag (S)
    2. Zero flag (Z)
    3. Auxiliary Carry flag (AC)
    4. Parity flag (P)
    5. Carry flag (CY)
    6. Over flow flag (O)
  - Control flag
    1. Directional flag (D)
    2. Interrupt flag (I)
    3. Trap flag (T)

16-Bit flag Register



- Empty Spot : Undefined
- 6 Status flag
- 3 Control flag

Status flag

1. Sign flag : If MSB is 1 then number is -ve (1) [Set] as for +ve, MSB 0 (0) [Reset]

② Zero flag : After any arithmatical or logical operation
if result ⇒ 0 (00)H

then, zero flag [Set] (1)

③ Auxiliary Carry flag : Used in BCD number system

after arith/logical operation D(3) generates any carry & passes onto D(4) then the flag is [Set]

Only flag not accessible by programmer

④ Parity flag : If the result is in even parity
ie.
even number of 1's, then the flag is [Set]

⑤ Carry flag : Carry is generated when performing n-bit operation, the result is more than n bits
if so then the flag is [Set]
↳ also called borrow flag

⑥ Overflow flag : flag is set when, result of signed operation is too large to fit in available no. of bits

Control flags

① Directional flags : used in string Instructions

- If Set = 1 ; access memory from higher memory location towards lower memory location

vice versa

② Interrupt flag : This flag is for Interrupt

- If Set = 1 ; processor will recognise Interrupt

request from peripherals

vice versa

(3) Trap flag : Used for On-chip debugging

- If Set = 1 ; puts processor into single step mode for debugging

  (or)

  generate automatic internal interrupt after each instruction

vice versa.

8086 Instruction Set

1. Data transfer Instruction
2. Arithmatic instructions
3. Bit manuplation instruction
4. String Instruction
5. Program execution transfer instruction
6. Processor Control Instruction
7. Iteration Control Instruction

Data transfer Instruction

used to transfer data from source operand to destination operand

- MOV, POP, PPUSH, PUSHA, POPA (for word)
- IN, OUT (Instruction for Input/output port transfer)
- LEA, LDS, LES (Ins. for address transfer)
- LAHF, SAHF, PUSHF, POPF (Ins. for flag transfer)

Arithematic Instructions

Used to perform arithmetic operations like addition, substraction, multiplication, division

; ADD, ADC, INC, AAA, DAC (Addition operation)

; SUB, SBB, DEC, NPG, AAS (Substration operations)

; MUL, IMUL, AAM (Multiplication operations)

; DIV, IDIV, AAD (Division operations)

Bit Manuplation Instructions

Used to perform operation where data bits are involved

; NOT, AND, OR, XOR, TEST (for logical operations)

; SHL, SHR, SAL, SAR (for shift operations)

; ROL, ROR, RCR, RCL (for rotate operations)

String Instruction

Basically all operations involving strings

; REP, MOVS, COMS, INS, OUTS, SCAS, LODS

Program Execution & transfer Instruction

(or Branch & loop Instruction)

Used to transfer/Branch Instruction during execution

- transfer the instruction during execution without condition

; CALL, RET, JMP

- transfer the instruction during execution with condition

; JAE/JBE, JC/JNC, JE/JZ, JGE JGNE,

etc.

Processor Control Instruction

Used to control processor by setting/resetting carry flag values

; STC, STD, STI, CLC, CLD, CLI

Set flag — Clear flag

Iteration Control Flag

Used to control the number of times the instruction is executed

; LOOP, LOOPE/LOOPZ, JCXZ

~NE ~NZ

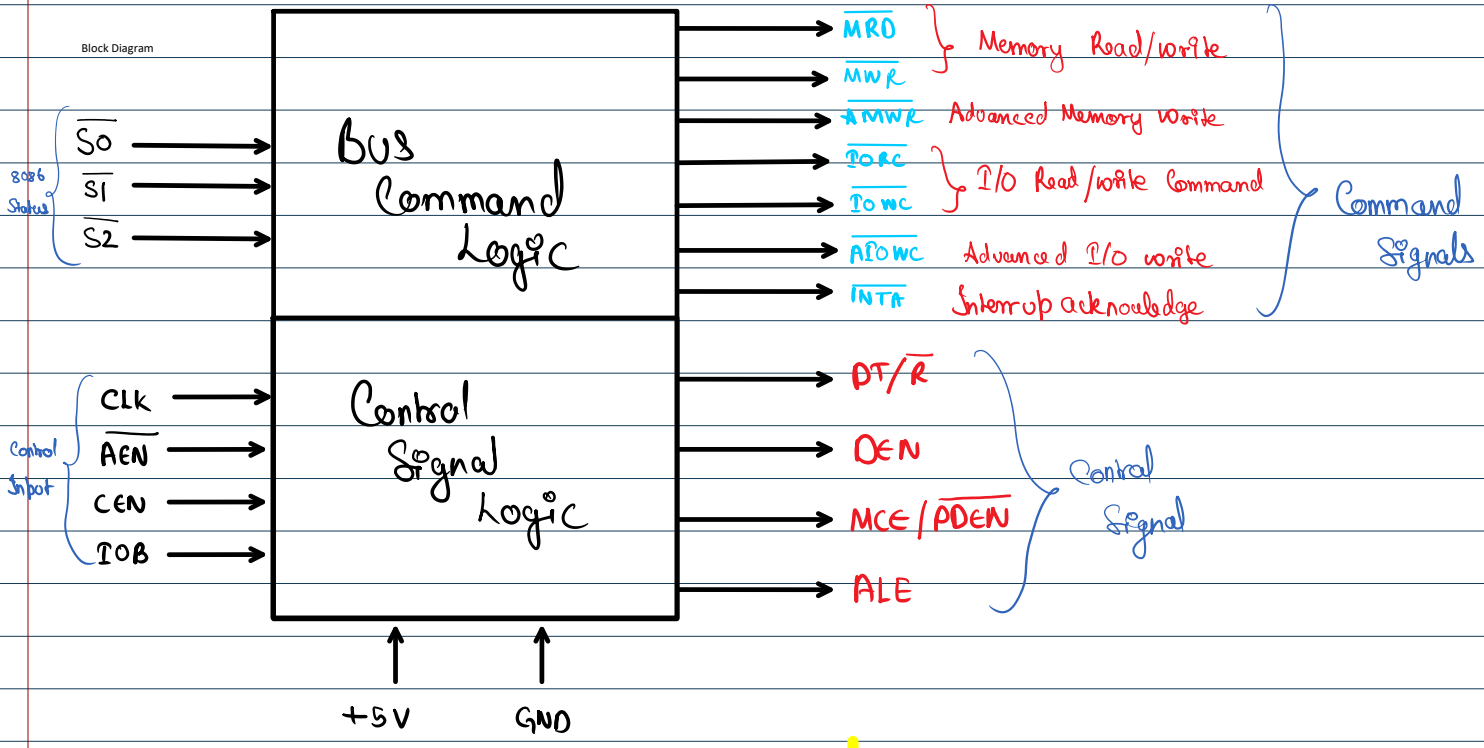
Interrupt Instructions

These are used to call Interrupt during program Execution

; INT, INTO , IRET

Bus Controller 8288

During Maximum Mode, There aren't enough pins on 8086 for bus control, thus required additional IC 8288 for controlling bus signals to memory & I/O



CLK ; Clock ; for Internal timing

$\overline{AEN}$; Address enable ; if 0, then Control signal gets enabled

CEN ; Control Enable ; if 1, then Comman signal gets enabled along with Control signal ($\overline{AEN}$ = 0)

IOB; I/O Bus mode ; Selects either I/O Bus mode or System bus mode

if IOB = 1 ⇒ Bus mode
& PDEN = 0
if IOB = 0 ⇒ System Bus Mode
& MCE = 1

ALE ; Address Latch enable

DEN ; Data bus enable

DT/$\overline{R}$; Data transmit / recive

MCE/PDEN ; Master Cascade enable / peripheral data enable

Status Signal			Command Signal Generated
$\overline{S2}$	$\overline{S1}$	$\overline{S0}$	
0	0	0	$\overline{INTA}$
0	0	1	$\overline{IORD}$
0	1	0	$\overline{IOWR}$ / $\overline{AIOWR}$
0	1	1	HALT STATE
1	0	0	OPCODE fetch
1	0	1	$\overline{MRD}$
1	1	0	$\overline{MWR}$ / $\overline{AMWR}$
1	1	1	Passive

It's a 20-pin chip

Clock Generator (8284)

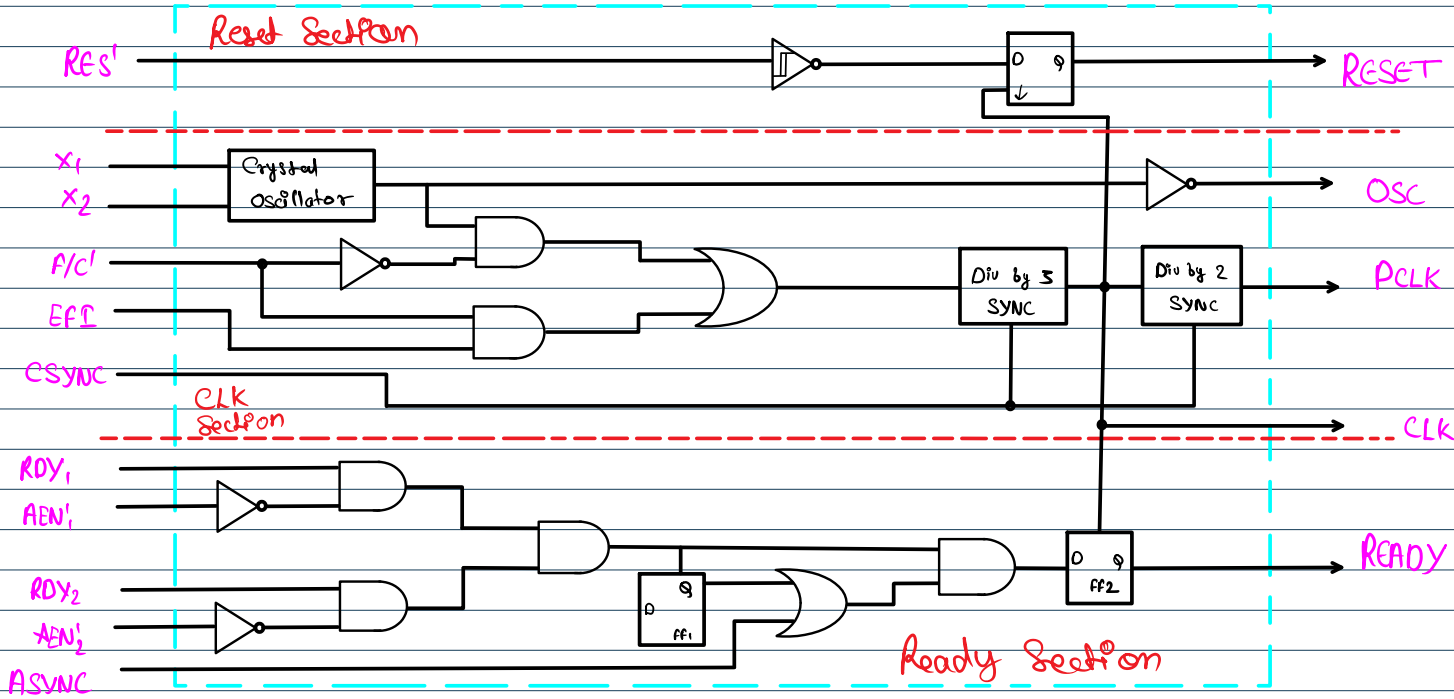
Clock generator provides clock signal, Ready & reset signal to processor

It is a 18-pin chip

Functions ;

- Provide stable clock
- for multiprocessor system, facilitate synchronization of multiple clock
- provide resetting to processor

Logical Circuit



- OSC, CLOCK, PCLK are the three outputs generated by the clock section

The crystal oscillator generates Square waves (frequency dependent on the crystal)

↓

This is fed to AND & NOT to generate OSC

- F/C' ; Frequency/crystal selection pin
- EFI ; External frequency Input

- CSYNC allows synchronization between multiple 8284's
  The clock signal is divided by 2 for peripheral devices
  The reset section is composed of Schmitt's trigger & flip flop

DMA Controller

DMA (Direct Memory Access)

It is a hardware device that directly accesses memory with less participation of processor

It communicates with CPU via data bus & Control lines

Through use of address bus & allowing the DMA & RS register to select inputs, the register within DMA is chosen by CPU

RD & WR : two way inputs

BG : Bus grant ; when 0, CPU relenquishes the buses & DMA directly communicates with the memory

DMA Controller Registers

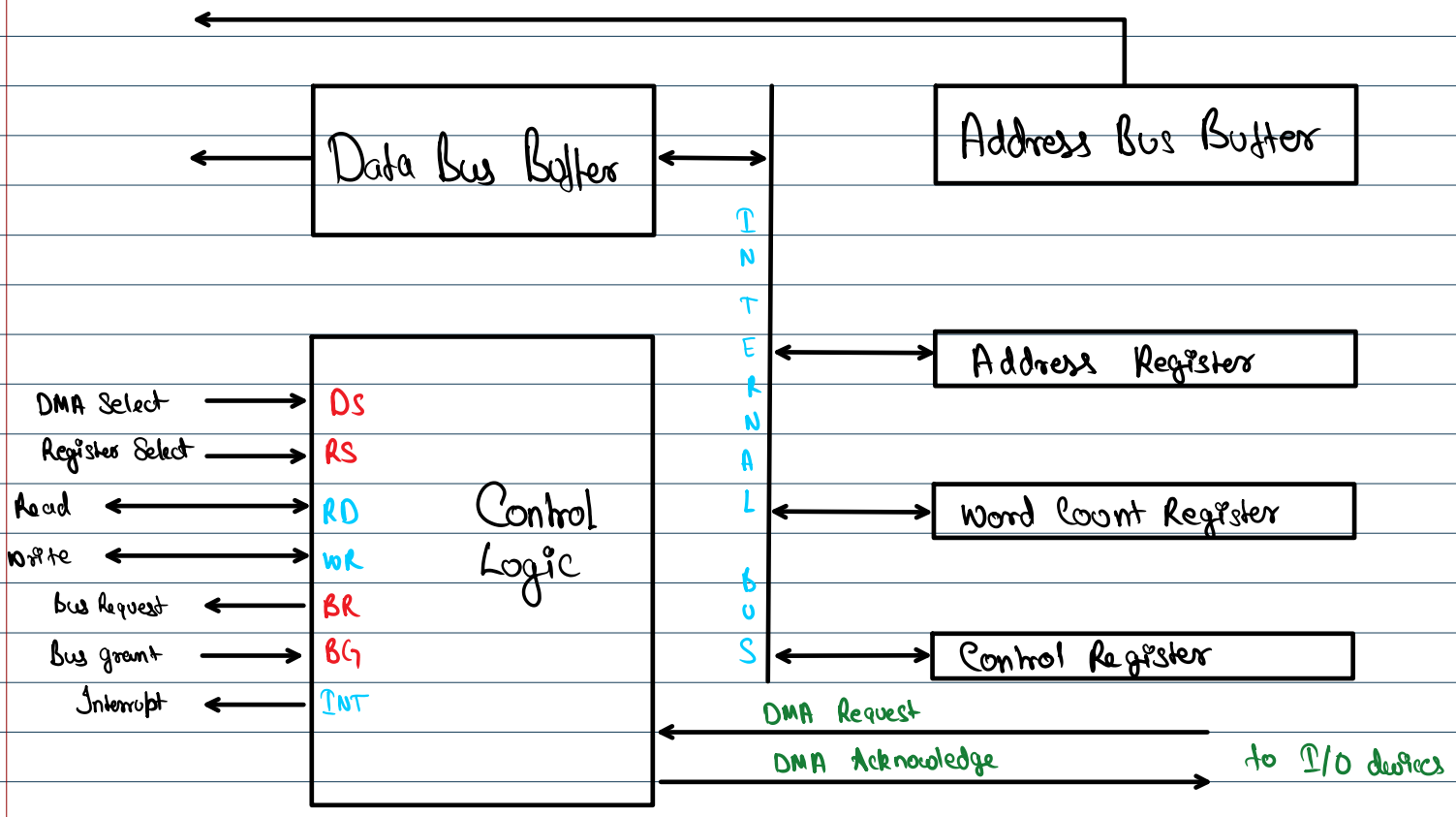
① Address Register : Contains address to specify the desired location in memory

② Word Count Registers : It contains the number of words to be transferred

③ Control Register : Specifies transfer mode

Note

- All registers of DMA appear to CPU as I/O interface registers, thus the CPU can read/write onto DMA registers under program control via databus



Working :

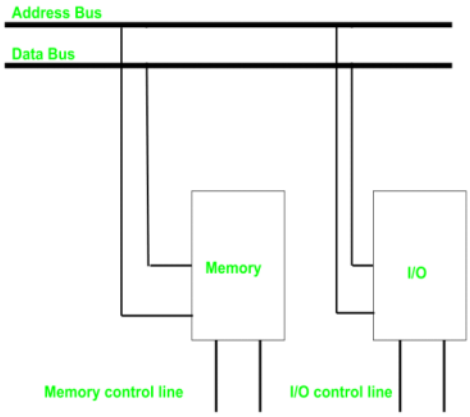
- CPU initializes the DMA by sending given information
  - → Starting address of Memory block where data is available (to read) or to where data is to be stored (to write)
  - → It also sends word count which is number of words in memory block to read/write
  - → Control to define mode of transfer (read/write)
  - → A control to begin DMA transfer.

Mapping I/O

Acp needs to communicate to various memory & I/O devices data flow;

Isolated I/O

- Common bus (data & address) for memory & I/O
- Seperate Read write control lines
- Address space for memory & I/O is isolated



- I/O addresses are called ports
- Complex logic

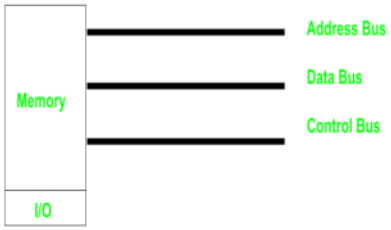
Memory-Mapped I/O

- In this case every bus is common

↓

Same set of instruction work for I/O & Memory

- I/O & Memory have same address space
- Addressing capability of Memory becomes less, as some part is occupied by I/O



- Smaller size
- Less efficient

- Simple logic, I/O located as memory