MICRO PROCESSORS AND INTERFACING

V Semester – IT IARE-R18

A.Y: 2020-2021

Course Code – AECB55 (Open Elective-1)

INSTITUTE OF AERONAUTICAL ENGINEERING

Prepared By

B. Lakshmi Prasanna

SYLLABUS

MODULE-I	INTRODUCTION TO 8 BIT AND 16 BIT MICROPROCESSOR
MODULE-II	OPERATION OF 8086 AND INTERRUPTS.
MODULE-III	INTERFACING WITH 8086.
MODULE-IV	ADVANCED MICRO PROCESSORS
MODULE-V	8051 MICROCONTROLLER ARCHITECTURE

COURSE OUTCOMES

CO1	Outline the internal architecture of 8085, 8086 and 8051 microcomputers to study the functionality.
CO2	Illustrate the organization of registers and memory in 8086 for programming and memory allocation within processor.
CO3	Explain various addressing modes and instruction set of target microprocessor and microcontroller useful for writing assembly language programs.
CO4	Distinguish between minimum mode and maximum mode operation of 8086 microprocessor with timing diagrams.
CO5	Interpret the functionality of various types of interrupts and their structure for controlling the processor or controller and program execution flow.

COURSE OUTCOMES

CO6	Demonstrate the internal architecture and various modes of operation of the devices used for interfacing memory and I/O devices with microprocessor.
CO7	Choose an appropriate data transfer scheme and hardware to perform serial data transfer among the devices.
CO8	Outline the salient features of 80286, 80386 and RISC processors in relation to basic 8086 microprocessor.
CO9	Illustrate the paging operation and segmentation of advanced microprocessors for memory management.
CO10	Interpret the internal building blocks and registers of 8051 microcontroller used to perform serial data transfer, timer operation, interfacing of memory and I/O devices.
CO11	Build necessary hardware and software interface using microcomputer based systems to provide solution for real world problems.

MODULE-I

Introduction to 8 bit and 16 bit Microprocessor

COURSE OUTCOMES

CO1	Outline the internal architecture of 8085, 8086 and 8051 microcomputers to study the functionality.
CO2	Illustrate the organization of registers and memory in 8086 for programming and memory allocation within processor.
CO3	Explain various addressing modes and instruction set of target microprocessor and microcontroller useful for writing assembly language programs.

An over view of 8085

Introduction to processor:

- A processor is the logic circuitry that responds to and processes the basic instructions that drives a computer.

- The term processor has generally replaced the term central processing unit . The processor in a personal computer or embedded in small devices is often called a microprocessor.

- The processor (CPU, for Central Processing Unit) is the computer's brain. It allows the processing of numeric data, meaning information entered in binary form, and the execution of instructions stored in memory.

Evolution of Microprocessor:

- Microprocessor is a program-controlled device, which fetches the instructions from memory, decodes and executes the instructions. Most Micro Processor are single- chip devices.
- Microprocessor is a backbone of computer system. which is called CPU
- Microprocessor speed depends on the processing speed depends on DATABUS WIDTH.
- A common way of categorizing microprocessors is by the no. of bits that their ALU can Work with at a time

- The address bus is unidirectional because the address information is always given by the Micro Processor to address a memory location of an input / output devices.
- The data bus is Bi-directional because the same bus is used for transfer of data between Micro Processor and memory or input / output devices in both the direction.
- It has limitations on the size of data. Most Microprocessor does not support floating-point operations.
- Microprocessor contain ROM chip because it contain instructions to execute data.
- Storage capacity is limited. It has a volatile memory. In secondary storage device the storage capacity is larger. It is a nonvolatile memory.

Primary devices are: RAM (Read / Write memory, High Speed, Volatile Memory) / ROM (Read only memory, Low Speed, Non Voliate Memory)

Secondary devices are: Floppy disc / Hard disk

Compiler:

Compiler is used to translate the high-level language program into machine code at a time. It doesn’t require special instruction to store in a memory, it stores automatically. The Execution time is less compared to Interpreter

8085 MICROPROCESSOR

- It is an 8-bit microprocessor designed by Intel in 1977.

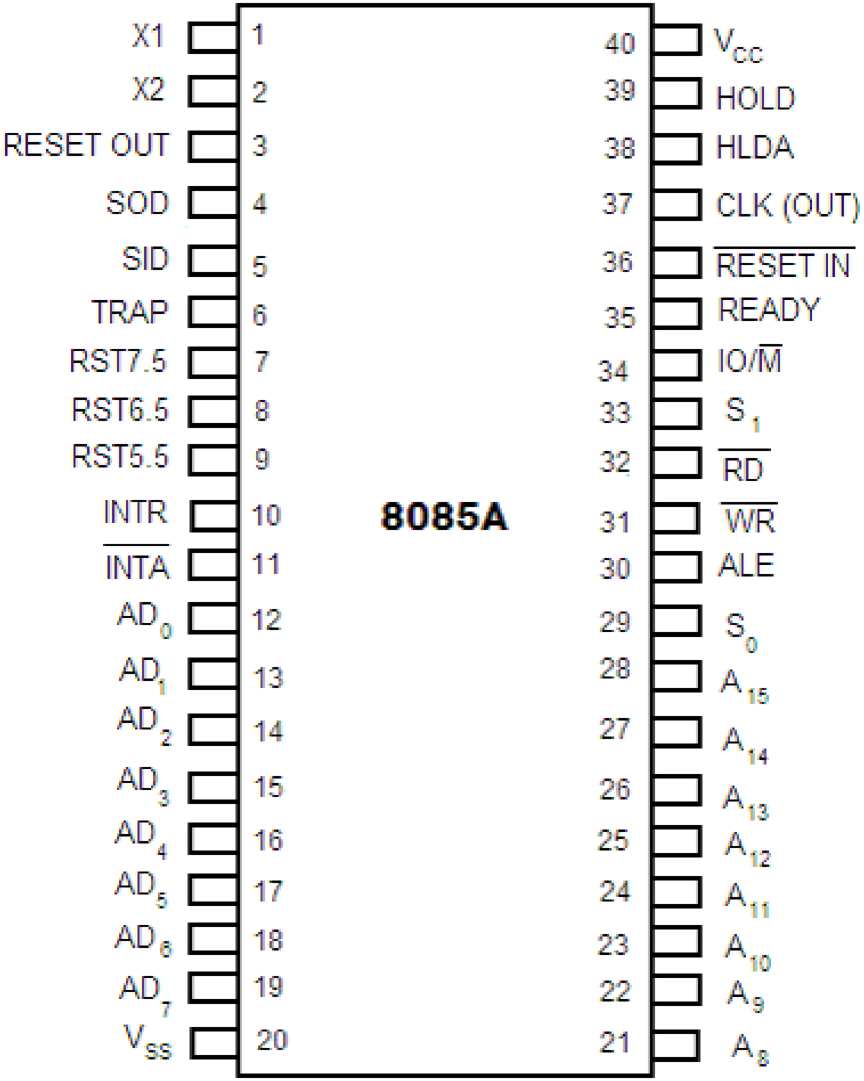
It has the following configuration –(FEATURES)

- 8-bit data bus
- 16-bit address bus, which can address up to 64KB
- A 16-bit program counter
- A 16-bit stack pointer
- Six 8-bit registers arranged in pairs: BC, DE, HL
- Requires +5V supply to operate.

It is a single chip with 40 pins.

- It has multiplexed address and data bus(AD0 - AD7)
- The maximum clock frequency is 3 MHz while minimum frequency is 500 KHz.
- It provides 74 instruction with 5 different addressing modes.

Pin Diagram of 8085



PIN DESCRIPTION

Address bus

A15-A8, it carries the most significant 8-bits of memory/IO address.

Data bus

AD7-AD0, it carries the least significant 8-bit address and data bus.

Power supply

There are 2 power supply signals VCC & VSS. VCC indicates +5v power supply and VSS indicates ground signal.

Control and status signals

There are 3 control signal and 3 status signals.

Three control signals are RD’, WR’ & ALE.

RD’ − When it is enabled, CPU reads the data available on data bus send by memory or I/O device.

WR’ – When it is enabled ,CPU write the data on to the data bus from memory or I/O device .

ALE – It is a multiplexed signal .When the pulse goes high, it indicates address. When the pulse goes down it indicates data.

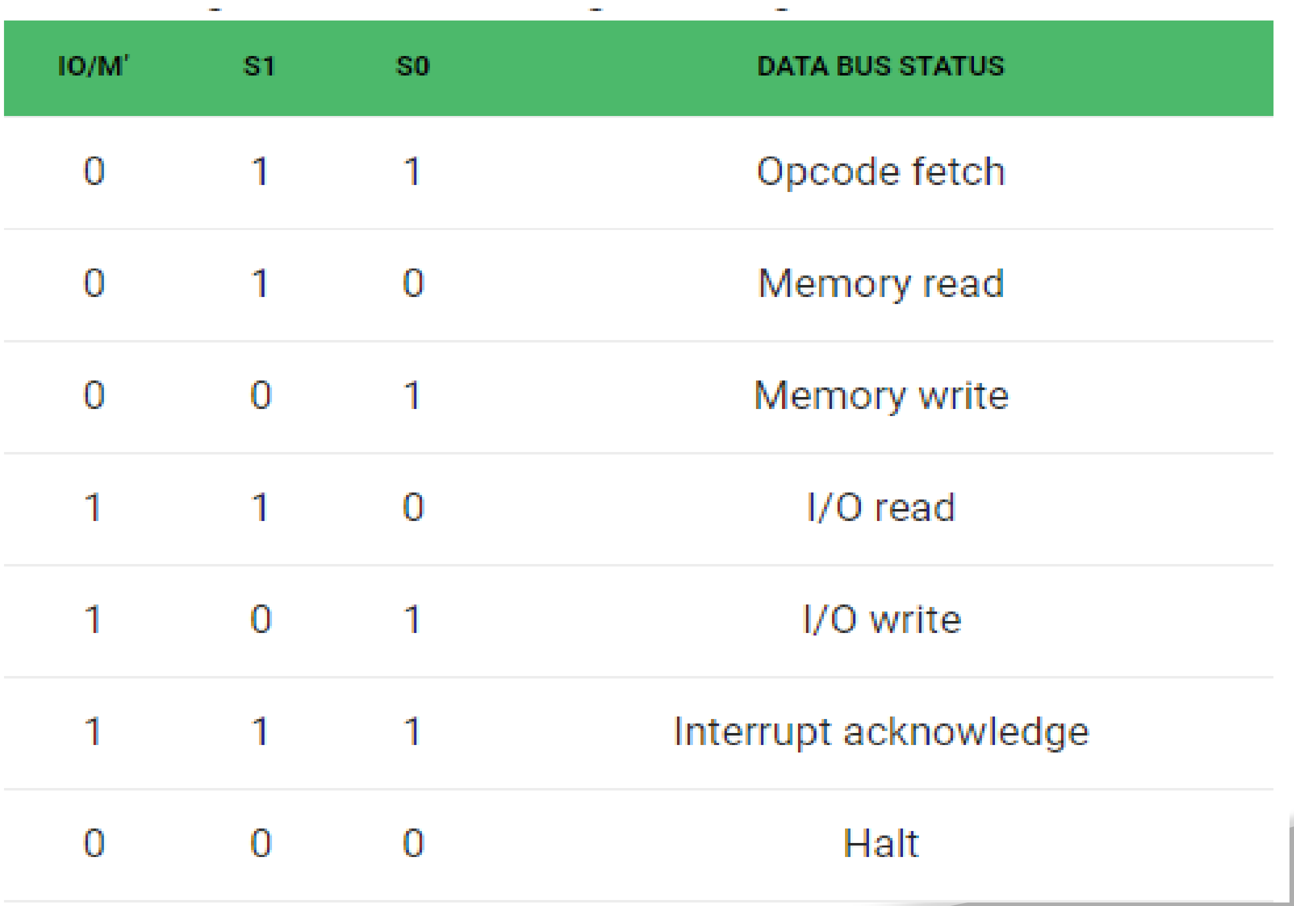
Three status signals are IO/M’, S0 & S1.

IO/M’

This signal is used to differentiate between IO and Memory operations, i.e. when it is high indicates IO operation and when it is low then it indicates memory operation.

S1 & S0

These signals are used to identify the type of current operation.



IO/M'	S1	S0	DATA BUS STATUS
0	1	1	Opcode fetch
0	1	0	Memory read
0	0	1	Memory write
1	1	0	I/O read
1	0	1	I/O write
1	1	1	Interrupt acknowledge
0	0	0	Halt

Clock signals

There are 3 clock signals, i.e. X1, X2, CLK OUT.

X1, X2 − A crystal (RC, LC N/W) is connected at these two pins and is used to set frequency of the internal clock generator. This frequency is internally divided by 2.

CLK OUT − This signal is used as the system clock for devices connected with the microprocessor.

Interrupts & externally initiated signals

Interrupts are the signals generated by external devices to request the microprocessor to perform a task. There are 5 interrupt signals, i.e. TRAP, RST 7.5, RST 6.5, RST 5.5, and INTR.

INTA’ − It is an interrupt acknowledgment signal.

RESET IN’ − This signal is used to reset the microprocessor by setting the program counter to zero.

RESET OUT − This signal is used to reset all the connected devices when the microprocessor is reset.

READY − This signal indicates that the device is ready to send or receive data. If READY is low, then the CPU has to wait for READY to go high.

HOLD − This signal indicates that another master is requesting the use of the address and data buses.

HLDA (HOLD Acknowledge) − It indicates that the CPU has received the HOLD request and it will gives the bus in the next clock cycle. HLDA is set to low after the HOLD signal is removed.

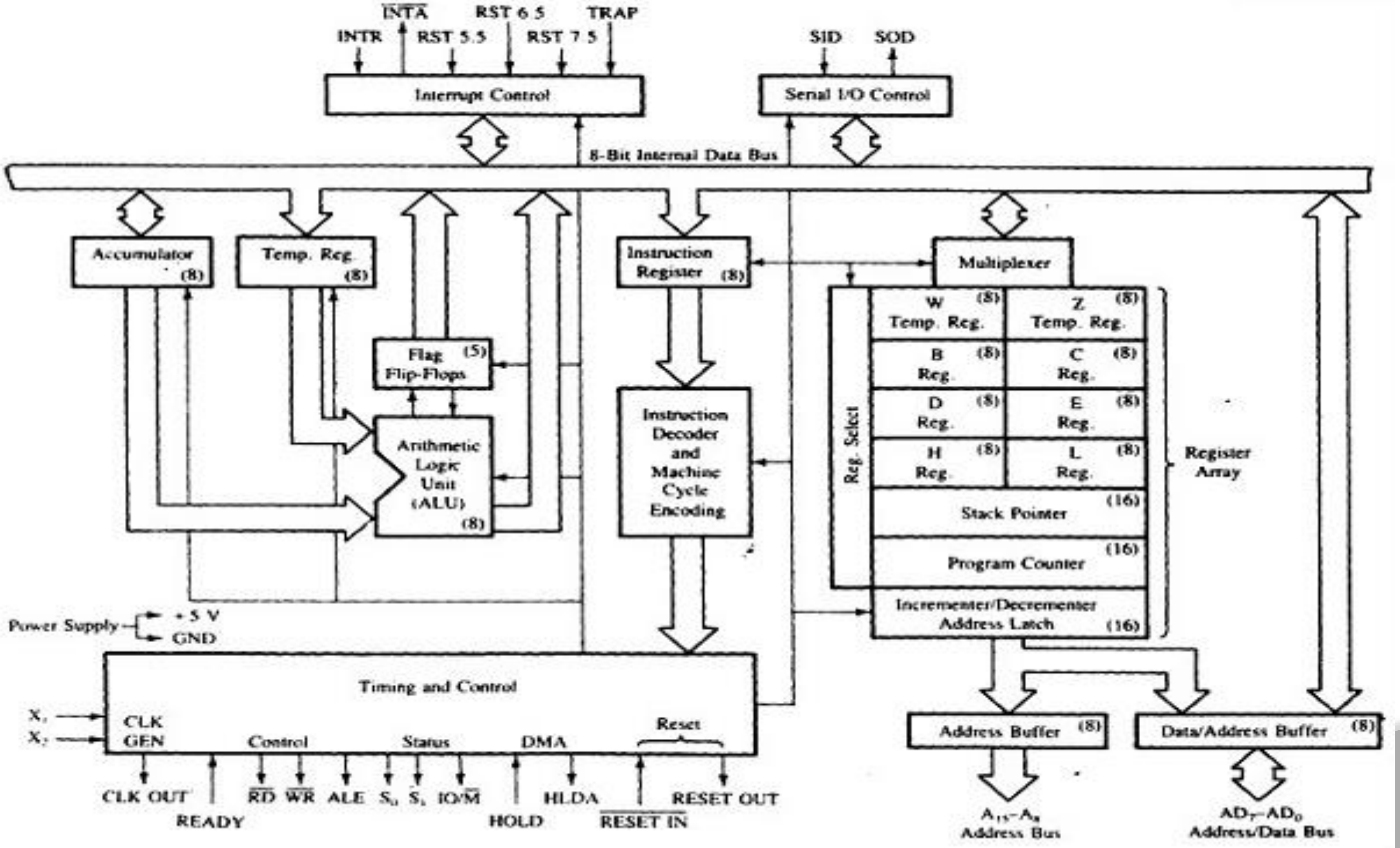
Serial I/O signals

There are 2 serial signals, i.e. SID and SOD and these signals are used for serial communication.

SOD (Serial output data line) − The output SOD is set/reset as specified by the SIM instruction.

SID (Serial input data line) − The data on this line is loaded into accumulator whenever a RIM instruction is executed.

8085 ARCHITECTURE



8085 Microprocessor – Functional Units

8085 consists of the following functional units −

Accumulator

It is an 8-bit register used to perform arithmetic, logical, I/O & LOAD/STORE operations. It is connected to internal data bus & ALU.

Arithmetic and logic unit

As the name suggests, it performs arithmetic and logical operations like Addition, Subtraction, AND, OR, etc. on 8-bit data.

General purpose register

There are 6 general purpose registers in 8085 processor, i.e. B, C, D, E, H & L. Each register can hold 8-bit data.

These registers can work in pair to hold 16-bit data and their pairing combination is like B-C, D-E & H-L.

Program counter

It is a 16-bit register used to store the memory address location of the next instruction to be executed. Microprocessor increments the program counter whenever an instruction is being executed, so that the program counter points to the memory address of the next instruction that is going to be executed.

Stack pointer

It is also a 16-bit register works like stack, which is always incremented/decremented by 2 during push & pop operations.

Temporary register

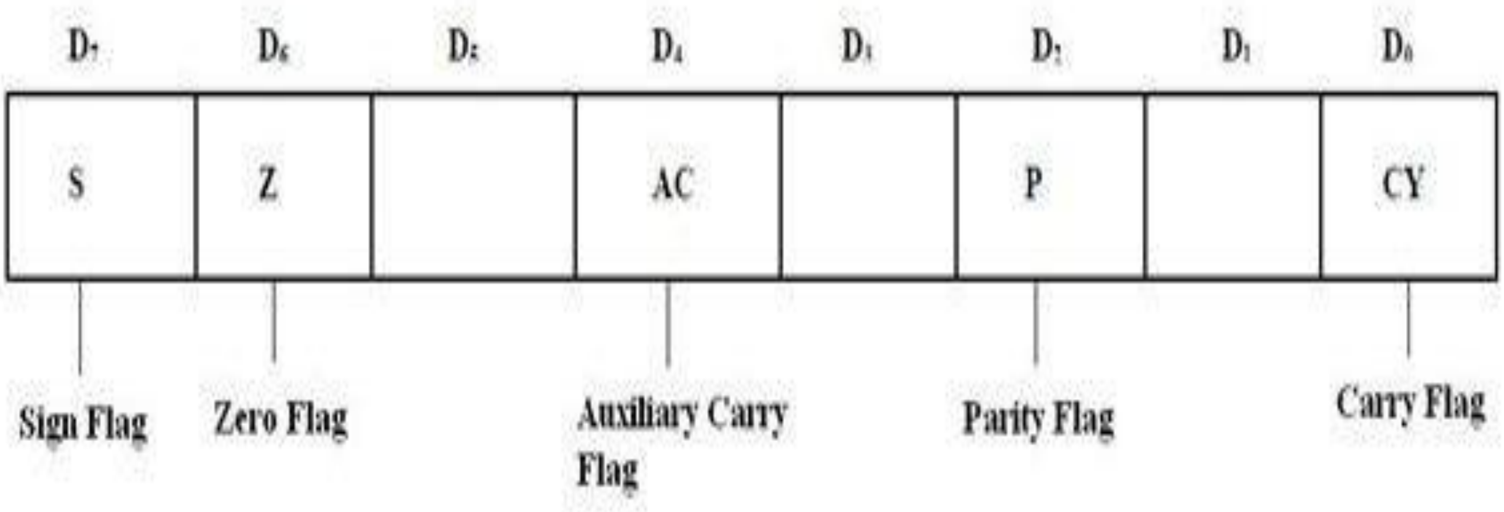
It is an 8-bit register, which holds the temporary data of arithmetic and logical operations.

Flag register

It is an 8-bit register having five 1-bit flip-flops, which holds either 0 or 1 depending upon the result stored in the accumulator.

These are the set of 5 flip-flops –

i. Sign (S)
ii. Zero (Z)
iii. Auxiliary Carry (AC)
iv. Parity (P)
v. Carry (C)



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

Instruction register and Instruction decoder

It is an 8-bit register. When an instruction is fetched from memory then it is stored in the Instruction register. Instruction decoder decodes the information present in the Instruction register.

Timing and control unit

It provides timing and control signal to the microprocessor to perform operations. Following are the timing and control signals, which control external and internal circuits –

Control Signals: READY, RD’, WR’, ALE

Status Signals: S0, S1, IO/M’

DMA Signals: HOLD, HLDA

RESET Signals: RESET IN’, RESET OUT

Interrupt control

As the name suggests it controls the interrupts during a process. When a microprocessor is executing a main program and whenever an interrupt occurs, the microprocessor shifts the control from the main program to process the incoming request. After the request is completed, the control goes back to the main program.

There are 5 interrupt signals in 8085 microprocessor: INTR, RST 7.5, RST 6.5, RST 5.5, TRAP.

Serial Input/output control

It controls the serial data communication by using these two instructions: SID (Serial input data) and SOD (Serial output data).

Address buffer and address-data buffer

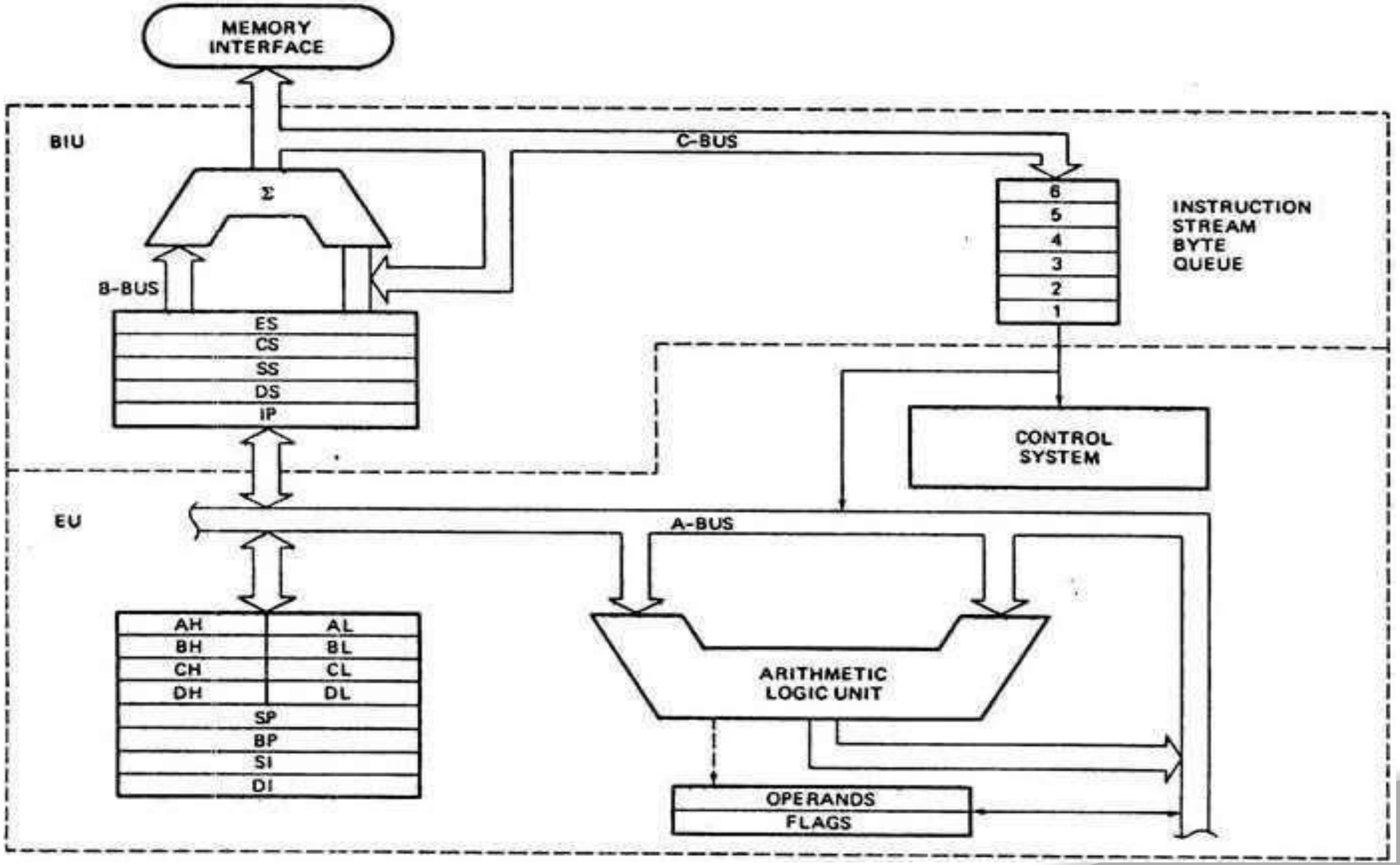
The content stored in the stack pointer and program counter is loaded into the address buffer and address-data buffer to communicate with the CPU. The memory and I/O chips are connected to these buses; the CPU can exchange the desired data with the memory and I/O chips.

Address bus and data bus

Data bus carries the data to be stored. It is bidirectional, whereas address bus carries the location to where it should be stored and it is unidirectional. It is used to transfer the data & Address I/O devices.

Architecture of 8086 microprocessor

Architecture :



- 8086 Microprocessor is divided into two functional units, i.e., EU(Execution Unit) and BIU (Bus Interface Unit).

EU (Execution Unit):

- Execution unit gives instructions to BIU stating from where to fetch the data and then decode and execute those instructions.
- Its function is to control operations on data using the instruction decoder & ALU.
- EU has no direct connection with system buses as shown in the above figure, it performs operations over data through BIU.

- **BIU(Bus Interface Unit):**

  BIU takes care of all data and addresses transfers on the buses for the EU like sending addresses, fetching instructions from the memory, reading data from the ports and the memory as well as writing data to the ports and the memory. EU has no direction connection with System Buses so this is possible with the BIU. EU and BIU are connected with the Internal Bus.

Instruction queue

- BIU contains the instruction queue. BIU gets up to 6 bytes of next instructions and stores them in the instruction queue. When EU executes instructions and is ready for its next instruction, then it simply reads the instruction from this instruction queue resulting in increased execution speed.

- **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.

Register organization of 8086

AX & DX registers:

- In 8 bit multiplication, one of the operands must be in AL. The other operand can be a byte in memory location or in another 8 bit register. The resulting 16 bit product is stored in AX, with AH storing the MS byte.
- In 16 bit multiplication, one of the operands must be in AX. The other operand can be a word in memory location or in another 16 bit register. The resulting 32 bit product is stored in DX and AX, with DX storing the MS word and AX storing the LS word.

BX register :

- In instructions where we need to specify in a general purpose register the 16 bit effective address of a memory location, the register BX is used (register indirect).

CX register :

- In Loop Instructions, CX register will be always used as the implied counter. In I/O instructions, the 8086 receives into or sends out data from AX or AL depending as a word or byte operation.
- In these instructions the port address, if greater than FFH has to be given as the contents of DX register.
- Ex : IN AL, DX
  DX register will have 16 bit address of the I/P device

- **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.

8086 flag register

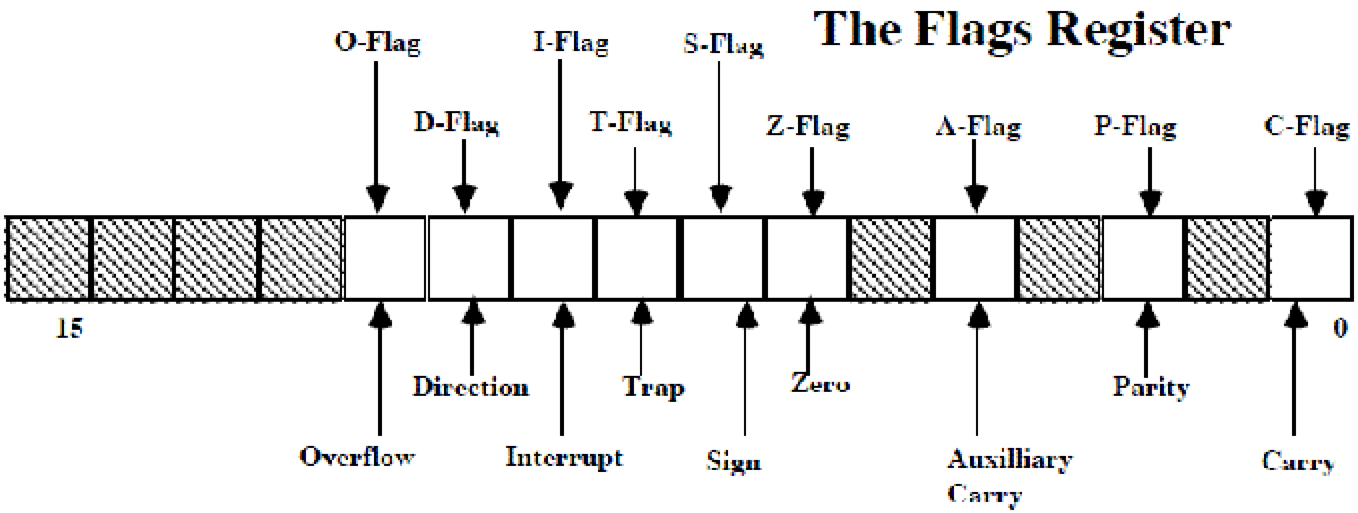
Flag Register

- Flag Register contains a group of status bits called flags that indicate the status of the CPU or the result of arithmetic operations.
- There are two types of flags:
- The status flags which reflect the result of executing an instruction. The programmer cannot set/reset these flags directly.
- The control flags enable or disable certain CPU operations. The programmer can set/reset these bits to control the CPU's operation.

- Nine individual bits of the status register are used as control flags (3 of them) and status flags (6 of them).The remaining 7 are not used.

- A flag can only take on the values 0 and 1. We say a flag is set if it has the value 1.The status flags are used to record specific characteristics of arithmetic and of logical instructions.

8086 flag register



- Control Flags: There are three control flags

- The Direction Flag (D): Affects the direction of moving data blocks by such instructions as MOVS, CMPS and SCAS. The flag values are 0 = up and 1 = down and can be set/reset by the STD (set D) and CLD (clear D) instructions.

- The Interrupt Flag (I): Dictates whether or not system interrupts can occur. Interrupts are actions initiated by hardware block such as input devices that will interrupt the normal execution of programs. The flag values are 0 = disable interrupts or 1 = enable interrupts and can be manipulated by the CLI (clear I) and STI (set I) instructions.

- The Trap Flag (T): Determines whether or not the CPU is halted after the execution of each instruction. When this flag is set (i.e. = 1), the programmer can single step through his program to debug any errors. When this flag = 0 this feature is off. This flag can be set by the INT 3 instruction.
- Status Flags: There are six status flags

- The Carry Flag (C): This flag is set when the result of an unsigned arithmetic operation is too large to fit in the destination register. This happens when there is an end carry in an addition operation or there an end borrows in a subtraction operation. A value of 1
= carry and 0 = no carry.

- The Overflow Flag (O): This flag is set when the result of a signed arithmetic operation is too large to fit in the destination register (i.e. when an overflow occurs). Overflow can occur when adding two numbers with the same sign (i.e. both positive or both negative). A value of 1 = overflow and 0 = no overflow.

- The Sign Flag (S): This flag is set when the result of an arithmetic or logic operation is negative. This flag is a copy of the MSB of the result (i.e. the sign bit). A value of 1 means negative and 0 = positive.

- The Zero Flag (Z): This flag is set when the result of an arithmetic or logic operation is equal to zero. A value of 1 means the result is zero and a value of 0 means the result is not zero.

- The Auxiliary Carry Flag (A): This flag is set when an operation causes a carry from bit 3 to bit 4 (or a borrow from bit 4 to bit 3) of an operand. A value of 1 = carry and 0 = no carry.

- The Parity Flag (P): This flags reflects the number of 1s in the result of an operation. If the number of 1s is even its value = 1 and if the number of 1s is odd then its value = 0.

Addressing Modes of 8086

Addressing Modes

Addressing Modes of 8086:

- Addressing mode indicates a way of locating data or operands. Depending up on the data type used in the instruction and the memory addressing modes, any instruction may belong to one or more addressing modes or same instruction may not belong to any of the addressing modes.

- The addressing mode describes the types of operands and the way they are accessed for executing an instruction. According to the flow of instruction execution, the instructions may be categorized as

- **Sequential control flow instructions and**
- **Control transfer instructions.**

Addressing Modes

- Sequential control flow instructions are the instructions which after execution, transfer control to the next instruction appearing immediately after it (in the sequence) in the program. For example the arithmetic, logic, data transfer and processor control instructions are Sequential control flow instructions.

- The control transfer instructions on the other hand transfer control to some predefined address or the address somehow specified in the instruction, after their execution. For example INT, CALL, RET & JUMP instructions fall under this category.

Addressing Modes

- The addressing modes for Sequential and control flow instructions are explained as follows.

- Immediate addressing mode:
- In this type of addressing, immediate data is a part of instruction,
  and appears in the form of successive byte or bytes.

Example: MOV AX, 0005H.

- In the above example, 0005H is the immediate data . The immediate data may be 8- bit or 16-bit in size.

Addressing Modes

Direct addressing mode:

- In the direct addressing mode, a 16-bit memory address (offset)
  directly specified in the instruction as a part of it.

  Example: MOV AX, [5000H].

Register addressing mode:

- In the register addressing mode, the data is stored in a register and it is referred using the particular register. All the registers, except IP, may be used in this mode.

  Example: MOV BX, AX

Addressing Modes

Register indirect addressing mode:

- Sometimes, the address of the memory location which contains data or operands is determined in an indirect way, using the offset registers. The mode of addressing is known as register indirect mode.

- In this addressing mode, the offset address of data is in either BX or SI or DI Register. The default segment is either DS or ES.

  Example: MOV AX, [BX].

Addressing Modes

- Indexed addressing mode:
- In this addressing mode, offset of the operand is stored one of the index registers. DS & ES are the default segments for index registers SI & DI respectively.

  Example: MOV AX, [SI]
- Here, data is available at an offset address stored in SI in DS.

- Register relative addressing mode:
- In this addressing mode, the data is available at an effective address formed by adding an 8-bit or 16-bit displacement with the content of any one of the register BX, BP, SI & DI in the default (either in DS & ES) segment.

  Example: MOV AX, 50H [BX]

Addressing Modes

- Based indexed addressing mode:
- The effective address of data is formed in this addressing mode, by adding content of a base register (any one of BX or BP) to the content of an index register (any one of SI or DI). The default segment register may be ES or DS.

  Example: MOV AX, [BX][SI]

- Relative based indexed:
- The effective address is formed by adding an 8 or 16-bit displacement with the sum of contents of any of the base registers (BX or BP) and any one of the index registers, in a default segment.

  Example: MOV AX, 50H [BX] [SI]

Addressing Modes

- Addressing Modes for control transfer instructions:

- Intersegment
  - Intersegment direct
  - Intersegment indirect

- Intrasegment
  - Intrasegment direct
  - Intrasegment indirect

Addressing Modes

- **Intersegment direct:**

- In this mode, the address to which the control is to be transferred is in a different segment. This addressing mode provides a means of branching from one code segment to another code segment. Here, the CS and IP of the destination address are specified directly in the instruction.

Example: JMP 5000H: 2000H;

- Jump to effective address 2000H in segment 5000H.

Addressing Modes

- **Intersegment indirect:**
- In this mode, the address to which the control is to be transferred lies in a different segment and it is passed to the instruction indirectly, i.e. contents of a memory block containing four bytes,
  i.e. IP(LSB), IP(MSB), CS(LSB) and CS(MSB) sequentially. The starting address of the memory block may be referred using any of the addressing modes, except immediate mode.

- Example: JMP [2000H].

Jump to an address in the other segment specified at effective address 2000H in DS.

Addressing Modes

- **Intrasegment direct mode:**

- In this mode, the address to which the control is to be transferred lies in the same segment in which the control transfers instruction lies and appears directly in the instruction as an immediate displacement value. In this addressing mode, the displacement is computed relative to the content of the instruction pointer.

Addressing Modes

- The effective address to which the control will be transferred is given by the sum of 8 or 16 bit displacement and current content of IP. In case of jump instruction, if the signed displacement (d) is of 8-bits (i.e. -128<d<+127), it as short jump and if it is of 16 bits (i.e. - 32768<d<+32767), it is termed as long jump.

Example: JMP SHORT LABEL.

Addressing Modes

- **Intrasegment indirect mode:**
- In this mode, the displacement to which the control is to be transferred is in the same segment in which the control transfer instruction lies, but it is passed to the instruction directly. Here, the branch address is found as the content of a register or a memory location.
- This addressing mode may be used in unconditional branch
  instructions.
- Example: JMP [BX]; Jump to effective address stored in BX.

Instruction set of 8086

INSTRUCTION SET OF 8086

- *The Instruction set of 8086 microprocessor* is classified into 7 Types, they are:-
  - Data transfer instructions
  - Arithmetic& logical instructions
  - Program control transfer instructions
  - Machine Control Instructions
  - Shift / rotate instructions
  - Flag manipulation instructions
  - String instructions

Data Transfer instructions

- Data transfer instruction, as the name suggests is for the transfer of data from memory to internal register, from internal register to memory, from one register to another register, from input port to internal register, from internal register to output port etc

MOV instruction

- It is a general purpose instruction to transfer byte or word from register to register, memory to register, register to memory or with immediate addressing.

- General Form:
- MOV destination, source
- Here the source and destination needs to be of the same size,
  that is both 8 bit or both 16 bit.
- MOV instruction does not affect any flags.

Example:-

- MOV BX, 00F2H; load the immediate number 00F2H in BX register

- MOV [589H], BX;

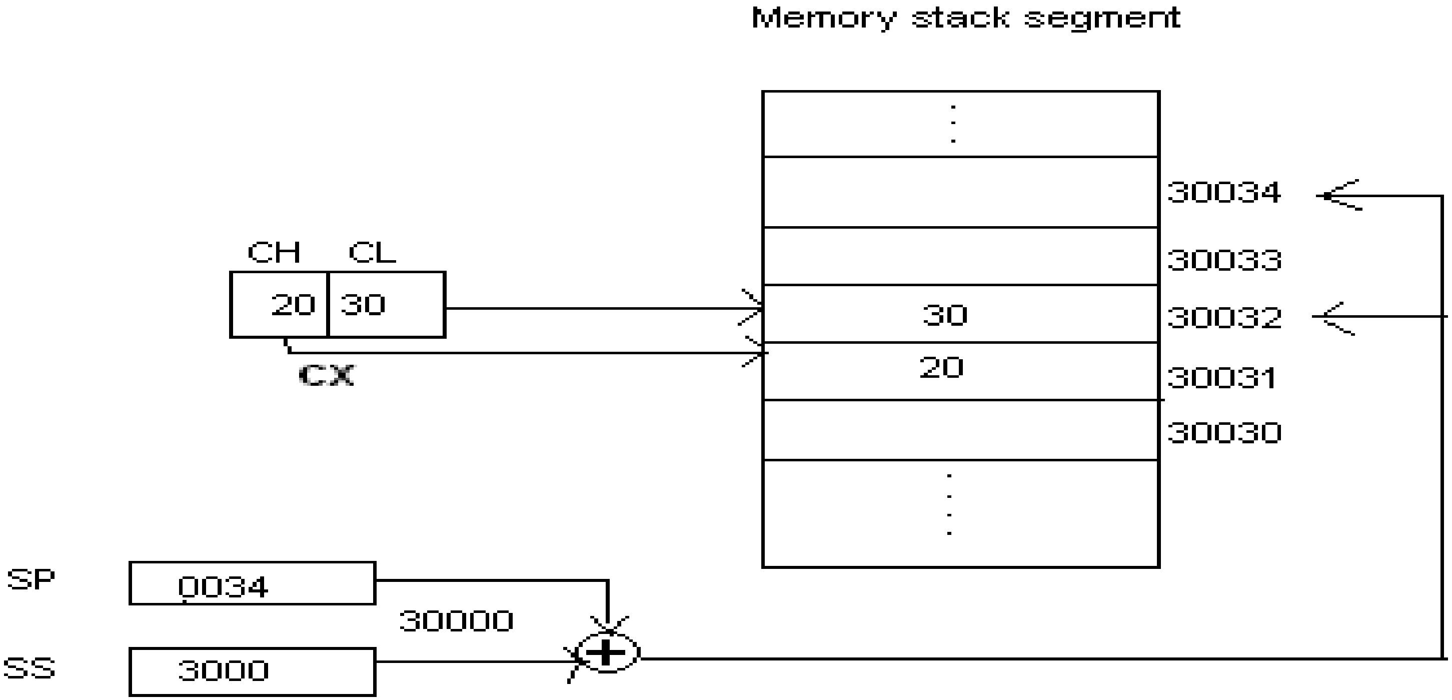
Copy the 16 bit content of BX register on to the memory location,
which at a displacementof 589H from the data segment base.

- MOV DS, CX;Move the content of CX to DS

PUSH instruction

- The PUSH instruction decrements the stack pointer by two and
  copies the word from source to the location where stack pointer now points. Here the source must of word size data. Source can be a general purpose register, segment register or a memory location.

The PUSH instruction first pushes the most significant byte to sp-1, then the least significant to the sp-2. Push instruction does not affect any flags.



Example:-

- PUSH CX ; Decrements SP by 2, copy content of CX to the stack

- **POP instruction**

The POP instruction copies a word from the stack location pointed by the stack pointer to the destination. The destination can be a General purpose register, a segment register or a memory location. Here after the content is copied the stack pointer is automatically incremented by two.

- The execution pattern is similar to that of the PUSH instruction. Example: POP CX; Copy a word from the top of the stack to CX and increment SP by 2.

- **IN & OUT instructions**
- The IN instruction will copy data from a port to the accumulator. If 8 bit is read the data will go to AL andif 16 bit then to AX. Similarly OUT instruction is used to copy data from accumulator to an output port.
- Both IN and OUT instructions can be done using direct and indirect addressing modes.

  Example:
- IN AL, 0F8H; Copy a byte from the port 0F8H to AL
- MOV DX, 30F8H;Copy port address in DX
- IN AL, DX; **Move 8 bit data from 30F8H port**
- IN AX, DX; **Move 16 bit data from 30F8H port**
- OUT 047H, AL; **Copy contents of AL to 8 bit port 047H**
- MOV DX, 30F8H;Copy port address in DX

XCHG instruction

- The XCHG instruction exchanges contents of the destination and source. Here destination and source can be register and register or register and memory location, but XCHG cannot interchange the value of 2 memory locations.

General Format

- XCHG Destination, Source

  Example:

- XCHG BX, CX; exchange word in CX with the word inBX
- XCHG AL, CL; exchange byte in CL with the byte in AL
- XCHG AX, SUM[BX];here physical address, which isDS+SUM+[BX]. The content at physical address and the content of AX are interchanged.

Instruction set of 8086
(Arithmetic Instructions in 8086)

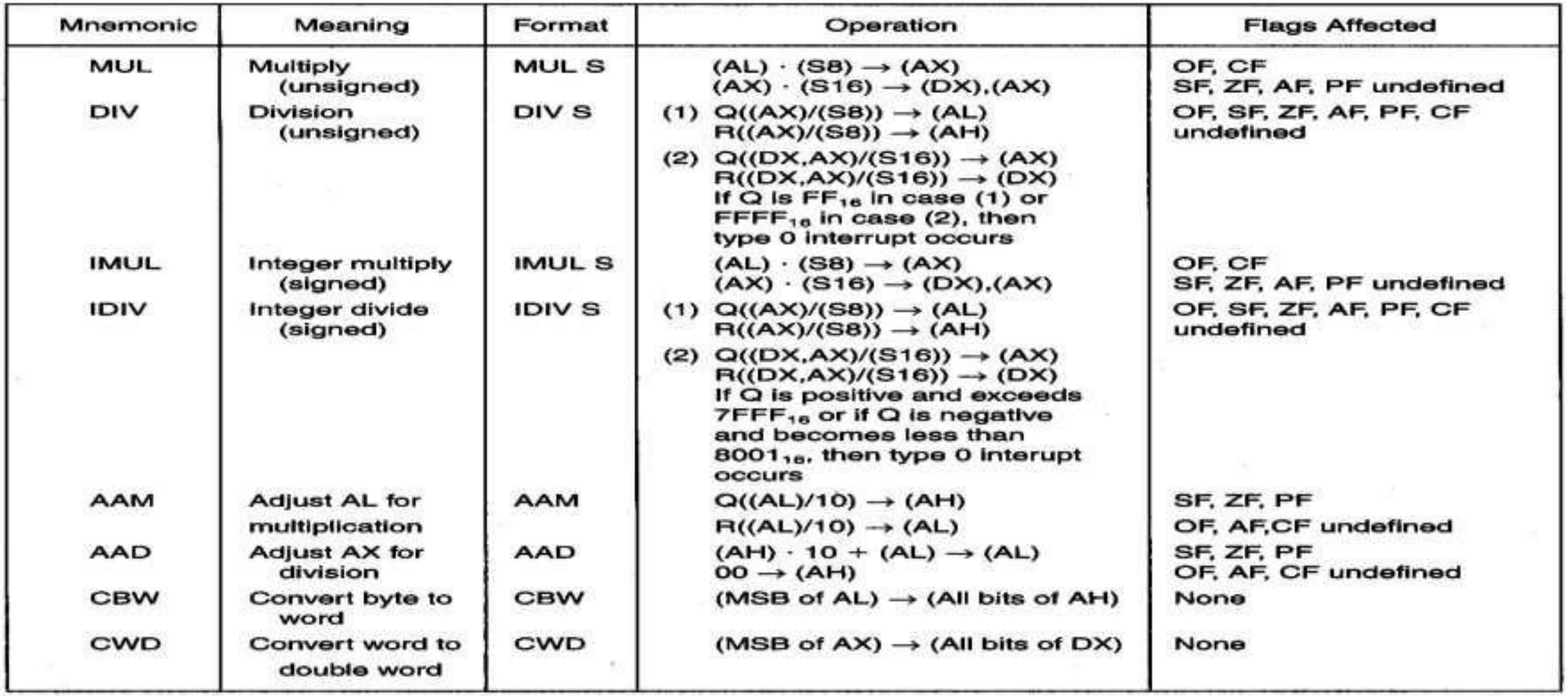
Arithmetic Instructions: ADD, ADC, INC, AAA, DAA

Mnemonic	Meaning	Format	Operation	Flags affected
ADD	Addition	ADD D,S	(S)+(D) → (D) carry → (CF)	ALL
ADC	Add with carry	ADC D,S	(S)+(D)+(CF) → (D) carry → (CF)	ALL
INC	Incrementby one	INC D	(D)+1 → (D)	ALL but CY
AAA	ASCII adjust for addition	AAA	If the sum is >9, AH is incremented by 1	AF,CF
DAA	Decimal adjust for addition	DAA	Adjust AL for decimal Packed BCD	ALL

Arithmetic Instructions–SUB, SBB, DEC, AAS, DAS, NEG

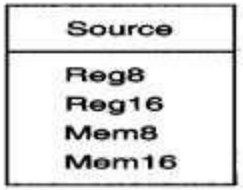
Mnemonic	Meaning	Format	Operation	Flags affected
SUB	Subtract	SUB D,S	(D) - (S) → (D) Borrow → (CF)	All
SBB	Subtract with borrow	SBB D,S	(D) - (S) - (CF) → (D)	All
DEC	Decrement by one	DEC D	(D) - 1 → (D)	All but CF
NEG	Negate	NEG D		All
DAS	Decimal adjust for subtraction	DAS	Convert the result in AL to packed decimal format	All
AAS	ASCII adjust for subtraction	AAS	(AL) difference (AH) dec by 1 if borrow	CY,AC

Multiplication and Division



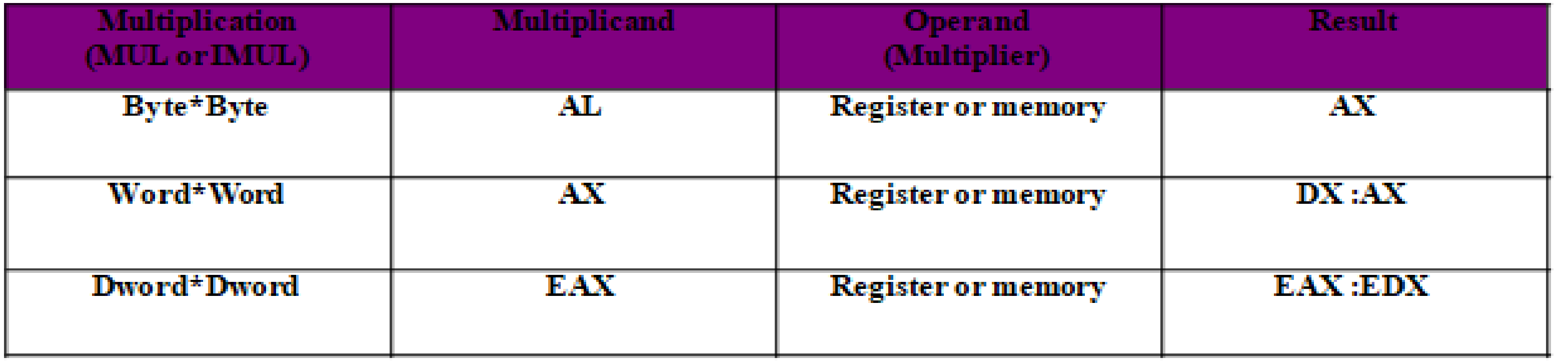
Mnemonic	Meaning	Format	Operation	Flags Affected
MUL	Multiply (unsigned)	MUL S	$(AL) \cdot (S8) \rightarrow (AX)$ $(AX) \cdot (S16) \rightarrow (DX),(AX)$	OF, CF SF, ZF, AF, PF undefined
DIV	Division (unsigned)	DIV S	(1) $Q((AX)/(S8)) \rightarrow (AL)$ $R((AX)/(S8)) \rightarrow (AH)$ (2) $Q((DX,AX)/(S16)) \rightarrow (AX)$ $R((DX,AX)/(S16)) \rightarrow (DX)$ If Q is FF_{16} in case (1) or $FFFF_{16}$ in case (2), then type 0 interrupt occurs	OF, SF, ZF, AF, PF, CF undefined
IMUL	Integer multiply (signed)	IMUL S	$(AL) \cdot (S8) \rightarrow (AX)$ $(AX) \cdot (S16) \rightarrow (DX),(AX)$	OF, CF SF, ZF, AF, PF undefined
IDIV	Integer divide (signed)	IDIV S	(1) $Q((AX)/(S8)) \rightarrow (AL)$ $R((AX)/(S8)) \rightarrow (AH)$ (2) $Q((DX,AX)/(S16)) \rightarrow (AX)$ $R((DX,AX)/(S16)) \rightarrow (DX)$ If Q is positive and exceeds $7FFF_{16}$ or if Q is negative and becomes less than 8001_{16}, then type 0 interupt occurs	OF, SF, ZF, AF, PF, CF undefined
AAM	Adjust AL for multiplication	AAM	$Q((AL)/10) \rightarrow (AH)$ $R((AL)/10) \rightarrow (AL)$	SF, ZF, PF OF, AF,CF undefined
AAD	Adjust AX for division	AAD	$(AH) \cdot 10 + (AL) \rightarrow (AL)$ $00 \rightarrow (AH)$	SF, ZF, PF OF, AF, CF undefined
CBW	Convert byte to word	CBW	(MSB of AL) → (All bits of AH)	None
CWD	Convert word to double word	CWD	(MSB of AX) → (All bits of DX)	None

(a)

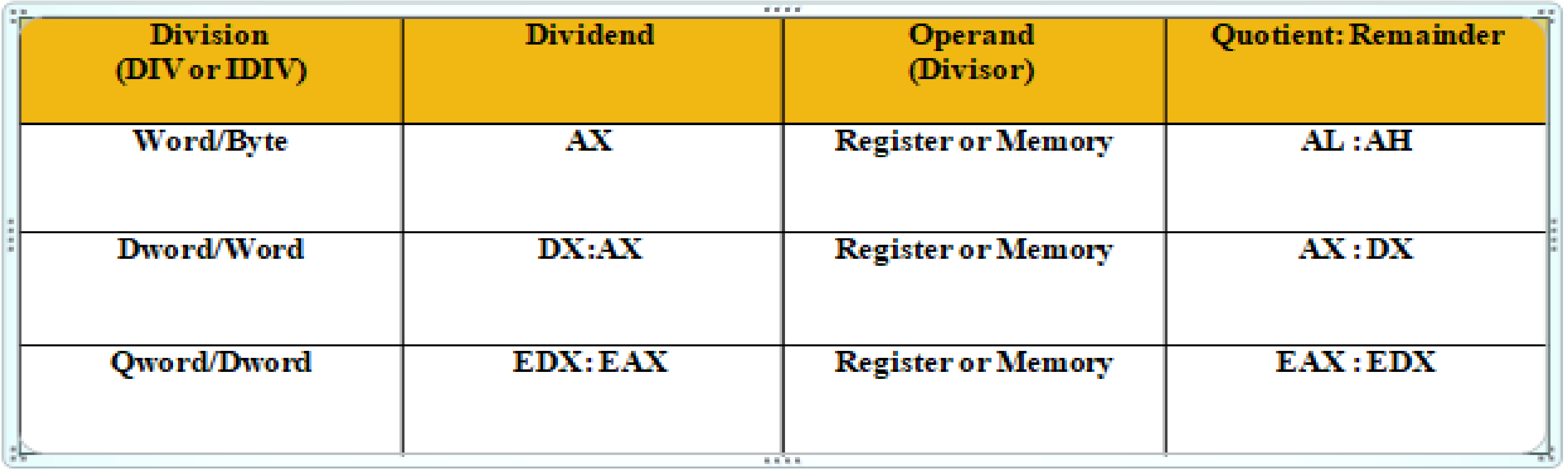


Source
Reg8
Reg16
Mem8
Mem16

(b)



Multiplication (MUL or IMUL)	Multiplicand	Operand (Multiplier)	Result
Byte*Byte	AL	Register or memory	AX
Word*Word	AX	Register or memory	DX :AX
Dword*Dword	EAX	Register or memory	EAX :EDX



Division (DIV or IDIV)	Dividend	Operand (Divisor)	Quotient: Remainder
Word/Byte	AX	Register or Memory	AL : AH
Dword/Word	DX:AX	Register or Memory	AX : DX
Qword/Dword	EDX: EAX	Register or Memory	EAX : EDX

Instruction set of 8086 (Logical Instructions in 8086)

AND instruction

- This instruction logically ANDs each bit of the source byte/word with the corresponding bit in the destination and stores the result in destination. The source can be an immediate number, register or memory location, register can be a register or memory location.
- The CF and OF flags are both made zero, PF, ZF, SF are affected by the operation and AF is undefined.

- General Format:
- AND Destination, Source

Example:

- AND BL, AL ;suppose BL=1000 0110 and AL = 1100 1010 then after the operation BL would be BL= 1000 0010.
- AND CX, AX ;CX <= CX AND AX
- AND CL, 08 ;CL<= CL AND (0000 1000)

OR instruction

- This instruction logically ORs each bit of the source byte/word with the corresponding bit in the destination and stores the result in destination. The source can be an immediate number, register or memory location, register can be a register or memory location.

- The CF and OF flags are both made zero, PF, ZF, SF are affected by the operation and AF is undefined.

- General Format:
- OR Destination, Source

Example:

- OR BL, AL; suppose BL=1000 0110 and AL = 1100 1010 then after the operation BL would be BL= 1100 1110.
- OR CX, AX;CX <= CX AND AX
- OR CL, 08;CL<= CL AND (0000 1000)

NOT instruction

- The NOT instruction complements (inverts) the contents of an operand register or a memory location, bit by bit. The examples are as follows:

Example:

- NOT AX (BEFORE AX= (1011)2= (B) 16 AFTER EXECUTION AX= (0100)2= (4)16).
- NOT [5000H]

XOR instruction

- The XOR operation is again carried out in a similar way to the AND and OR operation. The constraints on the operands are also similar. The XOR operation gives a high output, when the 2 input bits are dissimilar. Otherwise, the output is zero. The example instructions are as follows:

  Example:

  - XOR AX,0098H
  - XOR AX,BX
  - XOR AX,[5000H]

- **Shift / Rotate Instructions**
- Shift instructions move the binary data to the left or right by shifting them within the register or memory location. They also can perform multiplication of powers of 2+n and division of powers of 2-n.
- There are two type of shifts logical shifting and arithmetic shifting, later is used with signed numbers while former with unsigned.

- **SHL/SAL instruction**
- Both the instruction shifts each bit to left, and places the MSB in CF and LSB is made 0. The destination can be of byte size or of word size, also it can be a register or a memory location. Number of shifts is indicated by the count.
- All flags are affected.
- General Format:
- SAL/SHL destination, count

  Example:
- MOV BL, B7H;
- BL is made B7HSAL BL, 1;
- shift the content of BL register one place to left.
- Before execution,
- CY B7,B6 B5 B4 B3 B2 B1 B0

- **SHR instruction**
- This instruction shifts each bit in the specified destination to the right and 0 is stored in the MSB position. The LSB is shifted into the carry flag. The destination can be of byte size or of word size, also it can be a register or a memory location. Number of shifts is indicated by the count.
- All flags are affected
- General Format: SHR destination, count

  Example:
- MOV BL, B7H;BL is made B7H
- SHR BL, 1;shift the content of BL register one place to the right.
- Before execution,

  B7 B6 B5 B4 B3 B2 B1 B0 CY

- After execution,
- B7 B6 B5 B4 B3 B2 B1 B0 CY
- ROL instruction
- This instruction rotates all the bits in a specified byte or word to the left some number of bit positions. MSB is placed as a new LSB and a new CF. The destination can be of byte size or of word size, also it can be a register or a memory location. Number of shifts is indicated by the count.
- All flags are affected

- General Format: ROL destination, count

Example:

- MOV BL, B7H;BL is made B7H
- CY   B7 B6 B5 B4 B3 B2 B1 B0
- ROL BL, 1;rotates the content of BL register one place to the left.

Before execution,

- CY   B7 B6 B5 B4 B3 B2 B1 B0

- **ROR instruction**
- This instruction rotates all the bits in a specified byte or word to the right some number of bit positions. LSB is placed as a new MSB and a new CF. The destination can be of byte size or of word size, also it can be a register or a memory location. Number of shifts is indicated by the count.
- General Format: ROR destination,

  count Example:
- MOV BL, B7H; BL is made B7H
- ROR BL, 1;shift the content of BL register one place to the right.
- Before execution,
- B7 B6 B5 B4 B3 B2 B1 B0 CY

- **RCR instruction**
- This instruction rotates all the bits in a specified byte or word to the right some number of bit positions along with the carry flag. LSB is placed in a new CF and previous carry is placed in the new MSB. The destination can be of byte size or of word size, also it can be a register or a memory location. Number of shifts is indicated by the count.

- All flags are affected
- General Format: RCR destination, count

  Example:
- MOV BL, B7H;BL is made B7H
- RCR BL, 1;shift the content of BL register one place to the right.

instruction set of 8086 (String Instructions)

String Instruction Basics

- Source DS:SI, Destination ES:DI
  - You must ensure DS and ES are correct
  - You must ensure SI and DI are offsets into DS and ES
    respectively
- Direction Flag (0 = Up, 1 = Down)
  - CLD - Increment addresses (left to right)
  - STD - Decrement addresses (right to

String Control Instructions

1) MOVS/ MOVSB/ MOVSW

Dest string name, src stringname

This instruction moves data byte or word from location in DS
to location in ES.

2) REP / REPE / REPZ / REPNE / REPNZ

Repeat string instructions until specified conditions exist.

This is prefix a instruction.

String ControlInstructions

4)SCAS / SCASB / SCASW

Scan a string byte or string word.

Compares byte in AL or word in AX. String address is to be loaded in DI.

5)STOS / STOSB / STOSW

Store byte or word in a string.

Copies a byte or word in AL or AX to memory location pointed by DI.

6)LODS / LODSB /LODSW

Load a byte or word in AL or AX

- Copies byte or word from memory location pointed by SI into AL or AX register.

5. Program Execution TransferInstructions

- instructions are similar to branching or looping instructions. These instructions include unconditional jump or loop instructions.
- Classification:
- Unconditional transfer instructions
- Conditional transfer instructions
- Iteration control instructions
- Interrupt instructions

Unconditional transferinstructions

- CALL: Call a procedure, save return address onstack

- RET: Return from procedure to the mainprogram.

- JMP: Goto specified address to get next instruction CALL

instruction: The CALL instruction is used to transfer

execution   of program to a subprogram or procedure.

CALL instruction

- Near call

1. Direct Near CALL: The destination address is specified in the instruction itself.
2. Indirect Near CALL: The destination address is specified in any 16-bit register, except IP.

- Far call

1. Direct Far CALL: The destination address is specified in the instruction itself. It will be in different Code Segment.
2. Indirect Far CALL: The destination address is specified in twoword memory locations pointed by a register.

JMP instruction

The processor jumps to the specified location rather than the

instruction after the JMP instruction.

- Intra segment jump
- Inter segment jump

RET

RET instruction will return execution from a procedure to The next instruction after the CALL instruction in the calling program.

Conditional Transfer Instructions

- JA/JNBE: Jump if above / jump if not below or equal
- JAE/JNB: Jump if above /jump if notbelow
- JBE/JNA: Jump if below or equal/ Jump if not above
- JC: jump if carry flag CF=1
- JE/JZ: jump if equal/jump if zero flag ZF=1
- JG/JNLE: Jump if greater/ jump if not less than or equal.

Conditional Transfer Instructions

- JGE/JNL: jump if greater than or equal/ jump if not less than
- JL/JNGE: jump if less than/ jump if not greater than or equal
- JLE/JNG: jump if less than or equal/ jump if not greater than
- JNC: jump if no carry (CF=0).
- JNE/JNZ: jump if not equal/ jump if not zero(ZF=0)

Conditional Transfer Instructions

- JNO: jump if no overflow(OF=0)
- JNP/JPO: jump if not parity/ jump if parity odd(PF=0)
- JNS: jump if not sign(SF=0)
- JO: jump if overflow flag(OF=1)
- JP/JPE: jump if parity/jump if parityeven(PF=1)
- JS: jump if sign(SF=1).

Iteration Control Instructions

- These instructions are used to execute a series of instructions for certain number of times.
- LOOP: Loop through a sequence of instructions until CX=0.
- LOOPE/LOOPZ : Loop through a sequence of **instructions while** ZF=1 and instructions CX = 0.
- LOOPNE/LOOPNZ : Loop through a sequence of instructions while ZF=0 and CX =0.
- JCXZ : jump to specified

Interrupt Instructions

Two types of interrupt instructions:

- Hardware Interrupts (External Interrupts)
- Software Interrupts (Internal Interrupts and Instructions)

Hardware Interrupts:

- INTR is a maskable hardware interrupt.
- NMI is a non-maskable interrupt.

Software Interrupts

- INT : Interrupt program execution, call serviceprocedure
- INTO : Interrupt program execution if OF=1
- IRET: Return from interrupt service procedure to main program.

High Level Language Interface Instructions

- ENTER : enter procedure.
- LEAVE: Leave procedure.
- BOUND: Check if effective address within specified array bounds.

Processor ControlInstructions

I. Flag set/clearinstructions

- STC: Set carry flag CF to 1
- CLC: Clear carry flag CF to0
- CMC: Complement the state of the carry flagCF
- STD: Set direction flag DF to 1 (decrement stringpointers)
- CLD: Clear direction flag DF to0
- STI: Set interrupt enable flag to 1(enable INTRinput)
- CLI: Clear interrupt enable Flag to 0 (disable INTRinput)

II. External Hardware synchronizationinstructions

- HLT: Halt (do nothing) until interrupt or reset.
- WAIT: Wait (Do nothing) until signal on the test pin islow.
- ESC: Escape to external coprocessor such as 8087 or 8089.
- LOCK: An instruction prefix. Prevents another processor from taking the bus while the adjacent instruction executes.
- NOP: No operation. This instruction simply takes up three clock cycles and does no processing.

Assembler Directives

Assembler Directives

➢ **ASSUME**		
➢ **DB**	**-**	Defined Byte.
➢ **DD**	**-**	Defined Double Word
➢ **DQ**	**-**	Defined Quad Word
➢ **DT**	**-**	Define Ten Bytes
➢ **DW**	**-**	Define Word

- **ASSUME Directive**- The ASSUME directive is used to tell the assembler that the name of the logical segment should be used for a specified segment. The 8086 works directly with only 4 physical segments: a Code segment, a data segment, a stack segment, and an extra segment.

Example:

ASUME CS:CODE ;This tells the assembler that the logical segment named CODE contains the instruction statements for the program and should be treated as a code segment.

ASSUME DS:DATA ;This tells the assembler that for any instruction which refers to a data in the data segment, data will found in the logical segment DATA.

- **DB** - DB directive is used to declare a byte- type variable orto store a byte in memory location.
- **Example:**

1. **PRICE DB 49h, 98h, 29h** ;Declare an array of 3 bytes, named as PRICE and initialize.
2. **NAME DB ‘ABCDEF’** ;Declare an array of 6 bytes and initialize with ASCII code for letters
3. **TEMP DB 100 DUP(?)** ;Set 100 bytes of storage in memory and give it the name as TEMP, but leave the 100 bytes uninitialized. Program instructions will load values into these locations.

- DW-The DW directive is used to define a variable of type word or to reserve storage location of type word in memory.

- Example:

- MULTIPLIER DW 437Ah ; this declares a variable of type word and named it as MULTIPLIER. This variable is initialized with the value 437Ah when it is loaded into memory to run.

- EXP1 DW 1234h, 3456h, 5678h ; this declares an array of 3 words and initialized with specified values.

- STOR1 DW 100 DUP(0); Reserve an array of 100 words of memory and initialize all words with 0000.Array is named as STOR1.

- END-END directive is placed after the last statement of a program to tell the assembler that this is the end of the program module. The assembler will ignore any statement after an END directive.

- ENDP-ENDP directive is used along with the name of the procedure to indicate the end of a procedure to the assembler

Example:

- SQUARE_NUM PROCE ; It start the procedure ;Some steps to find the square root of a number
- SQUARE_NUM ENDP ;Hear it is the End for the procedure

END		End Program
ENDP	-	End Procedure
ENDS	-	End Segment
EQU		Equ-ate
EVEN	-	Align on Even Memory Address
EXTRN	-	

- ENDS - This ENDS directive is used with name of the segment to
  indicate the end of that logic segment.

  Example: CODE SEGMENT ;Hear it Start the logic segment
  containing code ;

- CODE ENDS ;End of segment named as CODE

- GLOBAL - Can be used in place of a PUBLIC directive or in place of an EXTRN directive.

- GROUP-Used to tell the assembler to group the logical statements named after the directive into one logical group segment, allowing the contents of all the segments to be accessed from the same group segment base.

- INCLUDE - Used to tell the assembler to insert a block of source code from the named file into the current source module.

- LABEL- Used to give a name to the current value in the location counter.
- NAME- Used to give a specific name to each assembly module when programs consisting of several modules are written. E.g.: NAME PC_BOARD

- OFFSET- Used to determine the offset or displacement of a named data item or procedure from the start of the segment which contains it.

  E.g.: MOV BX, OFFSET PRICES

- ORG- The location counter is set to 0000 when the assembler starts reading a segment. The ORG directive allows setting a desired value at any point in the program.

  E.g.: ORG 2000H

- PROC- Used to identify the start of a procedure. E.g.: SMART_DIVIDE PROC FAR

- PTR- Used to assign a specific type to a variable or to a label. E.g.: INCBYTE PTR[BX] tells the

- PUBLIC- Used to tell the assembler that a specified name or label will be accessed from other modules.
- SEGMENT- Used to indicate the start of a logical segment.

  E.g.: CODE SEGMENT indicates to the assembler the start of a logical segment called CODE
- SHORT- Used to tell the assembler that only a 1 byte displacement is needed to code a jump instruction.

  E.g.: JMP SHORT NEARBY_LABEL
- TYPE - Used to tell the assembler to determine the type of a specified variable.

  E.g.: ADD BX, TYPE WORD_ARRAY is used where we want to increment BX to point to the next word in an array of words.

Procedures and Macros

Procedures:

- While writing programs, it may be the case that a particular sequence of instructions is used several times. To avoid writing the sequence of instructions again and again in the program, the same sequence can be written as a separate subprogram called a procedure.

Defining Procedures:

- Assembler provides PROC and ENDP directives in order to define procedures. The directive PROC indicates beginning of a procedure. Its general form is:

  Procedure_name PROC [NEAR|FAR]

Passing parameters to and from procedures:

The data values or addresses passed between procedures and main program are called parameters. There are four ways of passing parameters:

- Passing parameters in registers
- Passing parameters in dedicated memory locations
- Passing parameters with pointers passed in registers
- Passing parameters using the stack

MACROS:

- When the repeated group of instruction is too short or not suitable to be implemented as a procedure, we use a MACRO. A macro is a group of instructions to which a name is given. Each time a macro is called in a program, the assembler will replace the macro name with the group of instructions.

Defining MACROS:

- Before using macros, we have to define them. MACRO directive informs the assembler the beginning of a macro. The general form is:
- Macro_name MACRO argument1, argument2, …
- Arguments are optional. ENDM informs the assembler the end of

  the macro. Its general form is : ENDM

Differences

Procedures	Macros
Accessed by CALL and RET mechanism during program execution	Accessed by name given to macro when defined during assembly
Machine code for instructions only put in memory once	Machine code generated for instructions each time called
Parameters are passed in registers, memory locations or stack	Parameters passed as part of statement which calls macro
Procedures uses stack	Macro does not utilize stack
A procedure can be defined anywhere in program using the directives PROC and ENDP	A macro can be defined anywhere in program using the directives MACRO and ENDM
Procedures takes huge memory for CALL(3 bytes each time CALL is used) instruction	Length of code is very huge if macro's are called for more number of times

Assembly language programs involving logical, Branch & Call instructions

Write an assembly language program for addition of two 8-bit numbers using 8086 microprocessors.

```
DATA SEGMENT
     A1 DB 50H
     A2 DB 51H
     RES DB ?
DATA ENDS
CODE SEGMENT
ASSUME CS: CODE, DS:DATA
START:                  MOV AX,DATA
           MOV DS,AX
           MOV AL,A1
           MOV  BL,A2
           ADD AL,BL
           MOV RES,AL
           MOV AX,4C00H
           INT 21H
CODE ENDS
END START
```

Write an assembly language program to find the factorial of given number using 8086 microprocessors.

```
DATA SEGMENT
     FIRST DW 03H
     SEC DW 01H
DATA ENDS
CODE SEGMENT
ASSUME CS:CODE,DS:DATA
 START:          MOV AX,DATA
          MOV DS,AX
          MOV AX,SEC
          MOV CX,FIRST
       L1:   MUL CX
          DEC CX
          JCXZ L2
          JMP L1
       L2:   INT 3H
CODE ENDS
END START
```

Write an assembly language program to find the sum of squares using 8086 microprocessors.

```
DATA  SEGMENT
      NUM DW 5H
      RES DW ?
DATA ENDS
CODE SEGMENT
ASSUME CS: CODE, DS: DATA
START:   MOV AX,DATA
         MOV DS,AX
         MOV CX,NUM
         MOV BX,00
      L1: MOV AX,CX
           MUL CX
                ADD BX,AX
                DEC CX
                JNZ L1
                MOV RES,BX
                INT 3H
   CODE ENDS
   END START
```

Assembly language programs

Programs using logical ,Branch and call instructions.

```
Data segment
Org 2000h
N1 dw 5678h
N2 dw 2345h
Data ends
Code segment
Assume cs:code,ds:dats
Mov ax,data
Mov ds,ax
Mov DI,2040h
Mov ax,N1
AND ax,bx
Mov [di],ax
Int 03h
Code ends
End
```

Assembly language programs

2)Data segment

```
Org 2000h
N1 dw 5678h
N2 dw 2345h
Data ends
Code segment
Assume cs:code,ds:dats
Mov ax,data
Mov ds,ax
Mov DI,2040h
Mov ax,N1
MOV bx,N2
OR  ax,bx
Mov [di],ax
Int 03h
Code ends
End
```

Assembly language

3)Data segment

- Org 2000h
- N1 dw 5678h
- N2 dw 2345h
- Data ends
- Code segment
- Assume cs:code,ds:dats
- Mov ax,data
- Mov ds,ax
- Mov DI,2040h
- Mov ax,N1
- MOV bx,N2
- xor ax,bx
- Mov [di],ax
- Int 03h
- Code ends
- End

Assembly language programs

4)Data segment

```
Org 2000h
N1 dw 5678h
Data ends
Code segment
Assume cs:code,ds:dats
Mov ax,data
Mov ds,ax
Mov DI,2040h
Mov ax,N1
SHL  ax,04
Mov [di],ax
Int 03h
Code ends
End
```

Assembly language programs

Programs using logical ,Branch and call instructions.

1)Data segment

```
Org 2000h
N1 dw 5678h
Data ends
Code segment
Assume cs:code,ds:dats
Mov ax,data
Mov ds,ax
Mov DI,2040h
Mov ax,N1
SHR ax,04
. Mov [di],ax
. Int 03h
. Code ends
.  End
```

Assembly language programs

2)Data segment

- **Org 2000h**
- **N1 dw 5678h**
- **Data ends**
- **Code segment**
- **Assume cs:code,ds:dats**
- **Mov ax,data**
- **Mov ds,ax**
- **Mov DI,2040h**
- **Mov ax,N1**
- **ROR ax,02**
- **Mov [di],ax**
- **Int 03h**
- **Code ends**
- **End**

Assembly language

3)Data segment

```
Org 2000h
N1 dw 5678h
Data ends
Code segment
Assume cs:code,ds:dats
Mov ax,data
Mov ds,ax
Mov DI,2040h
Mov ax,N1
RCR ax,03
Mov [di],ax
Int 03h
Code ends
End
```

Assembly language programs

4)Data segment

- **Org 2000h**
- **N1 dw 5678h**
- **Data ends**
- **Code segment**
- **Assume cs:code,ds:dats**
- **Mov ax,data**
- **Mov ds,ax**
- **Mov DI,2040h**
- **Mov ax,N1**
- **RCL ax,04**
- **Mov [di],ax**
- **Int 03h**
- **Code ends**
- **End**

Sorting

Assembly language program to sort the given numbers in Ascending order

```
ASSUME CS: CODE
CODE SEGMENT
START:  MOV  AX,0000H
        MOV CH, 0004H
        DEC CH
UP1:    MOV CL, CH
        MOV SI, 2000H
UP:     MOV AL, [SI]
        INC SI
        CMP AL, [SI]
        JC DOWN
        XCHG AL,
        [SI] DEC SI
        MOV [SI], AL
        INC SI
DOWN:   DEC CL
        JNZ UP
        DEC CH
        JNZ UP1
        INT 3
CODE ENDS
END START
```

Assembly language program to sort the given numbers in Descending order

```
ASSUME CS: CODE

CODE SEGMENT

START:          MOV   AX,   0000H

                MOV   CH,   0004H

                DEC CH

UP1:            MOV CL, CH

                MOV SI, 2000H

  UP:           MOV AL, [SI]

                INC SI

                CMP AL, [SI]
```

```
            JNC DOWN
            XCHG AL, [SI]
            DEC SI
            MOV [SI], AL
            INC SI
DOWN:       DEC CL
            JNZ UP
            DEC CH
            JNZ UP1
            I NT 3
CODE ENDS
END START
```

Evaluation of arithmetic expressions

An Assembly program for performing the following operation Z= ((A-B)/10*C)

```
DATA  SEGMENT
A DB 60
B DB 20
C DB 5
Z  DW?
ENDS
CODE SEGMENT
ASSUME DS: DATA CS: CODE
        START:              MOV AX, DATA
                            MOV DS, AX
                            MOV AH, 0         ; Clear content of AX
                            MOV AL, A         ; Move A to register AL
        SUB AL, B        ; Subtract AL and B
        MUL C            ; Multiply C to AL
        MOV BL, 10       ; Move 10 to register BL
        DIV BL           ; Divide AL content by BL
        MOV Z, AX        ; Move content of AX to Z
        MOV AH, 4CH
        INT 21H
ENDS
END START
```

Evaluation of arithmetic expressions

An Assembly program for performing the following operation Z= ((A-B)/10*C)

```
DATA  SEGMENT
A DB 60
B DB 20
C DB 5
Z  DW?
ENDS
CODE SEGMENT
ASSUME DS: DATA CS: CODE
        START:          MOV AX, DATA
                        MOV DS, AX
                        MOV AH, 0       ; Clear content of AX
                        MOV AL, A       ; Move A to register AL
        SUB AL, B       ; Subtract AL and B
        MUL C           ; Multiply C to AL
        MOV BL, 10      ; Move 10 to register BL
        DIV BL          ; Divide AL content by BL
        MOV Z, AX       ; Move content of AX to Z
        MOV AH, 4CH
        INT 21H
ENDS
END START
```

String manipulation

Program For String Transfer

```
DATA SEGMENT                              ; start of data segment
STR1 DB 'HOW ARE YOU'
LEN EQU $-STR1
STR2 DB 20 DUP (0)
DATA ENDS                                 ; end of data segment
CODE SEGMENT                              ; start of code segment
ASSUME CS: CODE, DS: DATA, ES: DATA
START:               MOV AX, DATA ; initialize data segment
                     MOV DS, AX
```

```
          MOV  ES,  AX        ; initialize extra segment for string operations
          LEA  SI,  STR1      ; SI points to starting address of string at ; STR1
          LEA DI, STR2        ; DI points to starting address of where the string
                              has to be transferred
          MOV CX, LEN         ; load CX with length of the string
          CLD                 ; clear the direction flag for auto increment SI;
      and                                   DI
          REP MOVSB           ; the source string is moved to destination
      address                               till CX=0(after every move CX is;
      decremented)
          MOV AH, 4CH         ; terminate the process
          INT 21H
CODE ENDS                     ; end of code segment
END START
```

Program To Reverse A String

```
DATA   SEGMENT                              ; start of data segment
STR1 DB 'HELLO'
LEN EQU $-STR1
STR2 DB 20 DUP (0)
DATA ENDS                                   ; end of data segment
CODE SEGMENT                                ; start of code segment
ASSUME CS: CODE, DS: DATA, ES: DATA
START:       MOV AX, DATA                   ; initialize data segment
             MOV DS, AX
             MOV ES, AX
```

```
            LEA SI, STR1
            LEA DI, STR2+LEN-1
            MOV CX, LEN
UP:         CLD
            LODSB
            STD
            STOSB
            LOOP UP
            MOV AH, 4CH
            INT 21H
CODE ENDS
END START
```

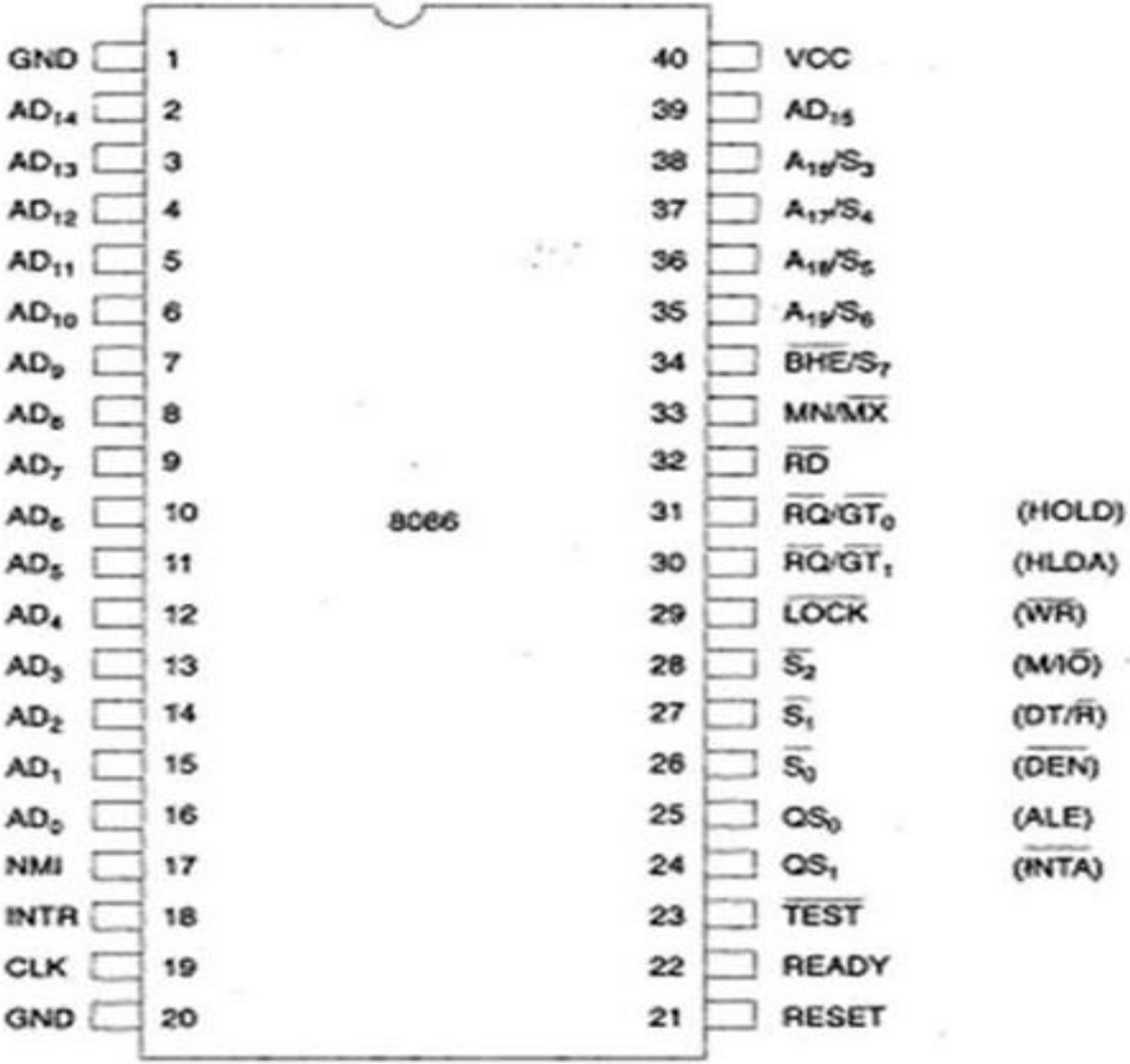
MODULE-II
Operation of 8086 and Interrupts.

COURSE OUTCOMES

CO4	Distinguish between minimum mode and maximum mode operation of 8086 microprocessor with timing diagrams.
CO5	Interpret the functionality of various types of interrupts and their structure for controlling the processor or controller and program execution flow.

Pin diagram of 8086

PIN DIAGRAM OF 8086



8086 operates in single processor or multiprocessor configuration to achieve high performance.

8086 is available in three clock rates 5,8,10 MHZ.

8086 signals can be categorised in to three groups

i) Signals having common functions in minimum as well as maximum mode.

ii) Signals having special functions for minimum mode.

iii) Signals having special functions for maximum mode.

Signals common to both modes

AD_{15}-AD_0 : (Address/Data lines)

These are time multiplexed address and data lines, which carry address when ALE is high and later function as data lines when ALE is low

Address is available on the address lines during T1 state.

Data is available on the data bus during T2,T3,TW,T4 clock states of machine cycles.

TW is wait state of machine cycle.

These lines are active high and float to a tri state during interrupt acknowledge and local bus hold acknowledge cycles.

A19/S6,A18/S5,A17/S4,A16/S3: (Address/Status lines)

These are time multiplexed address and status lines.

During T1 these are the most significant address lines for memory operations .

During memory or I/O operations status information is available on these lines for T2,T3,TW and T4.

The status of the interrupt enable flag bit is updated at the beginning of each clock cycle.

S4 and S3 indicates which segment register is presently being used for memory access.

S4	S3	indicates
0	0	Extra Segment(ES)
0	1	Stack Segment (SS)
1	0	Code Segment or none(CS)
1	1	Data Segment (DS)

These lines float to tri-stat during the local bus hold acknowledge.

S6 is always 0.

S5 is the condition of the interrupt flag .

Address bits are separated from the status bits using latches controlled by ALE signal.

<u>BHE’/S7: </u>Bus high enable signal is used to indicate the transfer of data over higher order (D15 –D8) data bus.

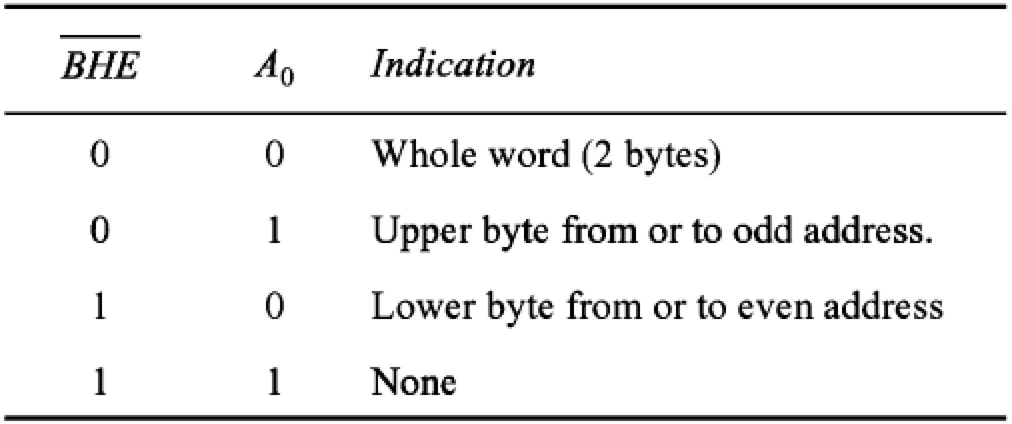
BHE’ is low for data transfer over(D15-D8).

BHE’ is low during T1 for read,write and interrupt acknowledge cycles whenever a byte is to be transferred on the higher byte of data bus.The status information is available during T2,T3,and T4.

The signal is active low and tristated during hold.

The status of this pin is latched along with the address information.S7 is always 1.

.



$\overline{BHE}$	A_0	*Indication*
0	0	Whole word (2 bytes)
0	1	Upper byte from or to odd address.
1	0	Lower byte from or to even address
1	1	None

RD': (Read)

When this signal is low data can be received from memory or input devices.

RD' is active low during T2,T3 and Tw of any read cycle.

RD' remains tristated during the hold acknowledgement.

Ready:

This is the acknowledgement from the slow device or memory that they have completed the data transfer.

The signal is active high.

If it is at logic low, wait states are inserted into the current bus cycle.

INTR- **(Interrupt Request)**

This is a triggered input.

This is sampled during the last clock cycles of each instruction to determine the availability of the request.

If any interrupt request is pending, the processor enters the interrupt acknowledge cycle.

This signal is active high and internally synchronized.

TEST':

This input is examined by a 'WAIT' instruction.

If the TEST' pin goes low, execution will continue, else the processor remains in an idle state.

The input is synchronized internally during each clock cycle on leading edge of clock.

NMI(Non Maskable Interrupt):

This is an edge triggered input which causes a type-2 interrupt.

A transition from low to high initiates the interrupt response at the end of the current instruction.

This input is internally synchronized.

RESET:

The input causes the processor to terminate current activity and start execution .

The signal is active high and must be active for atleast four clock cycles.

It restarts execution when the reset returns low.

Reset is also internally synchronized.

CLK:

The clock input provides the basic timing for processor operation and bus control activity.

The range of frequency for different 8086 versions is from 5MHZ to 10MHZ.

Vcc:

+5V power supply for the operation of internal circuit.

GND:

Ground for the internal circuit.

8086 Minimum Mode Pins

INTA’ (Interrupt Acknowledge):

In response to INTR,the processor sends acknowledge signal that the interrupt is accepted through INTA pin.

M/IO’: (Memory/input output)

When M/IO’=1 it performs the memory read/write operations.

When M/IO’=0 it performs the I/O read/write operations.

WR’: (Write)

It is active low pin.

It indicates microprocessor is sending data to memory or I/O devices.

DT/R’: (Data transmit/Receive)

It indicates transmitting or receiving data over system bus.

If DT/R’=1 data transmit; If DT/R’=0 data receive.

It is used to enable external data buffer.

DEN’: (Data enable)

It is used to enable external data bus buffer.

When DEN’=0 data transferred on data bus.

ALE: (Address Latch Enable)

When ALE=1 multiplexed line carries address only.

When ALE=0 multiplexed line carries data .

HOLD:

When HOLD line is high it indicates processor that another master is requesting bus access.

HLDA: (Hold Acknowledgement)

After receiving HOLD request, the processor issues the acknowledge on HLDA pin.

Maximum Mode Pins

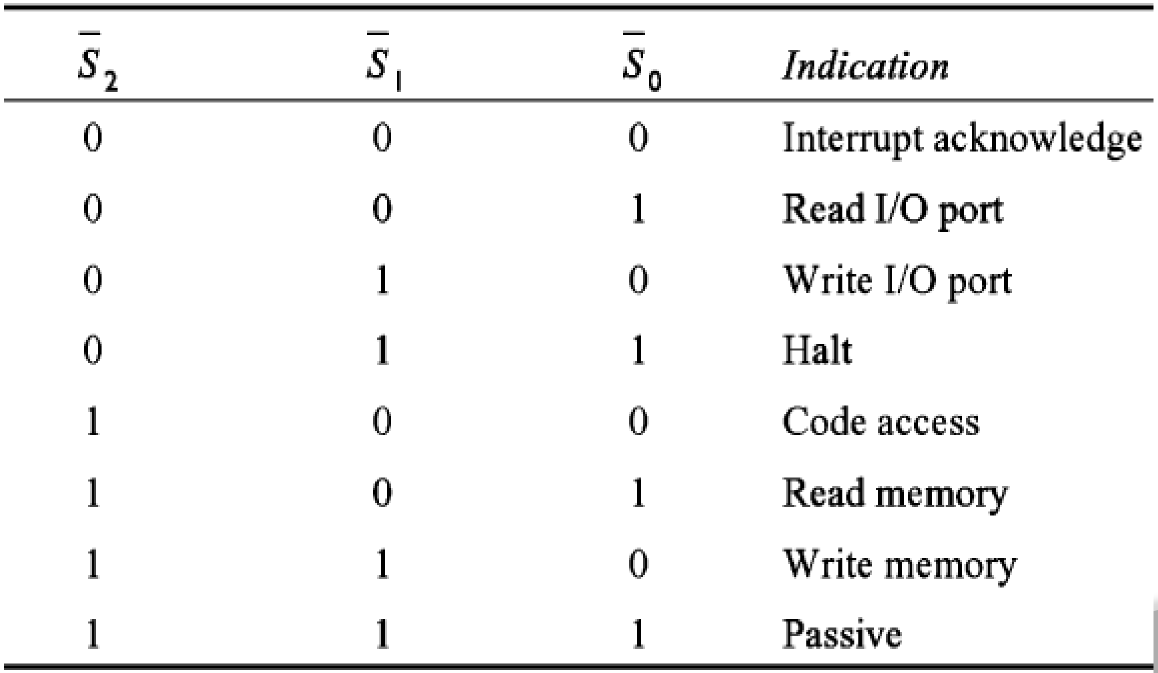
MN/MX’:

If MN/MX’=0 it indicates maximum mode of operation.

If MN/MX’=1 it indicates minimum mode of operation.

S2’,S1’,S0’: (Status Lines)

It indicates the type of operation carried out by processor.



$\overline{S}_2$	$\overline{S}_1$	$\overline{S}_0$	*Indication*
0	0	0	Interrupt acknowledge
0	0	1	Read I/O port
0	1	0	Write I/O port
0	1	1	Halt
1	0	0	Code access
1	0	1	Read memory
1	1	0	Write memory
1	1	1	Passive

RQ1’/GT1’; RQ0’/GT0: (Request/Grant)

If the other processor wants to access the system bus ,then it is going to request the processor which is currently using system bus through this pin.

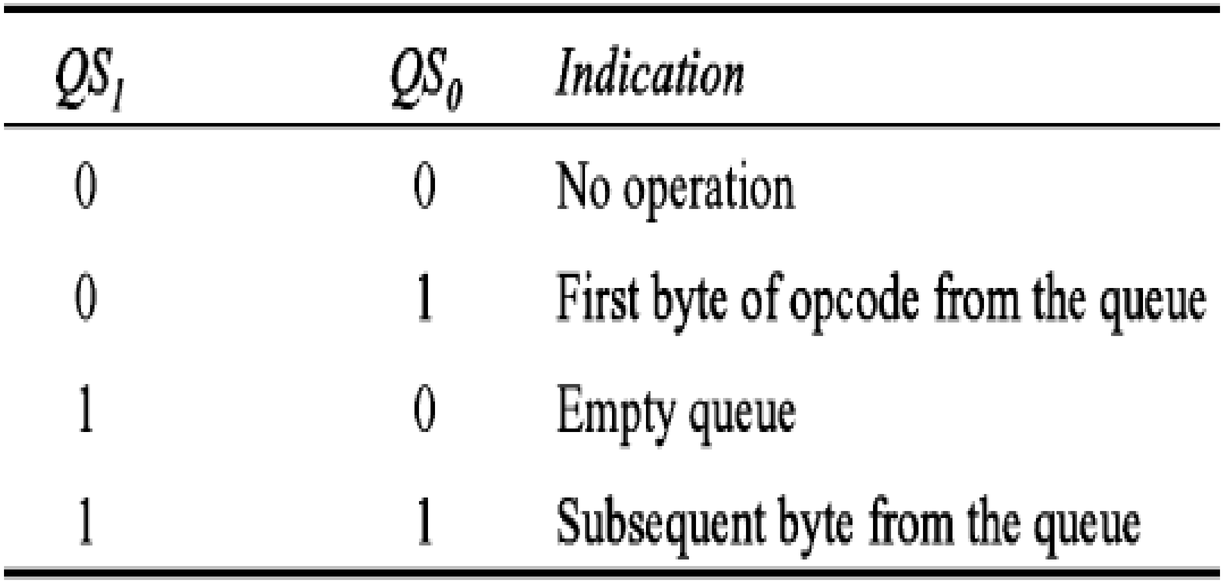
These are bidirectional pins.

It will send the acknowledge through same pins.i.e; grant.

LOCK’:

When this signal is enabled the system bus is locked for certain duration. So it cannot be used by other masters for some duration.

QS1,QS0:(Queue status):

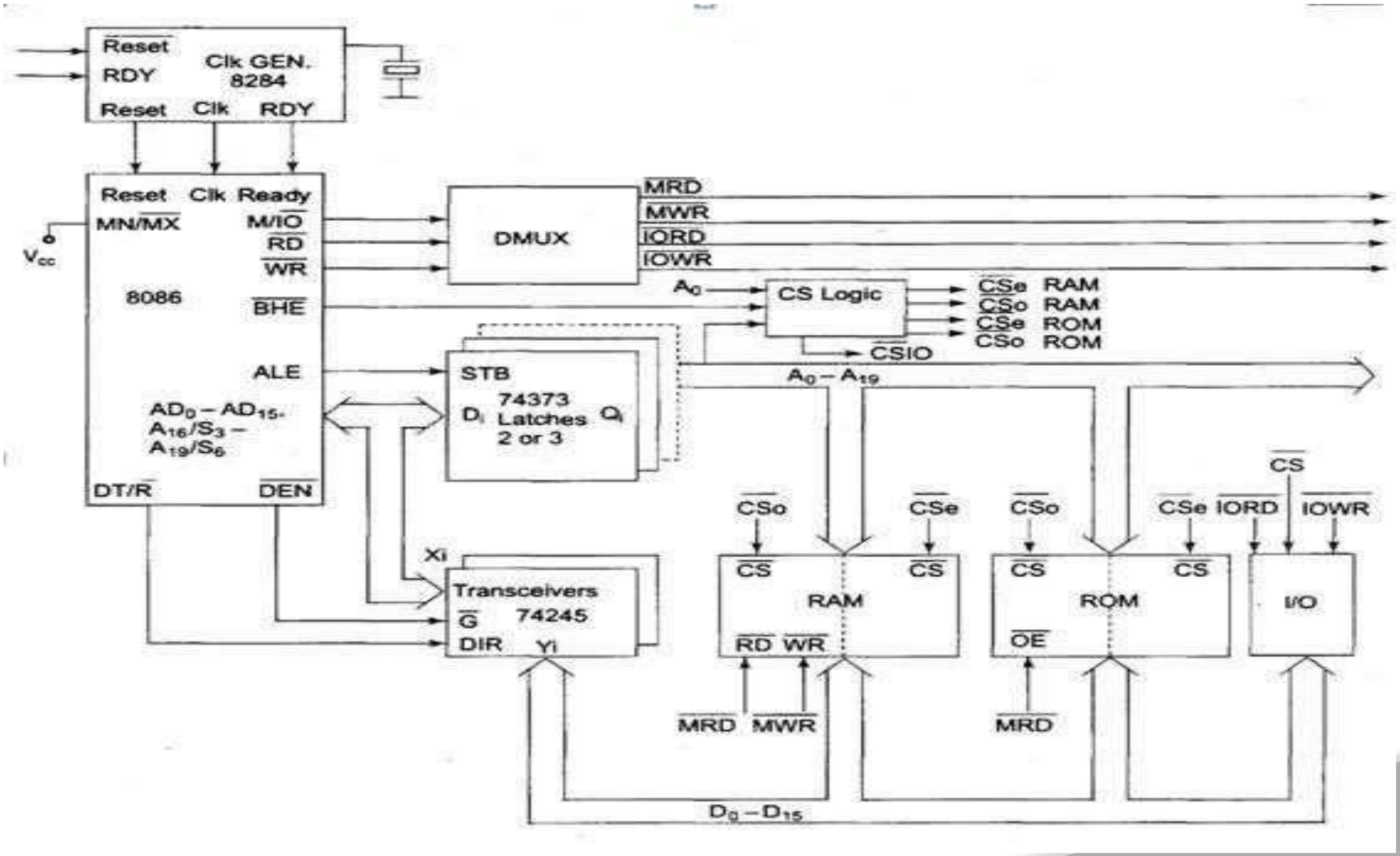


QS_1	QS_0	*Indication*
0	0	No operation
0	1	First byte of opcode from the queue
1	0	Empty queue
1	1	Subsequent byte from the queue

Minimum mode and maximum mode of operation with Timing diagrams

Minimum mode operation in 8086

Minimum mode operation in 8086:



- In a minimum mode 8086 system, the microprocessor 8086 is operated
  in minimum mode by strapping its MN/MX pin to logic 1.
- In this mode, all the control signals are given out by the microprocessor chip itself. There is a single microprocessor in the minimum mode system.
- The remaining components in the system are latches, transceivers, clock generator, memory and I/O devices. Some type of chip selection logic may be required for selecting memory or I/O devices, depending upon the address map of the system.
- Latches are generally buffered output D-type flip-flops like 74LS373 or 8282. They are used for separating the valid address from the multiplexed address/data signals and are controlled by the ALE signal generated by 8086.

- Transceivers are the bidirectional buffers and sometimes they are called as data amplifiers. They are required to separate the valid data from the time multiplexed address/data signals.

- They are controlled by two signals namely, DEN and DT/R.

- The DEN signal indicates the direction of data, i.e. from or to the processor. The system contains memory for the monitor and users program storage.

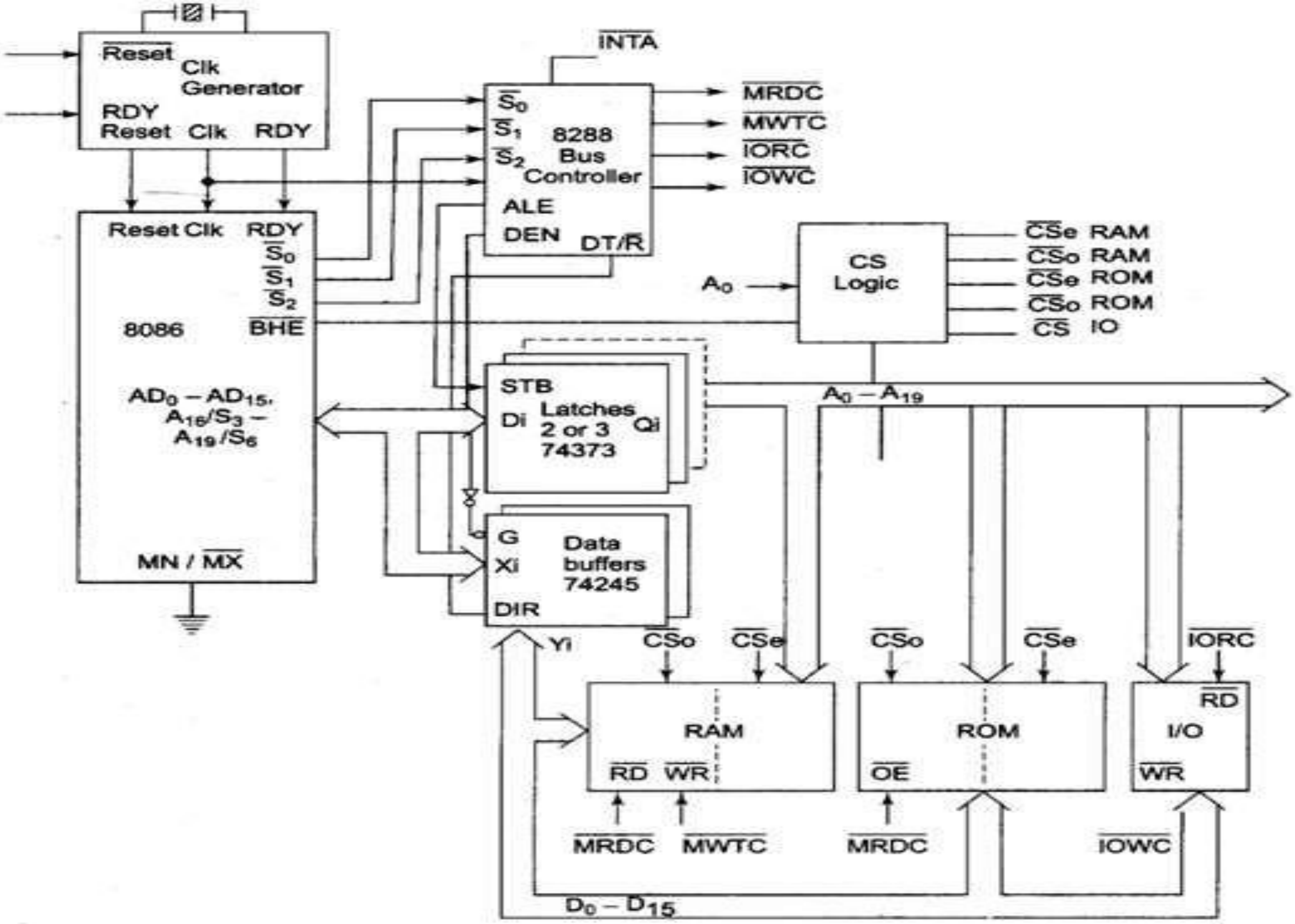
- Usually, EPROM is used for monitor storage, while RAM for users
  program storage. A system may contain I/O devices.

Maximum mode operation in 8086

- In the maximum mode, the 8086 is operated by strapping the MN/MX pin to ground.
- In this mode, the processor derives the status signal S2, S1, S0. Another chip called bus controller derives the control signal using this status information.
- In the maximum mode, there may be more than one microprocessor in the system configuration.

- The components in the system are same as in the minimum mode system.
- The basic function of the bus controller chip IC8288 is to derive control signals like RD and WR (for memory and I/O devices), DEN, DT/R, ALE etc. using the information by the processor on the status lines.
- The bus controller chip hasinput lines S2, S1, S0 and CLK. These inputs to 8288 are driven by CPU.

Maximum mode



- It derives the outputs ALE, DEN, DT/R, MRDC, MWTC, AMWC, IORC, IOWC and AIOWC. The AEN, IOB and CEN pins are especially useful for multiprocessor systems.

- AEN and IOB are generally grounded. CEN pin is usually tied to
  +5V. The significance of the MCE/PDEN output depends upon the
  status of the IOB pin.

- If IOB is grounded, it acts as master cascade enable to control cascade 8259A, else it acts as peripheral data enable used in the multiple bus configurations.

- INTA pin used to issue two interrupt acknowledge pulses to the interrupt controller or to an interrupting device.

- IORC, IOWC are I/O read command and I/O write command signals respectively.

- These signals enable an IO interface to read or write the data from or to the address port.

- The MRDC, MWTC are memory read command and memory write command signals respectively and may be used as memory read or write signals.

- The MRDC, MWTC are memory read command and memory write command signals respectively and may be used as memory read or write signals.

- All these command signals instructs the memory to accept or send data from or to the bus.
- For both of these write command signals, the advanced signals namely AIOWC and AMWTC are available.

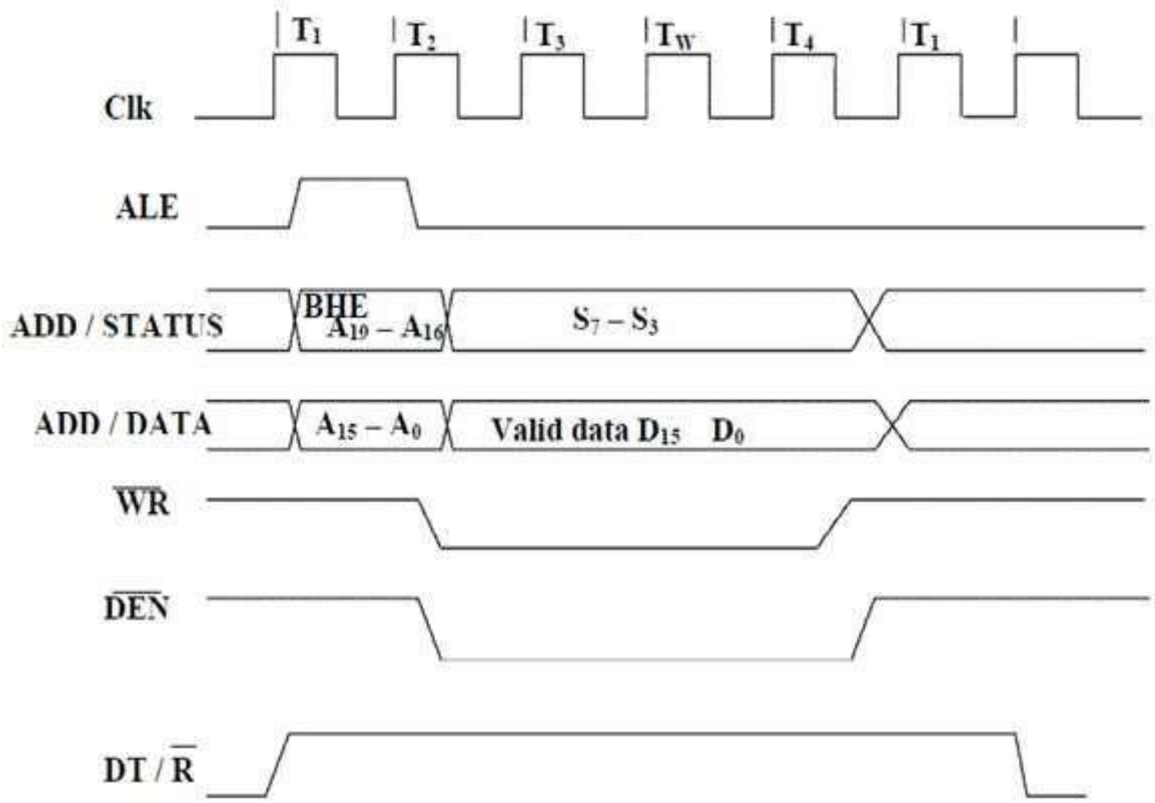
- Here the only difference between in timing diagram between minimum mode and maximum mode is the status signals used and the available control and advanced command signals.

- R0, S1, S2 are set at the beginning of bus cycle.8288 bus controller will output a pulse as on the ALE and apply a required signal to its DT / R pin during T1.

- In T2, 8288 will set DEN=1 thus enabling transceivers, and for an input it will activate MRDC or IORC. These signals are activated until T4. For an output, the AMWC or AIOWC is activated from T2 to T4 and MWTC or IOWC is activated from T3 to T4.

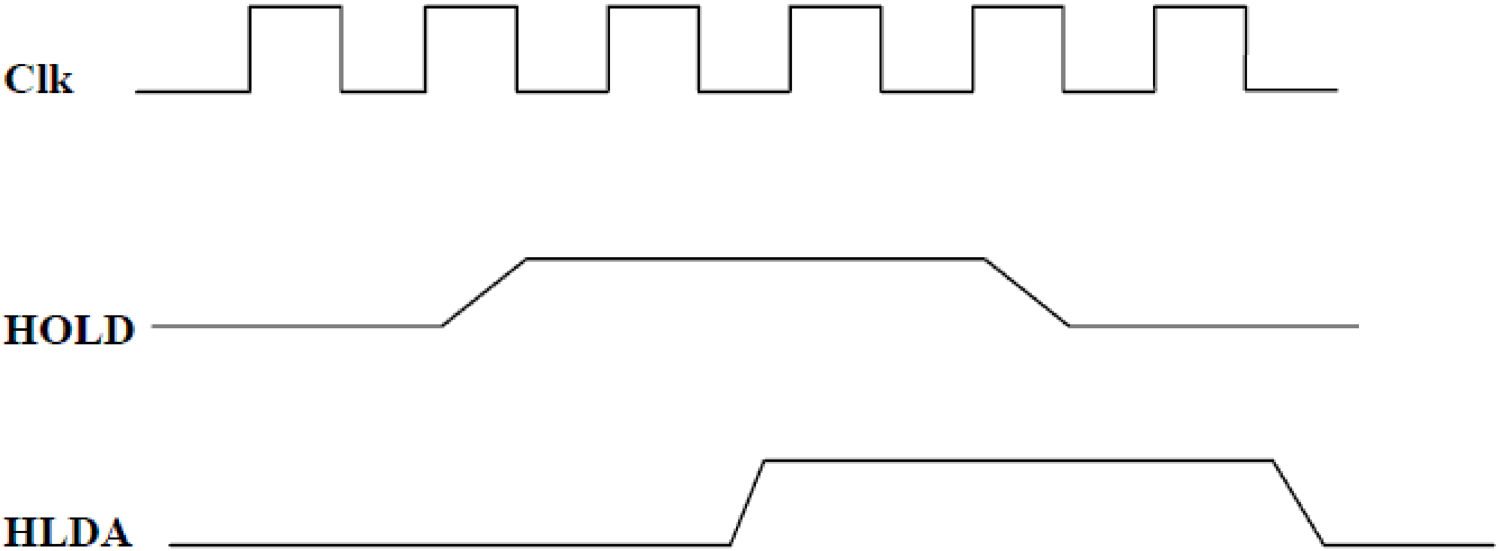
Timing diagram for minimum mode

Write Cycle Timing Diagram for



- The working of the minimum mode configuration system can be better described in terms of the timing diagrams rather than qualitatively describing the operations.
- The opcode fetch and read cycles are similar. Hence the timing diagram can be categorized in two parts, the first is the timing diagram for read cycle and the second is the timing diagram for write cycle.

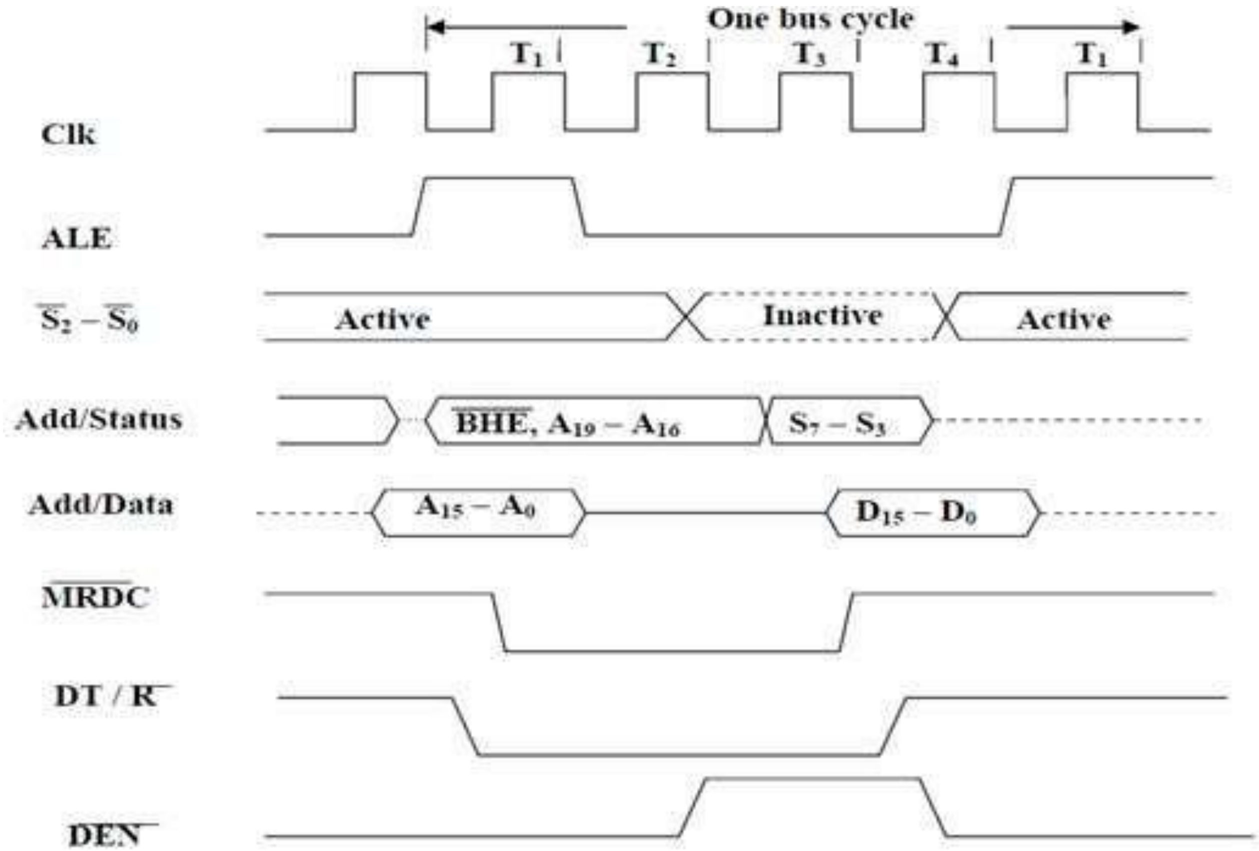
Bus Request and Bus Grant Timings in Minimum Mode System of 8086



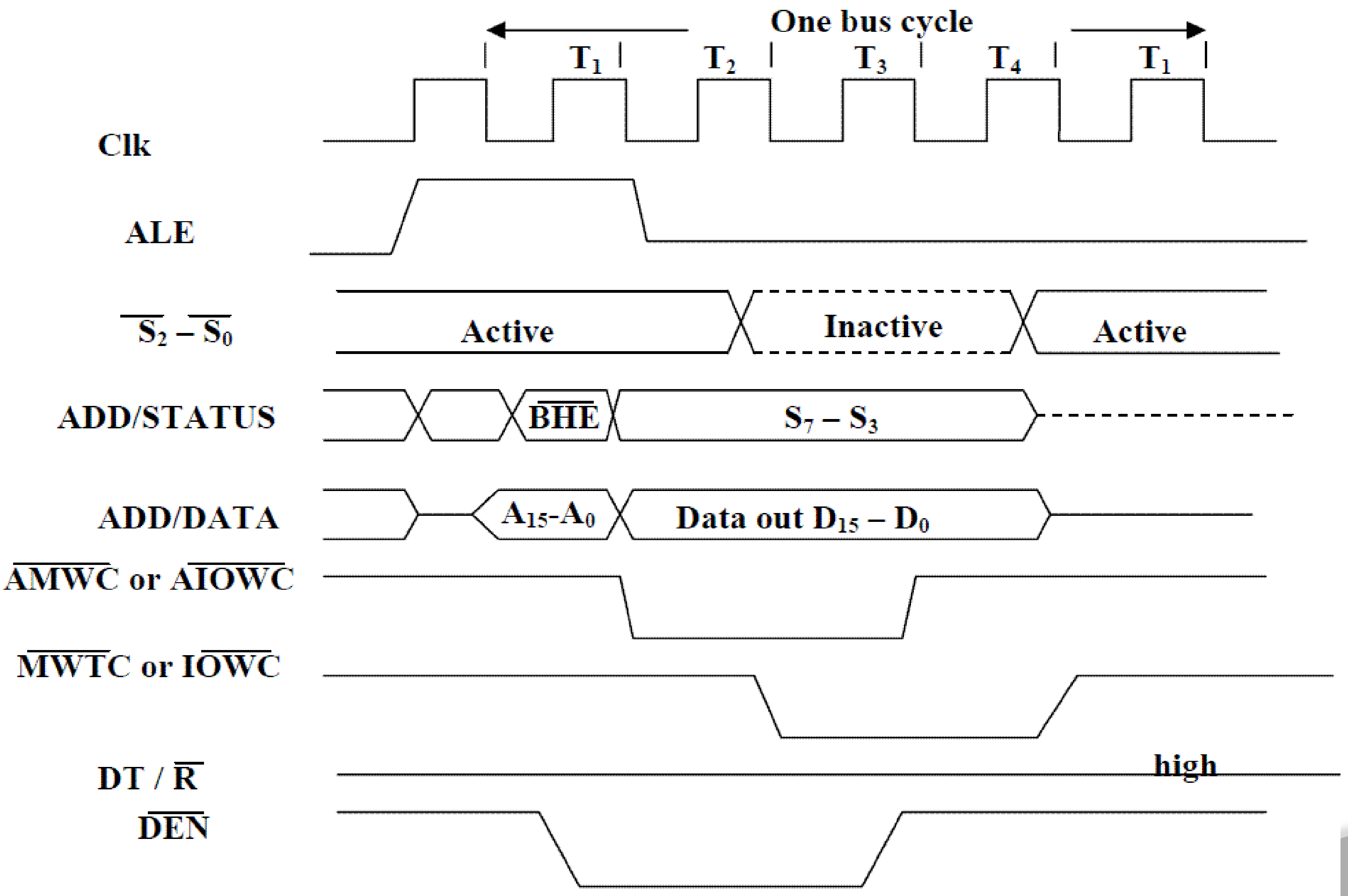
Bus Request and
Bus Grant Timings in Minimum Mode System

Timing diagram for maximum mode

Memory Read Timing Diagram in Maximum Mode of 8086

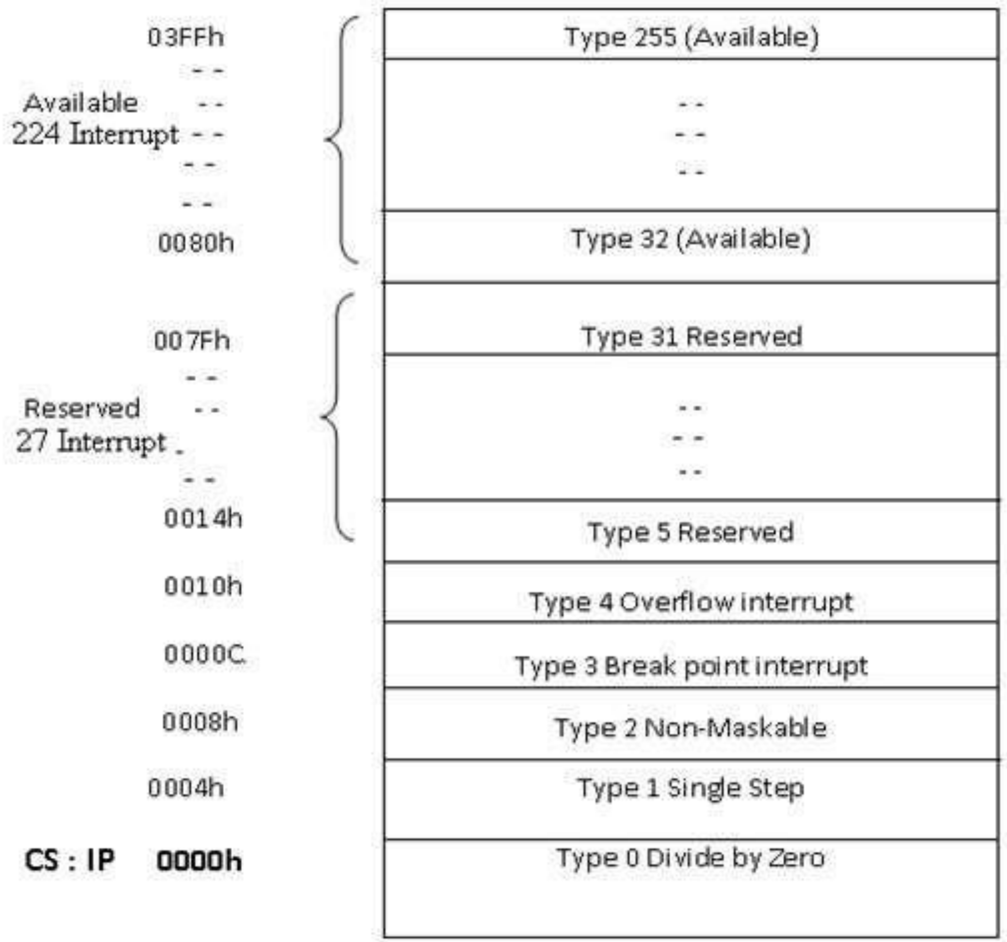


Memory Write Timing in Maximum mode of 8086



Interrupt structure of 8086

Interrupt structure of 8086



03FFh

- -

Available - -

224 Interrupt - -

- -

- -

0080h

007Fh

- -

Reserved - -

27 Interrupt -

- -

0014h

0010h

0000C

0008h

0004h

CS : IP 0000h

Type 255 (Available)

- -

- -

- -

Type 32 (Available)

Type 31 Reserved

- -

- -

- -

Type 5 Reserved

Type 4 Overflow interrupt

Type 3 Break point interrupt

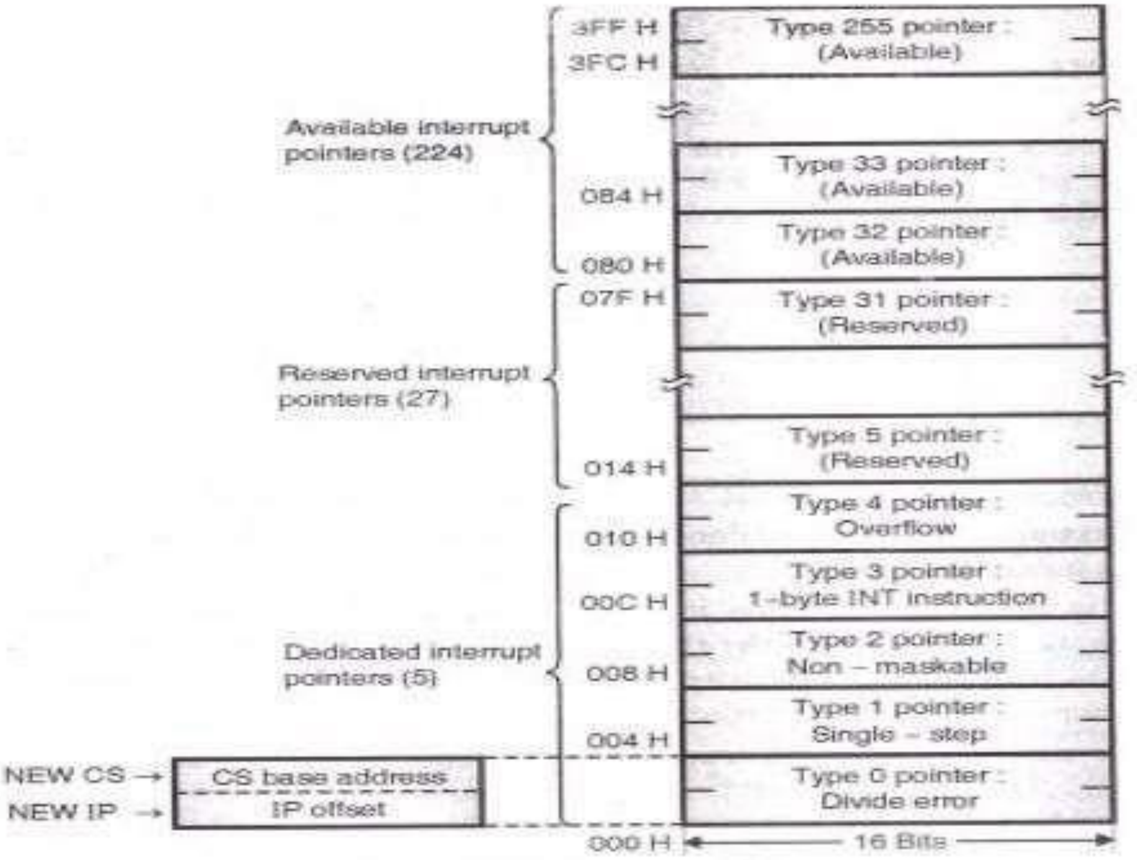
Type 2 Non-Maskable

Type 1 Single Step

Type 0 Divide by Zero

Vector interrupt table, interrupt service routines

Vector interrupt



Introduction to DOS and BIOS interrupts

BIOS INTERRUPT

BIOS INTERRUPT

- INT 10H – Video Screen
  - The option is chosen by putting a specific value in register AH
  - The video screen is text mode is divided into 80 columnsand 25 rows
  - A row and column number are associated with each location on the screen with the top left corner as 00,00 and the bottom right corner as 24,79. The center of the screen is at 12,39 or (0C,27 in hex)
  - Specific registers has to be set to specific values before invoking INT 10H

BIOS INTERRUPT

- *Function 06 – clear the screen*
- AH = 06 ; function number
- AL = 00 ; page number
- BH = 07 ; normal attribute
- CH = 00 ; row value of start point
- CL = 00 ; column value of start point
- DH = 24 ; row value of ending point
- DL = 79 ; column value of ending point

- *Function 02 – setting the cursor to a specific location*
- AH = 06 **; function number**
- DH = row **; cursor**
- DL = column ; position

BIOS INTERRUPT

- *Function 03 – get the current cursor position*
- AH = 03 **; function number**
- BH= 00 **; currently viewed page**
- The position is returned in DH = row and DL = column
-
- *Function 0E – output a character to the screen*
- AH = 0E ; function number
- AL = Character to be displayed
- BH = 00 **; currently viewed page**
- BL = 00 **; default foreground color**

DOS INTERRUPT

- *Function 09 – outputting a string of data to the monitor*
- AH = 09 ; function number
- DX = offset address of the ASCII data to be displayed, data segment is assumed
- The ASCII string must end with the dollar sign $

- *Function 02 – outputting a single character to the monitor*
- AH = 02 ; function number
- DL = ASCII code of the character to be displayed

- *Function 01 – inputting a single character, with an echo*
- AH = 01 ; function number.After the interrupt AL = ASCII code of the input and is echoed to the monitor

DOS INTERRUPT

- *Function 0A – inputting a string of data from the keyboard*
- AH = 0A ; function number
- DX = offset address at which the string of data is stored (buffer area), data
- segment is assumed and the string must end with <RETURN>
- After execution:
- DS:DX = buffer in bytes (n characters + 2)
- DS:DX+1 = number of entered characters excluding the return key
- DS:DX+2 = first character input
- . . .
- DS:DX+n = last character input
- To set a buffer, use the following in the data segment:
- Buffer DB 10, ? , 10 DUP(FF)

DOS INTERRUPT

- *Function 07 – inputting a single character from the keyboard without an echo*
- AH = 07 ; function number
- Waits for a single character to be entered and provides it in AL
- INT16 – Keyboard Programming
- *Function 01 – check for a key press without waiting for theuser*
- AH = 01
- Upon execution ZF = 0 if there is a key pressed

- *Function 00 – keyboard read*
- AH = 00
- Upon execution AL = ASCII character of the pressed key
- Note this function must follow function01

MODULE-III
Interfacing with 8086

COURSE OUTCOMES

CO6	Demonstrate the internal architecture and various modes of operation of the devices used for interfacing memory and I/O devices with microprocessor.
CO7	Choose an appropriate data transfer scheme and hardware to perform serial data transfer among the devices.

Memory interfacing to 8086 (Static RAM and EPROM)

- **Interface two 4Kx8 EPROMS and two 4Kx8 RAM chips with 8086. select suitable maps.**

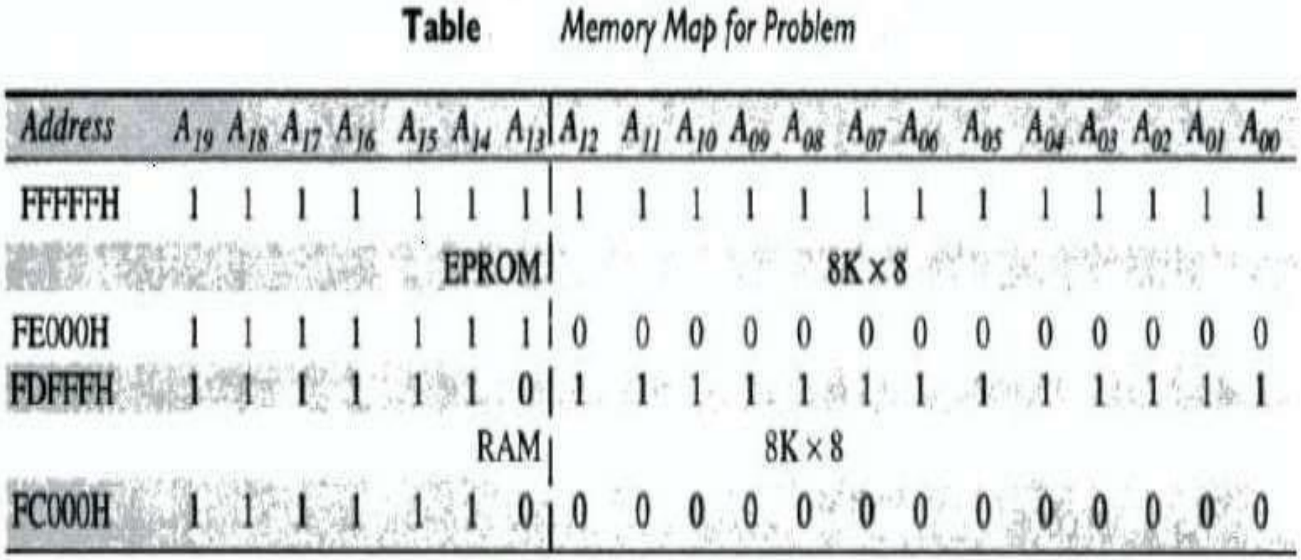


Table *Memory Map for Problem*

Address	A_{19}	A_{18}	A_{17}	A_{16}	A_{15}	A_{14}	A_{13}	A_{12}	A_{11}	A_{10}	A_{09}	A_{08}	A_{07}	A_{06}	A_{05}	A_{04}	A_{03}	A_{02}	A_{01}	A_{00}
FFFFFH	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
							EPROM						8K × 8							
FE000H	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0
FDFFFH	1	1	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1
							RAM					8K × 8								
FC000H	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0

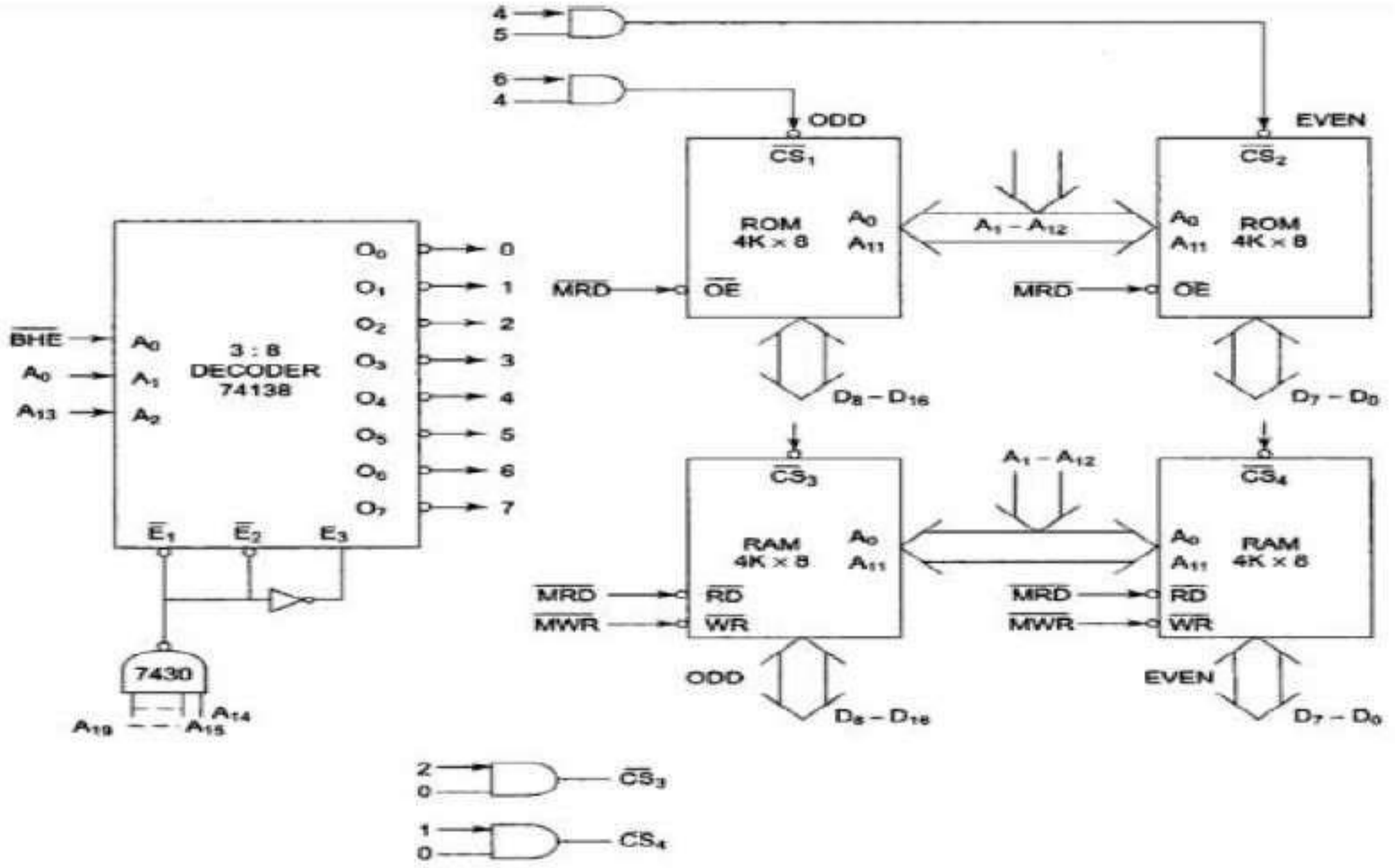


Fig shows the interfacing diagram for the memory system

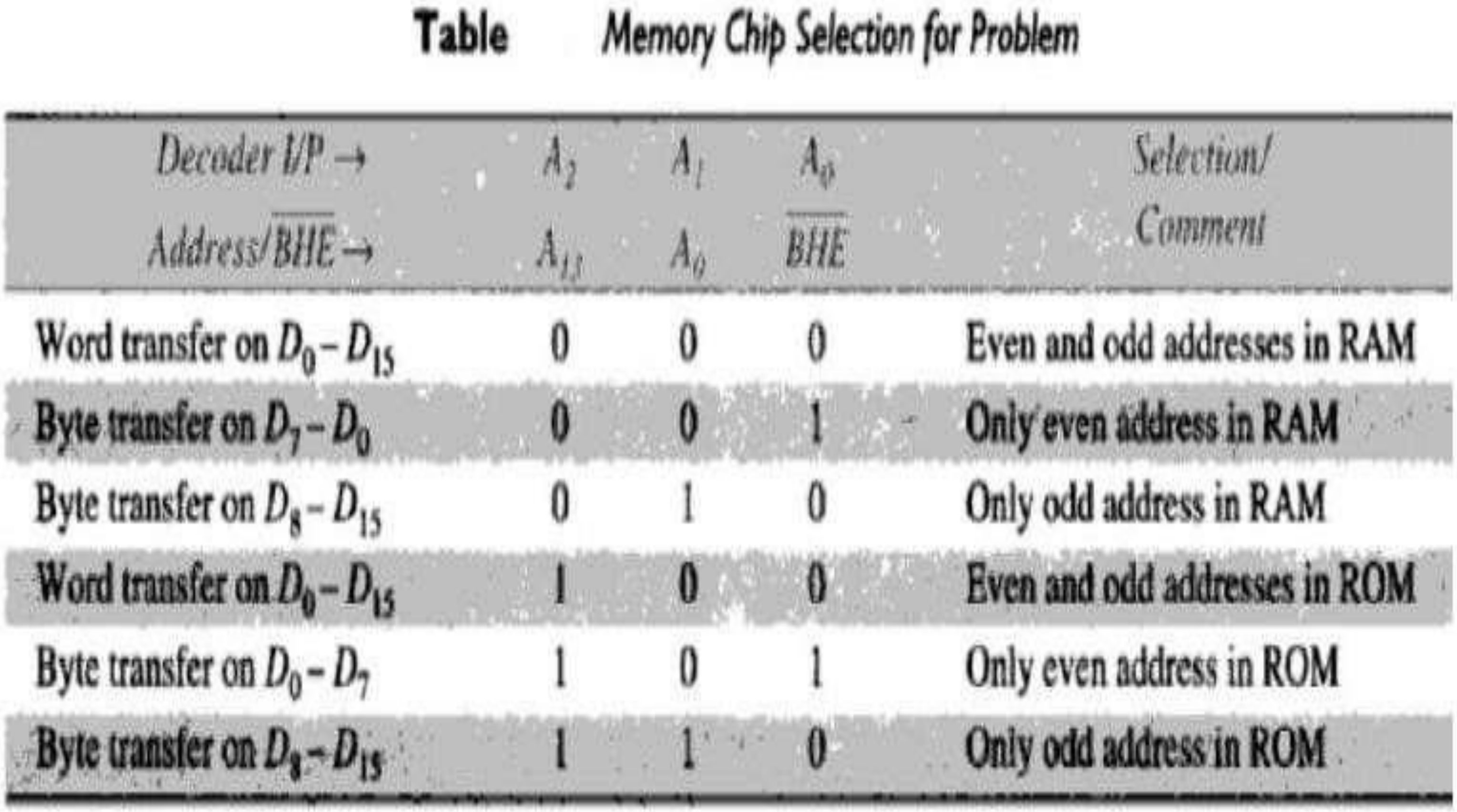


Table *Memory Chip Selection for Problem*

Decoder I/P →	A_2	A_1	A_0	*Selection/ Comment*
Address/$\overline{BHE}$ →	A_{13}	A_0	$\overline{BHE}$	
Word transfer on $D_0 - D_{15}$	0	0	0	Even and odd addresses in RAM
Byte transfer on $D_7 - D_0$	0	0	1	Only even address in RAM
Byte transfer on $D_8 - D_{15}$	0	1	0	Only odd address in RAM
Word transfer on $D_0 - D_{15}$	1	0	0	Even and odd addresses in ROM
Byte transfer on $D_0 - D_7$	1	0	1	Only even address in ROM
Byte transfer on $D_8 - D_{15}$	1	1	0	Only odd address in ROM

Need for DMA

Need For DMA

- Direct memory access (DMA) is a feature of modern computer systems that allows certain hardware subsystems to read/write data to/from memory without microprocessor intervention, allowing the processor to do other work.

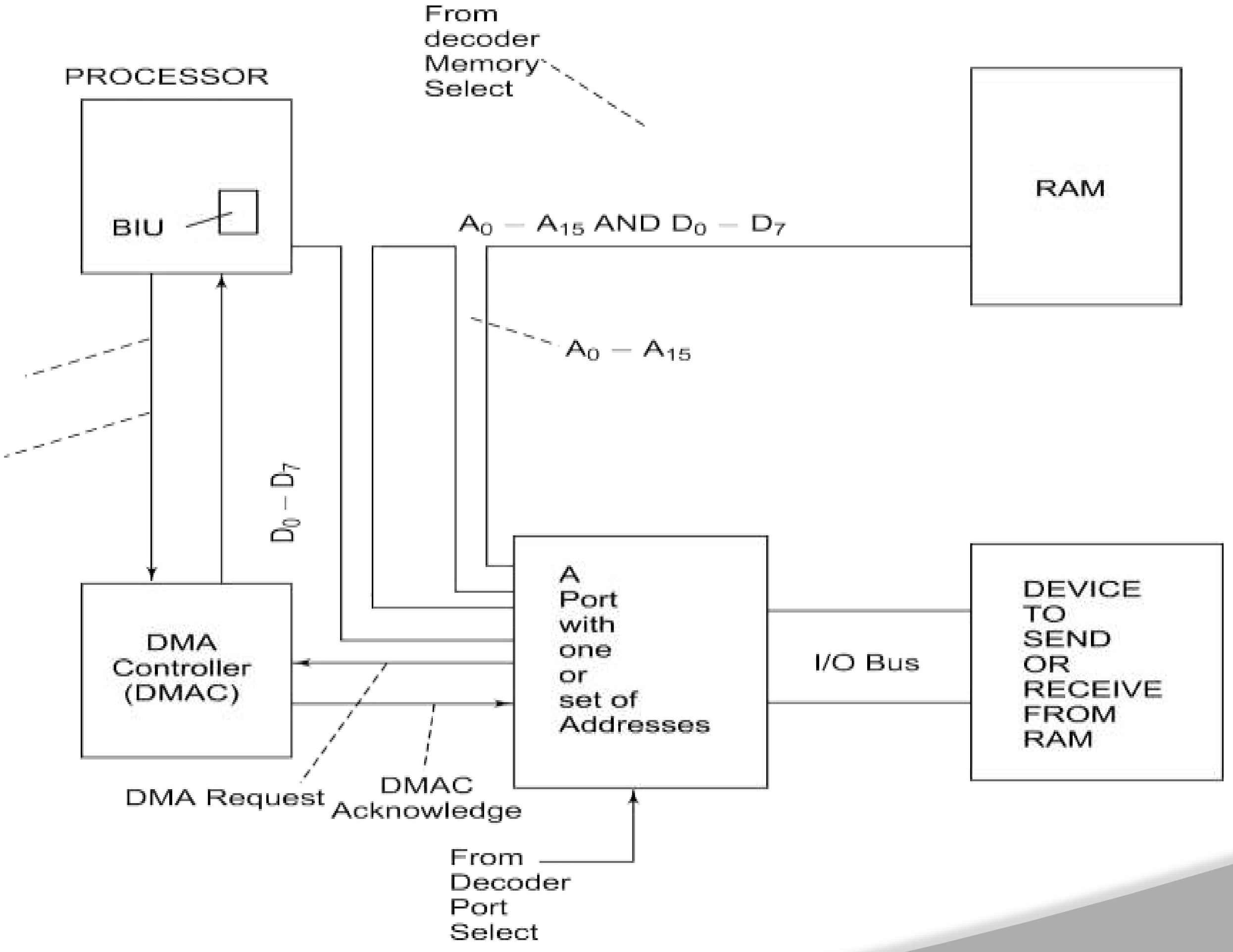
- Used in disk controllers, video/sound cards etc, or between
memory locations.

- Typically, the CPU initiates DMA transfer, does other operations while the transfer is in progress, and receives an interrupt from the DMA controller once the operation is complete.

- Can create cache coherency problems (the data in the cache may be different from the data in the external memory after DMA)

DMA Data transfer Method

DMA Data Transfer Method



- The I/O device asserts the appropriate DRQ signal for the channel.

- The DMA controller will enable appropriate channel, and ask the CPU to release the bus so that the DMA may use the bus. The DMA requests the bus by asserting the HOLD signal which goes to the CPU.

- The CPU detects the HOLD signal, and will complete executing the current instruction. Now all of the signals normally generated by the CPU are placed in a tri-stated condition (neither high or low) and then the CPU asserts the HLDA signal which tells the DMA controller that it is now in charge of the bus.
- The CPU may have to wait (hold cycles).

- DMA activates its -MEMR, -MEMW, -IOR, -IOW output signals, and the address outputs from the DMA are set to the target address, which will be used to direct the byte that is about to transferred to a specific memory location.

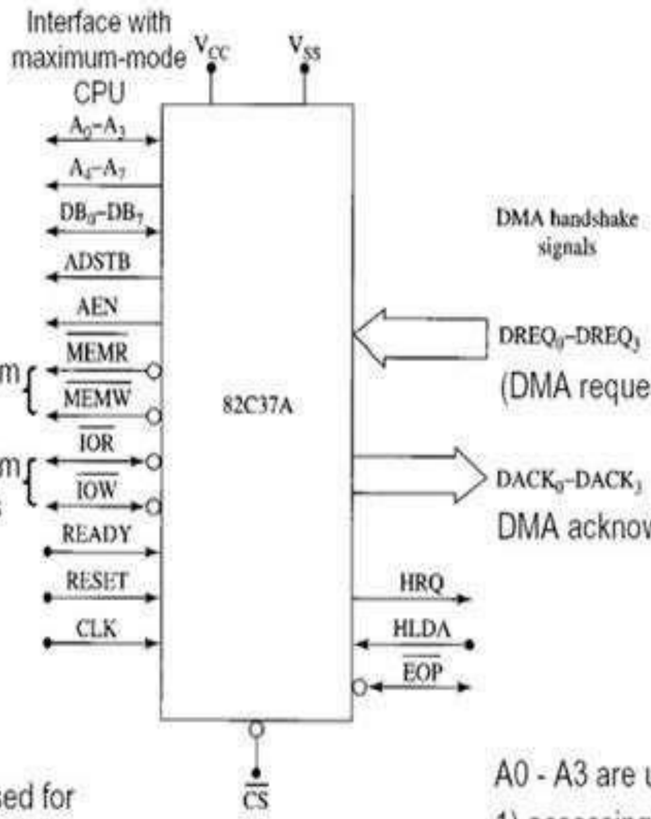
- The DMA will then let the device that requested the DMA transfer know that the transfer is commencing by asserting the - DACK signal.
- The peripheral places the byte to be transferred on the bus Data lines.
- Once the data has been transferred, The DMA will de-assert the - DACK2 signal, so that the FDC knows it must stop placing data on the bus.

- The DMA will now check to see if any of the other DMA channels have any work to do. If none of the channels have their DRQ lines asserted, the DMA controller has completed its work and will now tri-state the -MEMR, -MEMW, -IOR, -IOW and address signals.

- Finally, the DMA will de-assert the HOLD signal. The CPU sees this, and de-asserts the HOLDA signal. Now the CPU resumes control of the buses and address lines, and it resumes executing instructions and accessing main memory and the peripherals.

8237-DMA Controller

Pin diagram



control signals from and to memory

control signals from and to peripherals

(DMA requests for the 4 channels)

DMA acknowledge

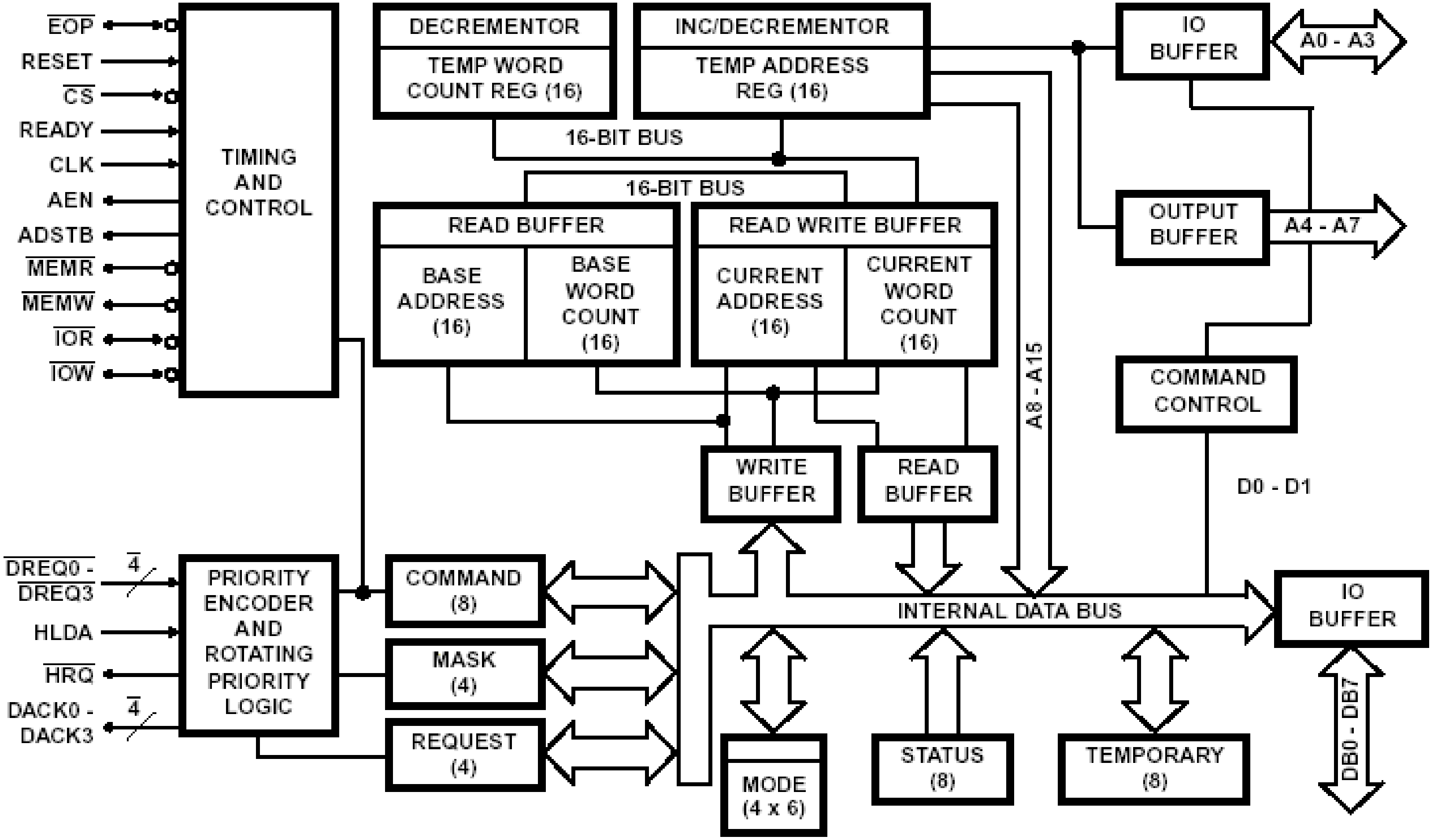
DB0 - DB7 are used for

1) transfer of data
2) 8237 programming

A0 - A3 are used for

1) accessing 8237 internal ports
2) carrying memory address in DMA read and write operations

Block Diagram



8237 Internal Registers

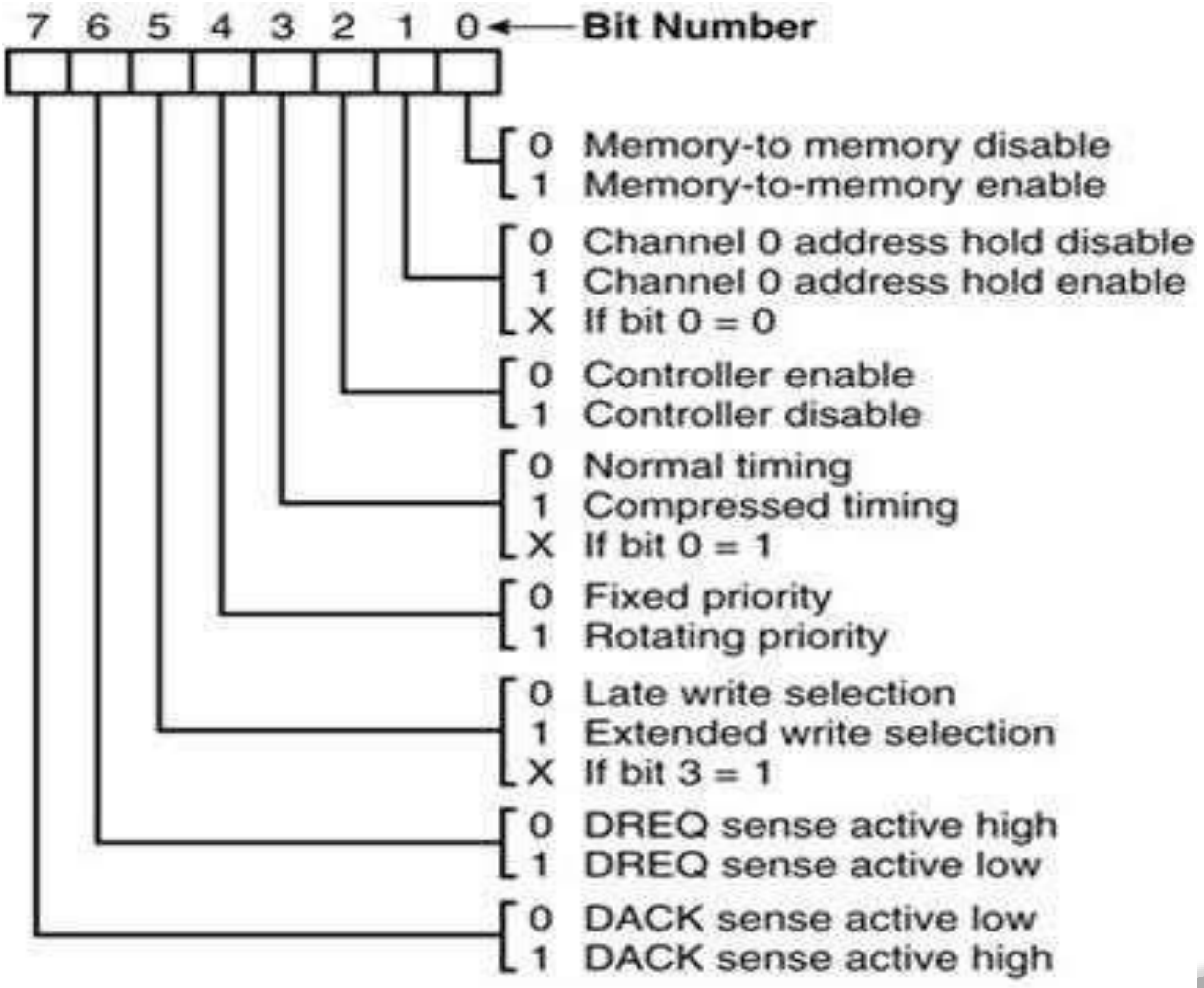
- *CAR*
- The current address register holds a 16-bit memory address used for the DMA transfer.
- each channel has its own current address register for this purpose.
- When a byte of data is transferred during a DMA operation, CAR is either incremented or decremented. depending on how it is programmed
- *CWCR*
- The current word count register programs a channel for the number of bytes to transferred during a DMA action.

*CR(*Command Register)

- The command register programs the operation of the 8237 DMA
controller.

- The register uses bit position 0 to select the memory-to-memory DMA transfer mode.

  - Memory-to-memory DMA transfers use DMA channel
  - DMA channel 0 to hold the source address
  - DMA channel 1 holds the destination address



7 6 5 4 3 2 1 0 ← Bit Number

Bit 0:
- 0 Memory-to memory disable
- 1 Memory-to-memory enable

Bit 1:
- 0 Channel 0 address hold disable
- 1 Channel 0 address hold enable
- X If bit 0 = 0

Bit 2:
- 0 Controller enable
- 1 Controller disable

Bit 3:
- 0 Normal timing
- 1 Compressed timing
- X If bit 0 = 1

Bit 4:
- 0 Fixed priority
- 1 Rotating priority

Bit 5:
- 0 Late write selection
- 1 Extended write selection
- X If bit 3 = 1

Bit 6:
- 0 DREQ sense active high
- 1 DREQ sense active low

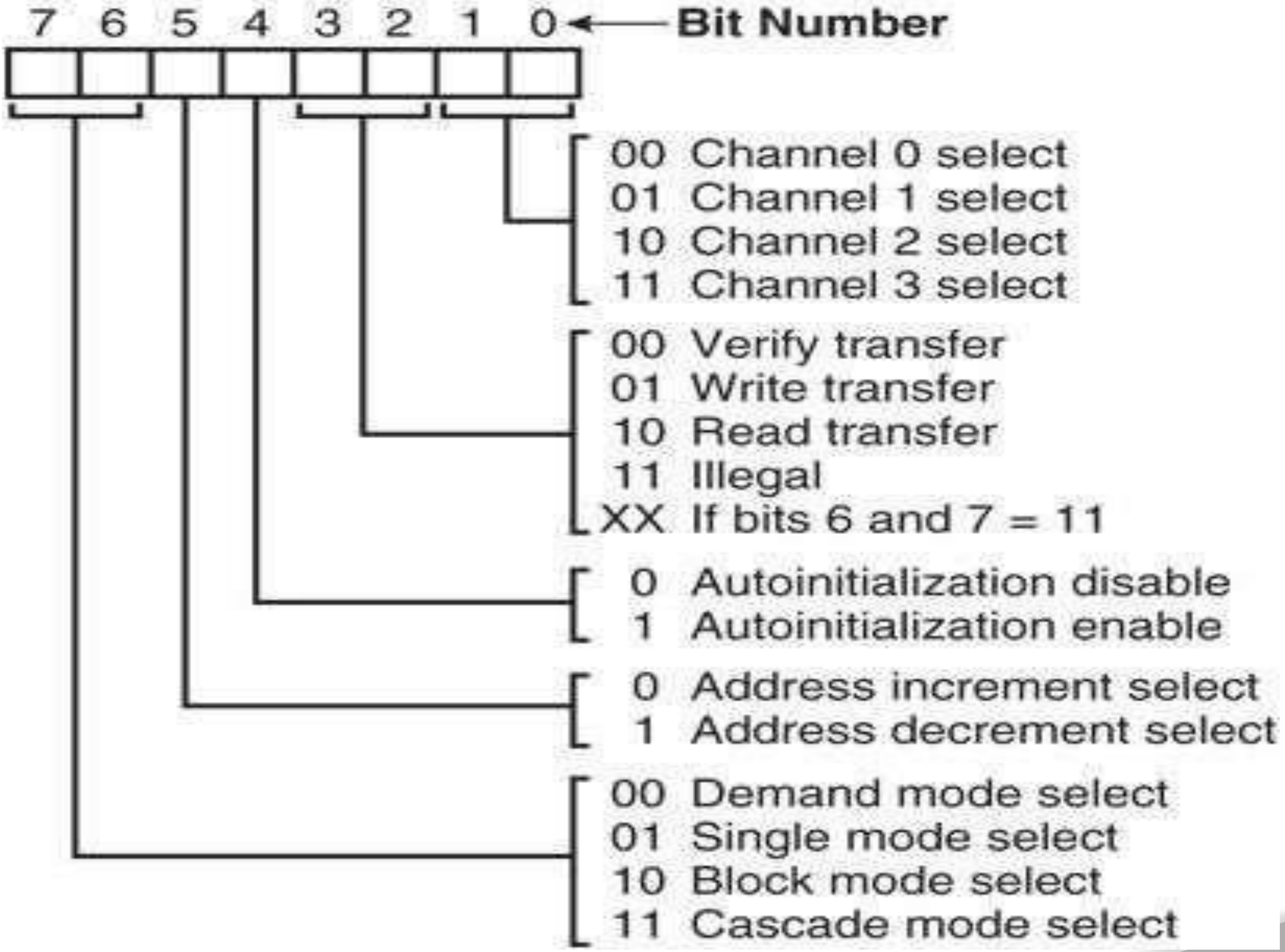
Bit 7:
- 0 DACK sense active low
- 1 DACK sense active high

BA and BWC

- The base address (BA) and base word count (BWC) registers are used when auto-initialization is selected for a channel.
- In auto-initialization mode, these registers are used to reload the CAR and CWCR after the DMA action is completed.

MR-(Mode Register)

- The mode register programs the mode of operation for a channel.
- Each channel has its own mode register as selected by bit positions 1 and 0.
- Remaining bits of the mode register select operation, auto-initialization, increment/decrement, and mode for the channel



00 Channel 0 select
01 Channel 1 select
10 Channel 2 select
11 Channel 3 select

00 Verify transfer
01 Write transfer
10 Read transfer
11 Illegal
XX If bits 6 and 7 = 11

0 Autoinitialization disable
1 Autoinitialization enable

0 Address increment select
1 Address decrement select

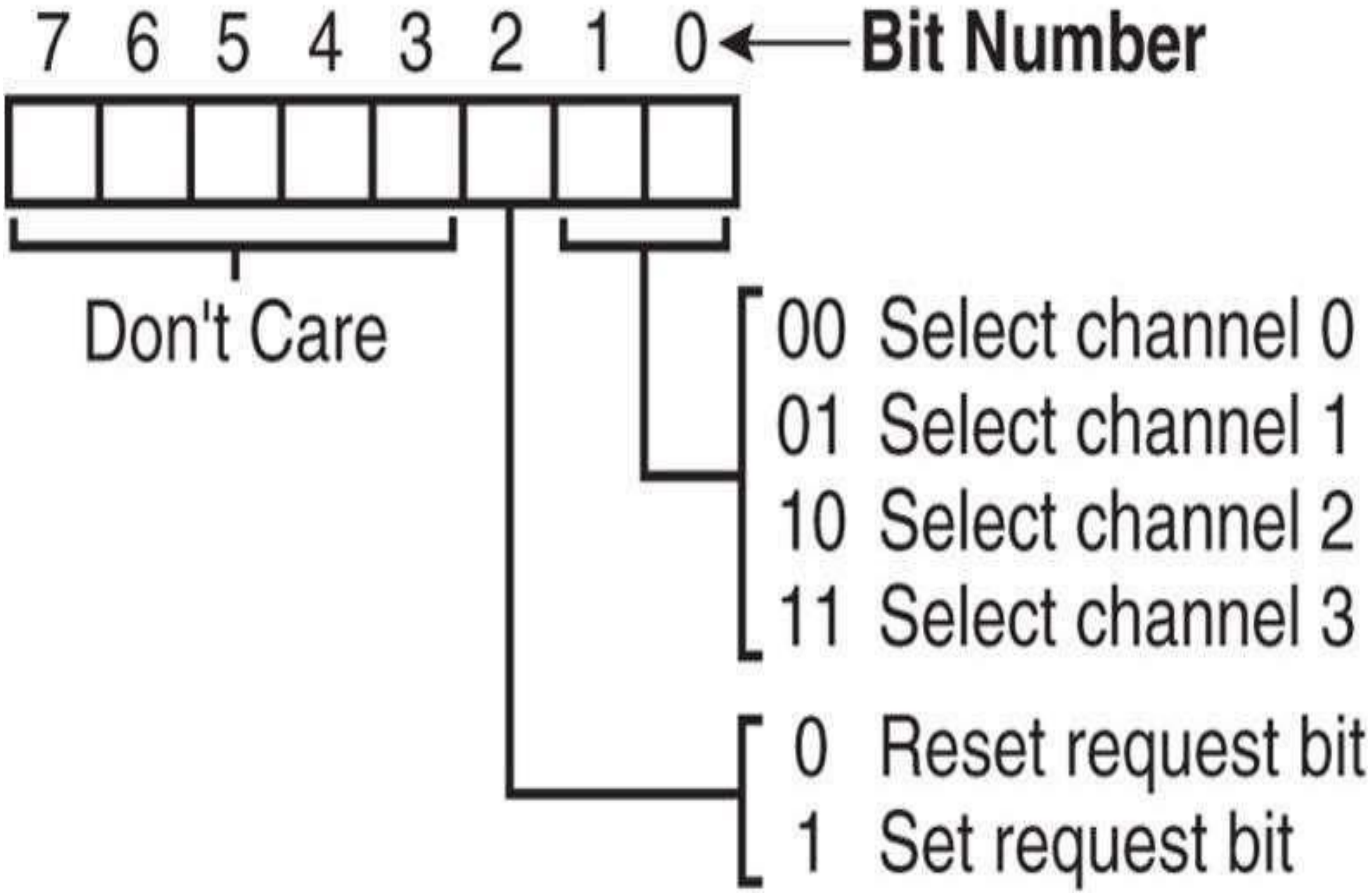
00 Demand mode select
01 Single mode select
10 Block mode select
11 Cascade mode select

RR(Request Register)

- The request register is used to request a DMA transfer via software.
- very useful in memory-to-memory transfers, where an

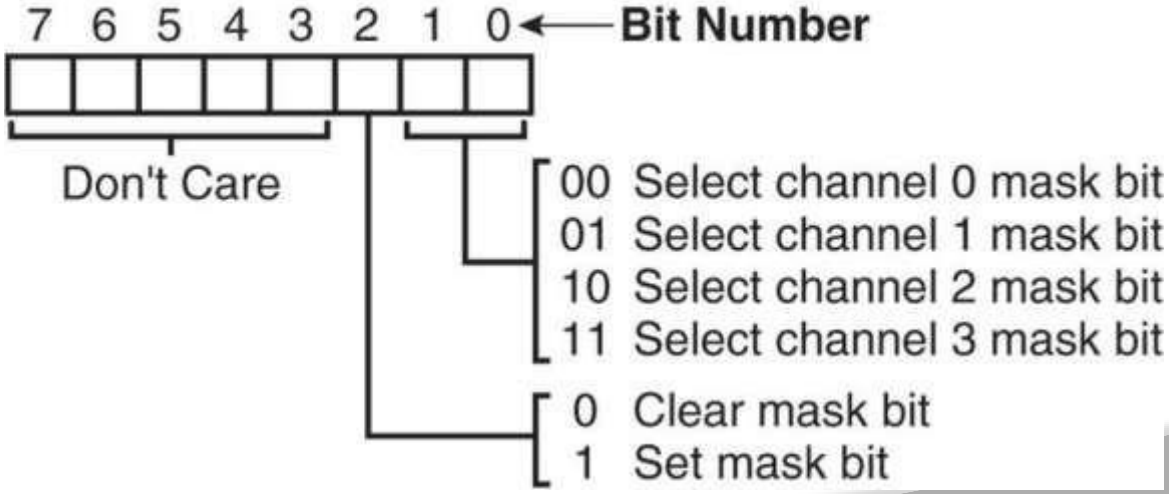
  external signal is not available to begin the DMA transfer

Request Register



MR(Mask Register)

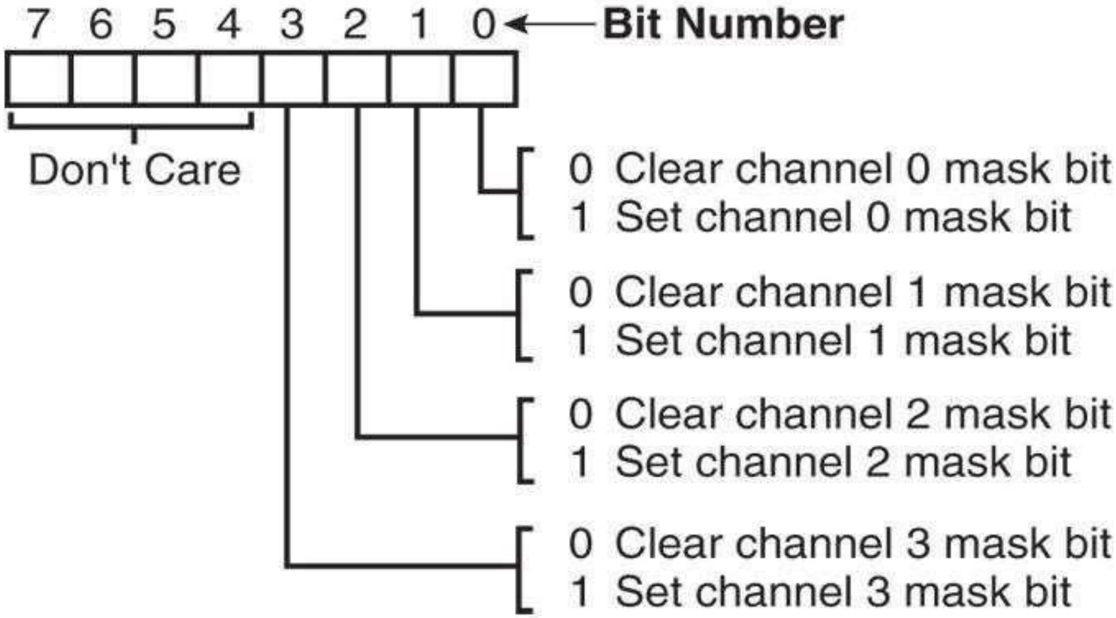
- The mask register set/reset sets or clears the channel mask.
- if the mask is set, the channel is disabled.
- The RESET signal sets all channel masks to disable them



7 6 5 4 3 2 1 0
Bit Number
Don't Care
00 Select channel 0 mask bit
01 Select channel 1 mask bit
10 Select channel 2 mask bit
11 Select channel 3 mask bit
0 Clear mask bit
1 Set mask bit

MSR

The mask register clears or sets all of the masks with one command instead of individual channels, as with the MRSR.



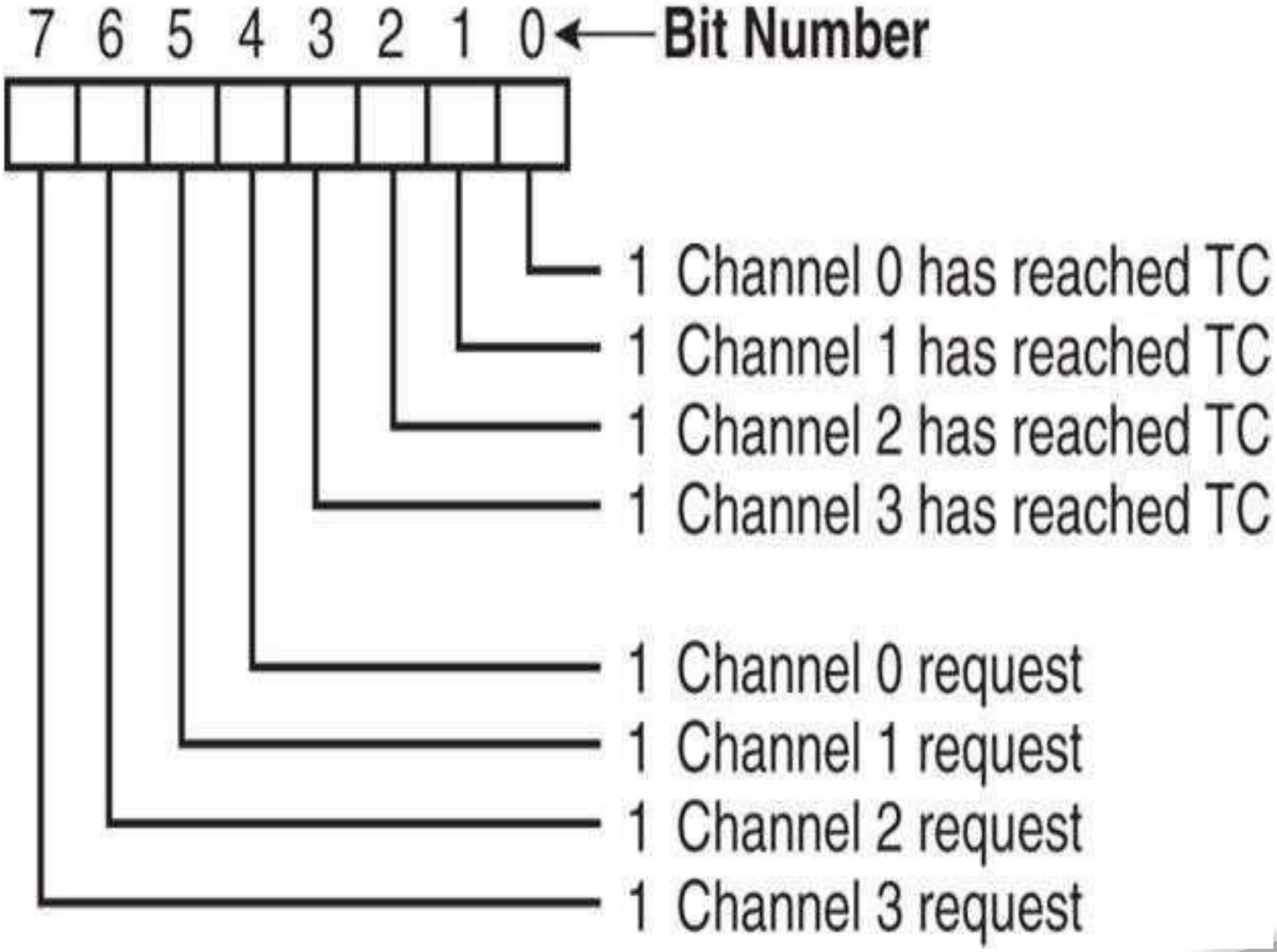
SR(Status Register)

- The status register shows status of each DMA channel. The TC bits indicate if the channel has reached its terminal count (transferred all its bytes).

- When the terminal count is reached, the DMA transfer is terminated for most modes
  of operation.

- The request bits indicate whether the DREQ input for a given channel is active.

Status Register



7 6 5 4 3 2 1 0 ← Bit Number

1 Channel 0 has reached TC

1 Channel 1 has reached TC

1 Channel 2 has reached TC

1 Channel 3 has reached TC

1 Channel 0 request

1 Channel 1 request

1 Channel 2 request

1 Channel 3 request

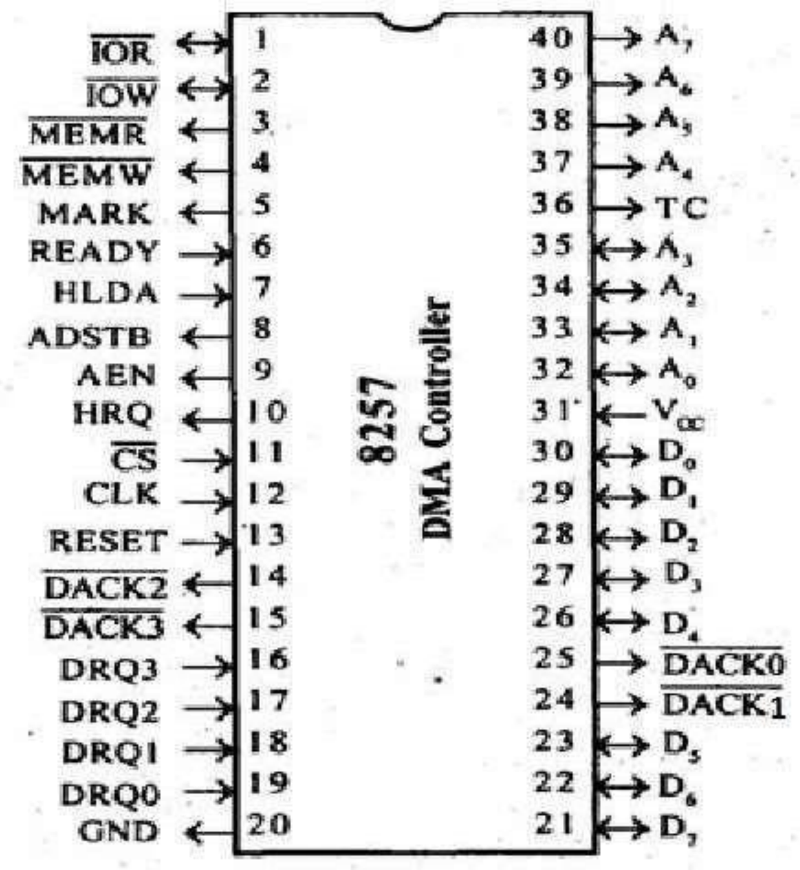
DMA Controller-8257

Features of 8257

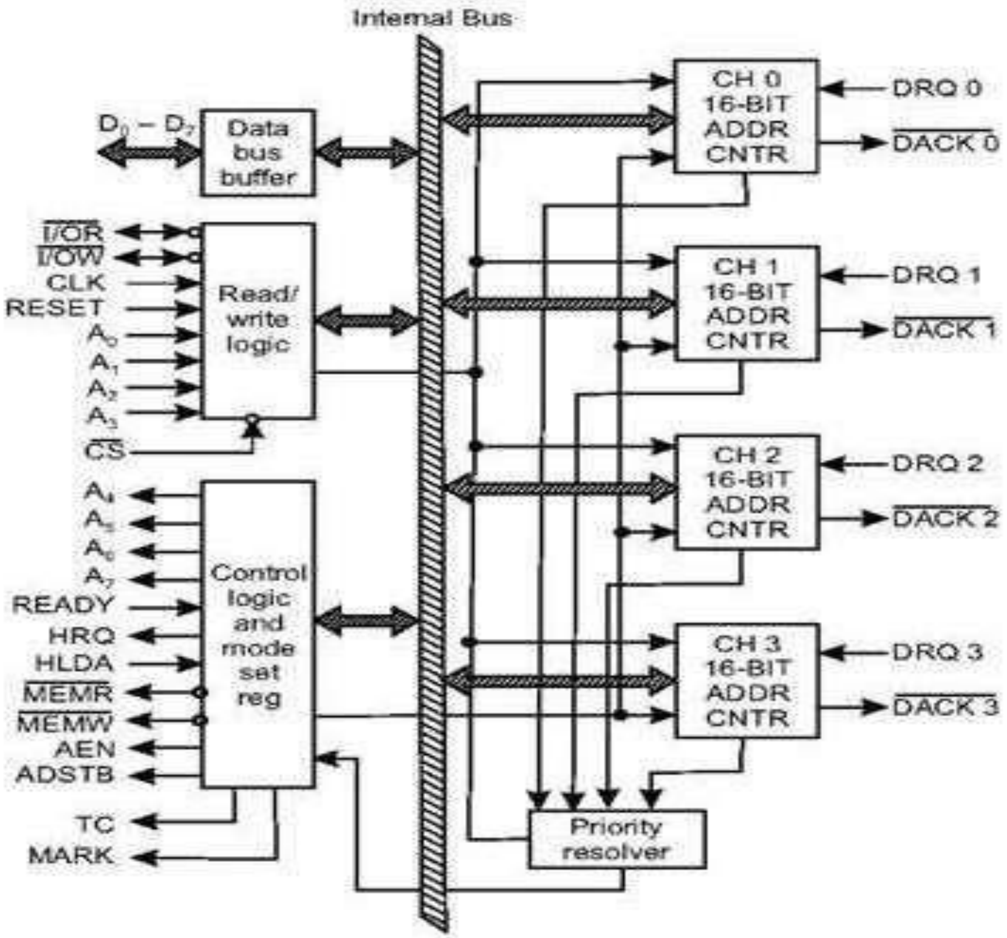
- Here is a list of some of the prominent features of 8257 –
- It has four channels which can be used over four I/O devices.
- Each channel has 16-bit address and 14-bit counter.
- Each channel can transfer data up to 64kb.
- Each channel can be programmed independently.
- Each channel can perform read transfer, write transfer and verify transfer operations.
- It generates MARK signal to the peripheral device that 128 bytes
  have
- been transferred.
- It requires a single phase clock.
- Its frequency ranges from 250Hz to 3MHz.

8257 Pin Description

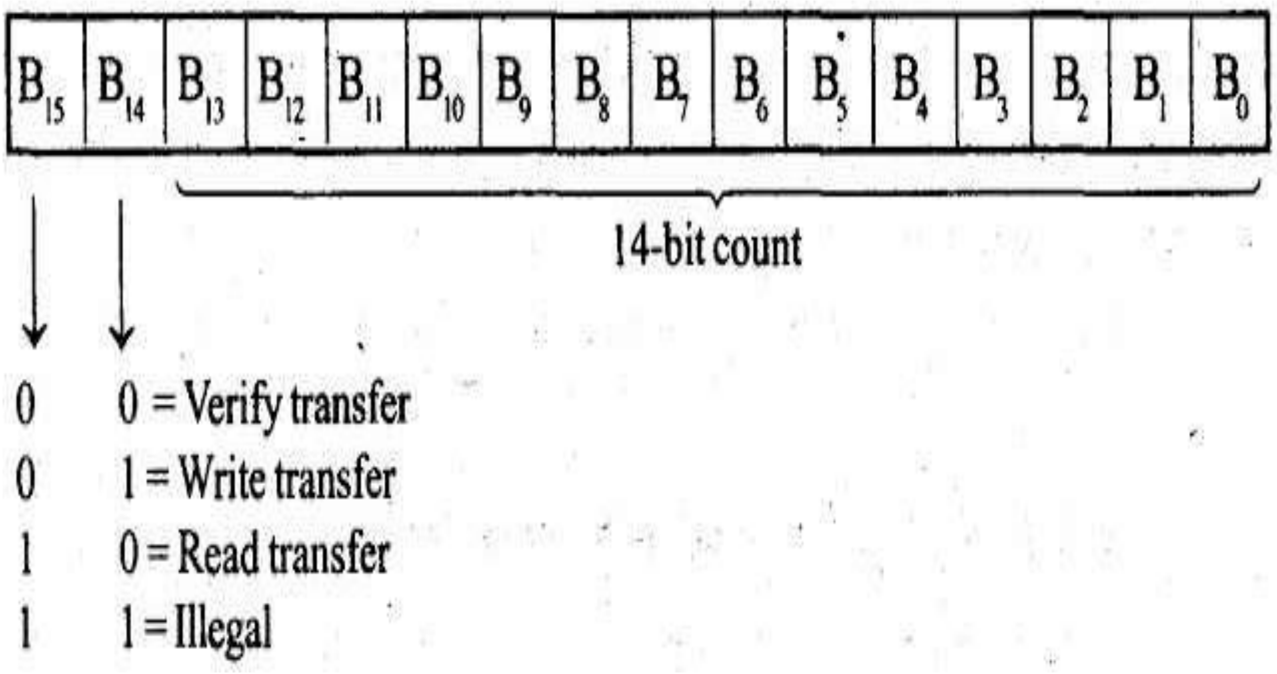
- The following image shows the pin diagram of a 8257 DMA controller



Block Diagram of 8257



Terminal Count Register:


B_{15} B_{14} B_{13} B_{12} B_{11} B_{10} B_9 B_8 B_7 B_6 B_5 B_4 B_3 B_2 B_1 B_0
14-bit count

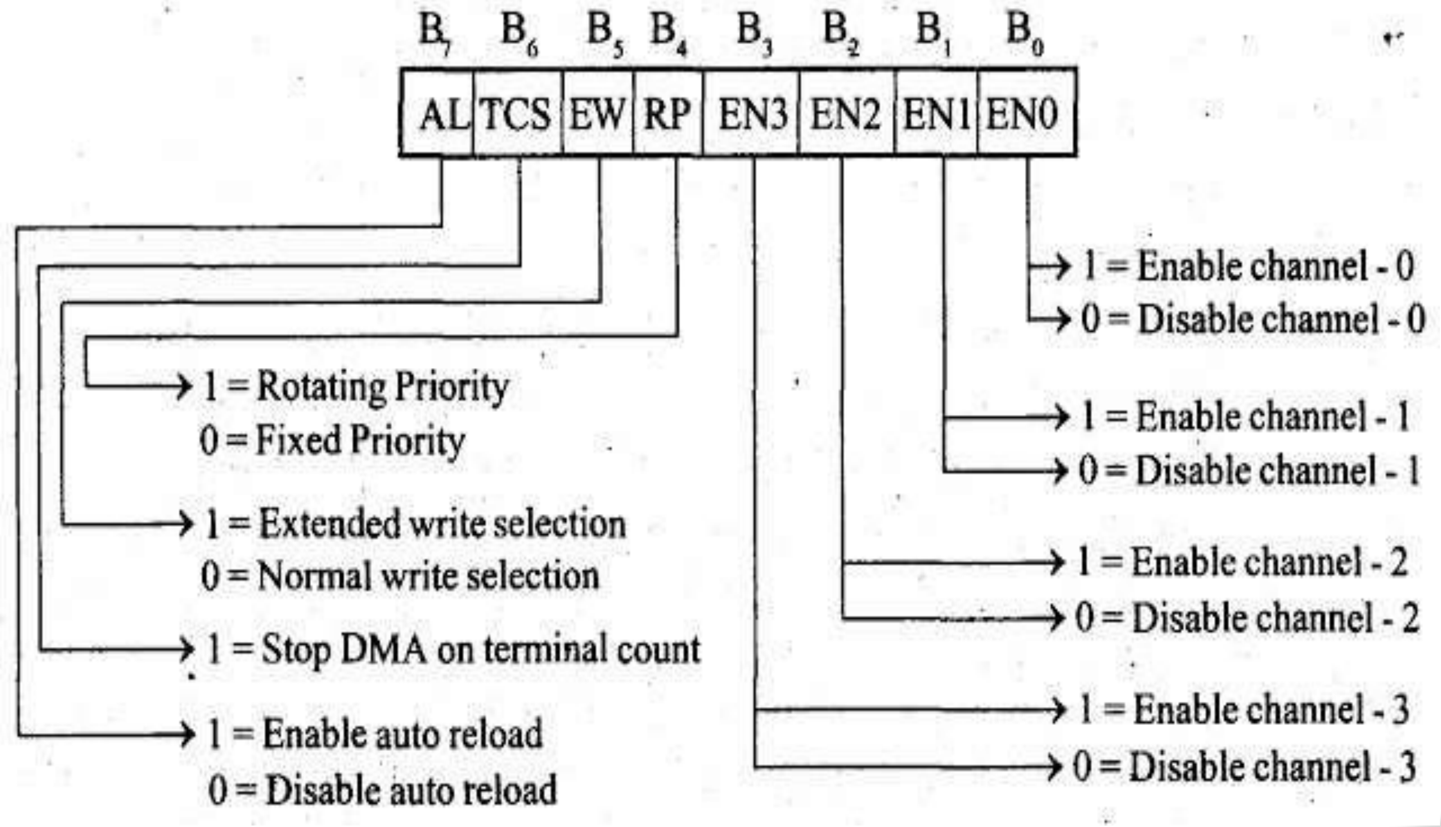
0 0 = Verify transfer

0 1 = Write transfer

1 0 = Read transfer

1 1 = Illegal

Mode Set Register:



B_7	B_6	B_5	B_4	B_3	B_2	B_1	B_0
AL	TCS	EW	RP	EN3	EN2	EN1	EN0

1 = Rotating Priority
0 = Fixed Priority

1 = Extended write selection
0 = Normal write selection

1 = Stop DMA on terminal count

1 = Enable auto reload
0 = Disable auto reload

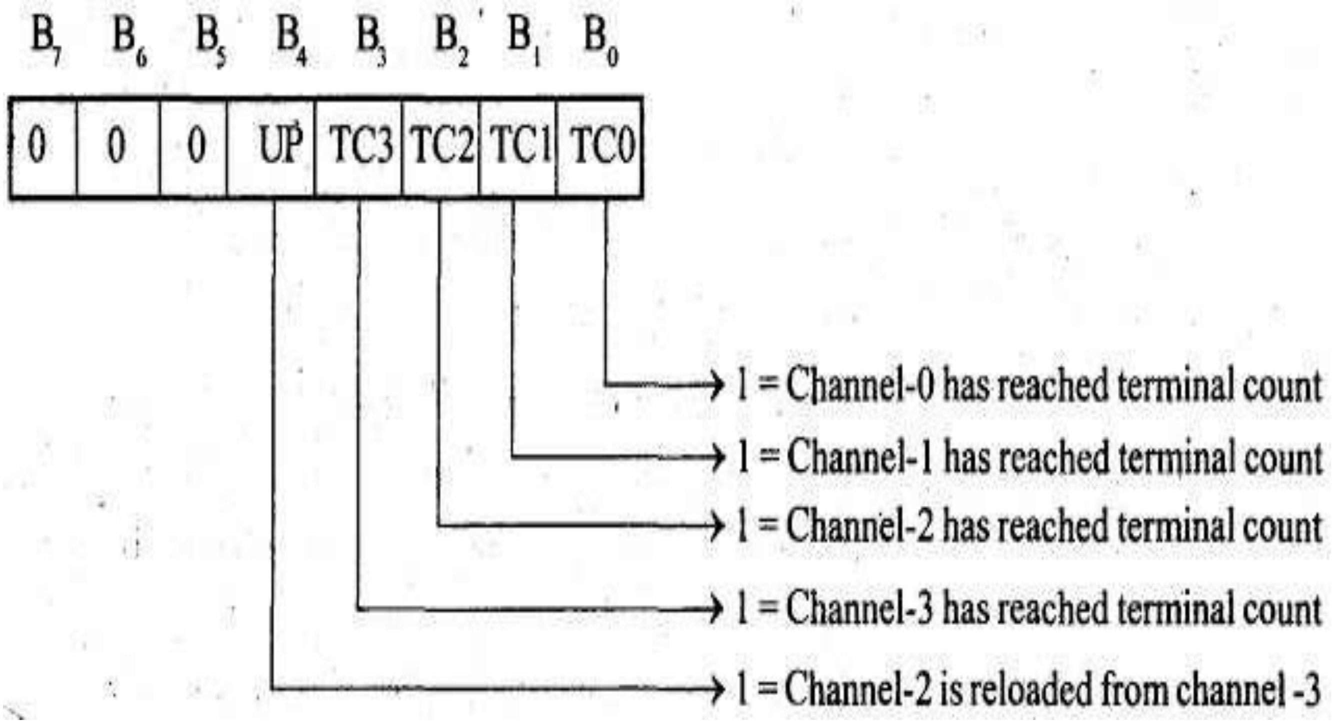
1 = Enable channel - 0
0 = Disable channel - 0

1 = Enable channel - 1
0 = Disable channel - 1

1 = Enable channel - 2
0 = Disable channel - 2

1 = Enable channel - 3
0 = Disable channel - 3

Status Register:



B_7	B_6	B_5	B_4	B_3	B_2	B_1	B_0
0	0	0	UP	TC3	TC2	TC1	TC0

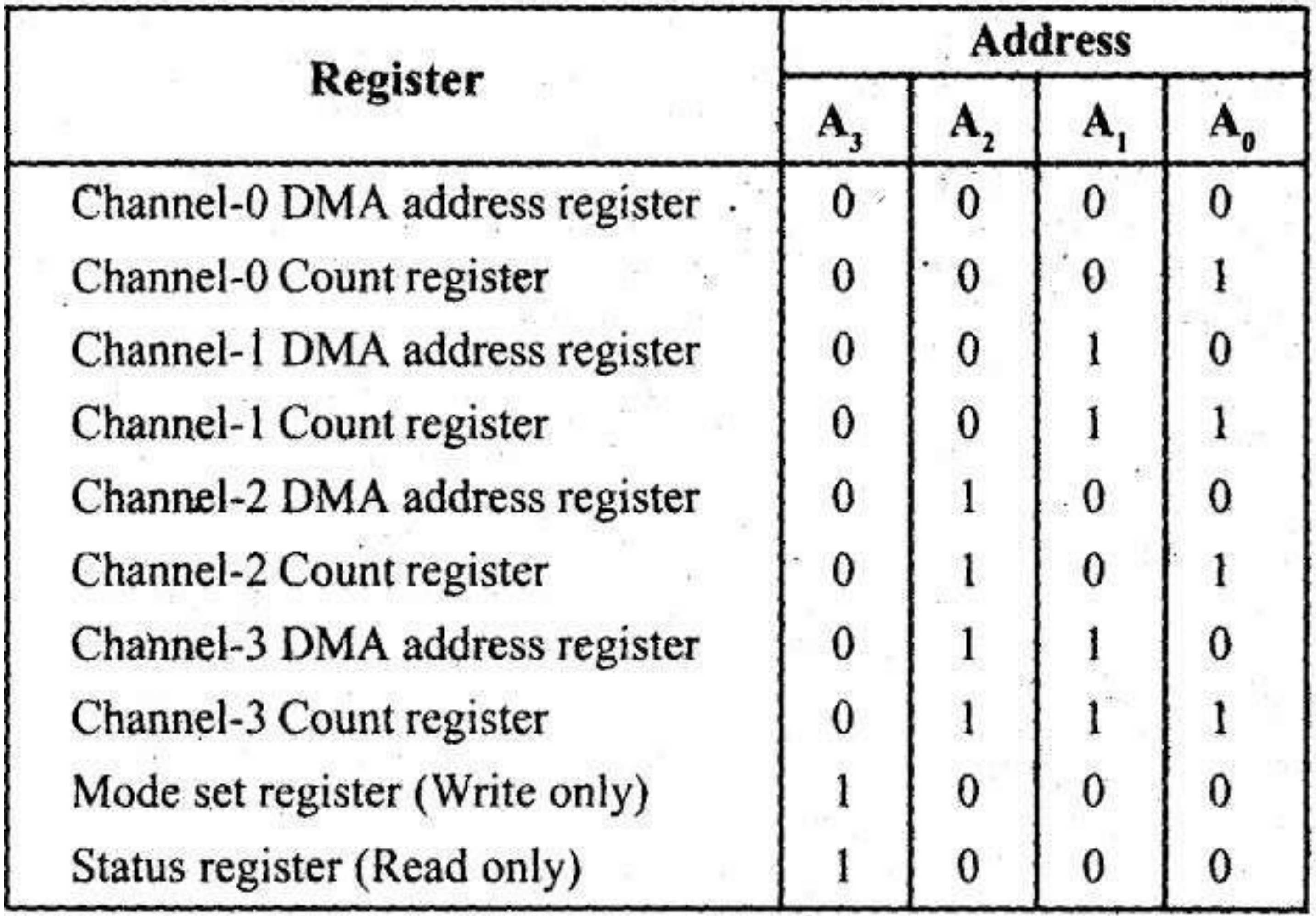
1 = Channel-0 has reached terminal count

1 = Channel-1 has reached terminal count

1 = Channel-2 has reached terminal count

1 = Channel-3 has reached terminal count

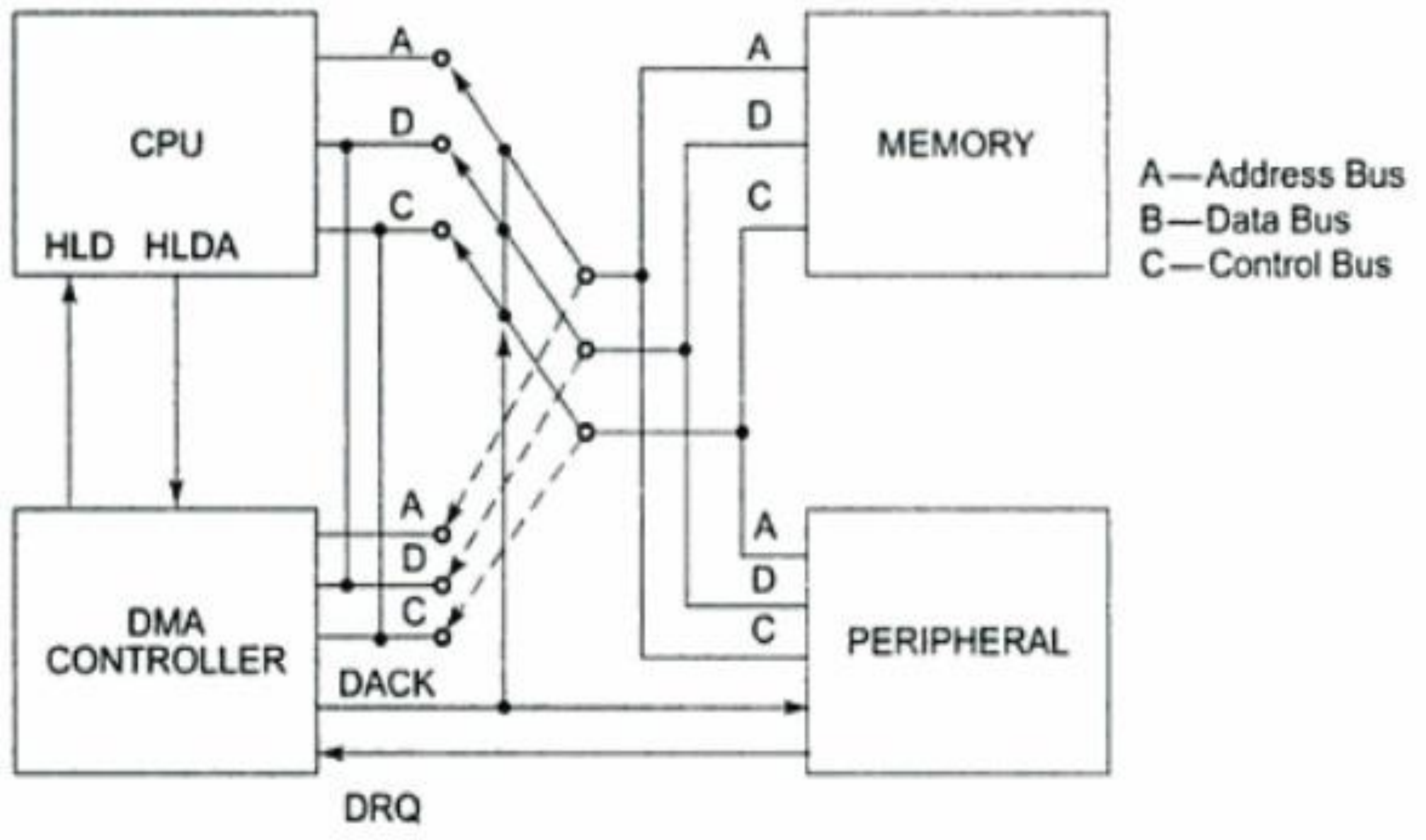
1 = Channel-2 is reloaded from channel -3



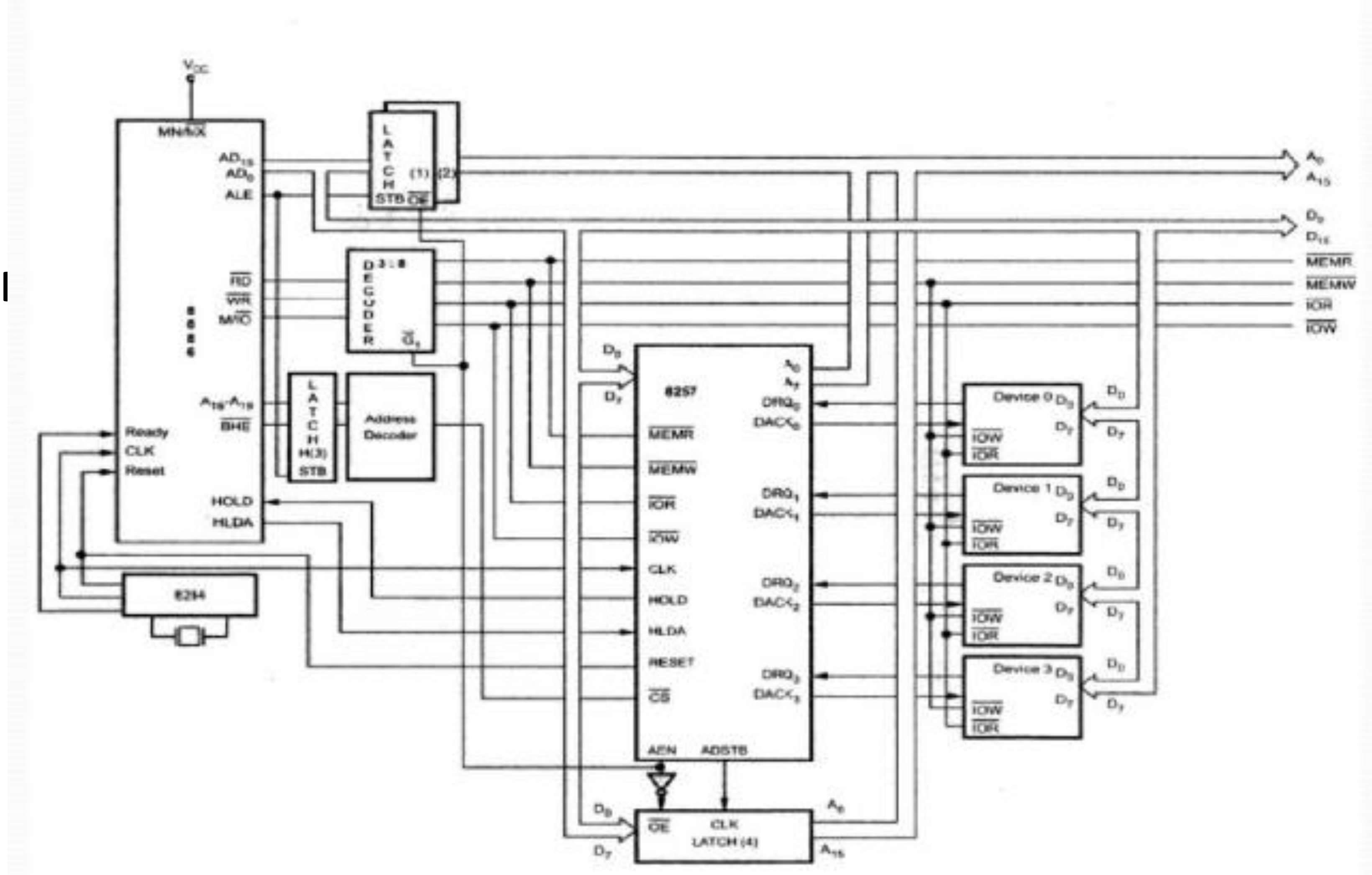
Register	Address			
	A_3	A_2	A_1	A_0
Channel-0 DMA address register	0	0	0	0
Channel-0 Count register	0	0	0	1
Channel-1 DMA address register	0	0	1	0
Channel-1 Count register	0	0	1	1
Channel-2 DMA address register	0	1	0	0
Channel-2 Count register	0	1	0	1
Channel-3 DMA address register	0	1	1	0
Channel-3 Count register	0	1	1	1
Mode set register (Write only)	1	0	0	0
Status register (Read only)	1	0	0	0

Interfacing with 8237/8257

Int


CPU
HLD
HLDA
A
D
C
MEMORY
A—Address Bus
B—Data Bus
C—Control Bus
A
D
C
DMA
CONTROLLER
DACK
DRQ
PERIPHERAL

Il



VCC
MN/MX
AD15
AD0
ALE
RD
WR
M/IO
8086
A16-A19
BHE
Ready
CLK
Reset
HOLD
HLDA
8284
LATCH
(1) (2)
STB OE
D 3 : 8
DECODER
G1
LATCH (3)
STB
Address Decoder
A0
A15
D0
D15
MEMR
MEMW
IOR
IOW
D0
D7
8257
MEMR
MEMW
IOR
IOW
CLK
HOLD
HLDA
RESET
CS
A0
A7
DRQ0
DACK0
DRQ1
DACK1
DRQ2
DACK2
DRQ3
DACK3
AEN
ADSTB
D0
D7
OE
CLK
LATCH (4)
A8
A15
Device 0
Device 1
Device 2
Device 3
D0
D7
IOW
IOR

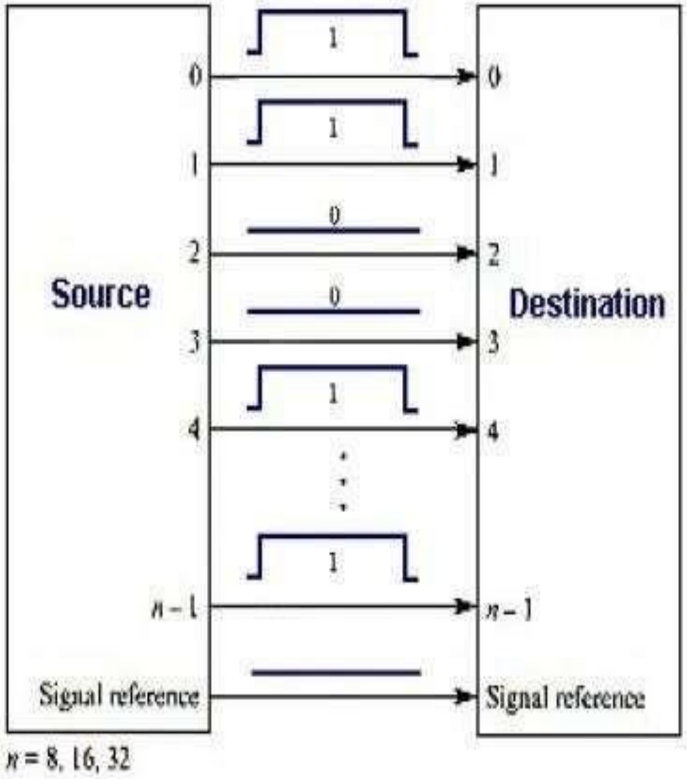
- Once a DMA controller is initialised by a CPU property, it is ready to take control of the system bus on a DMA request, either from a peripheral or itself (in case of memory-to-memory transfer).
- The DMA controller sends a HOLD request to the CPU and waits for the CPU to assert the HLDA signal. The CPU relinquishes the control of the bus before asserting the HLDA signal.
- Once the HLDA signal goes high, the DMA controller activates the DACK signal to the requesting peripheral and gains the control of the system bus. The DMA controller is the sole master of the bus, till the DMA operation is over. The CPU remains in the HOLD status (all of its signals are tristate except HOLD and HLDA), till the DMA controller is the master of the bus.

SERIAL DATA TRANSFER SCHEMES

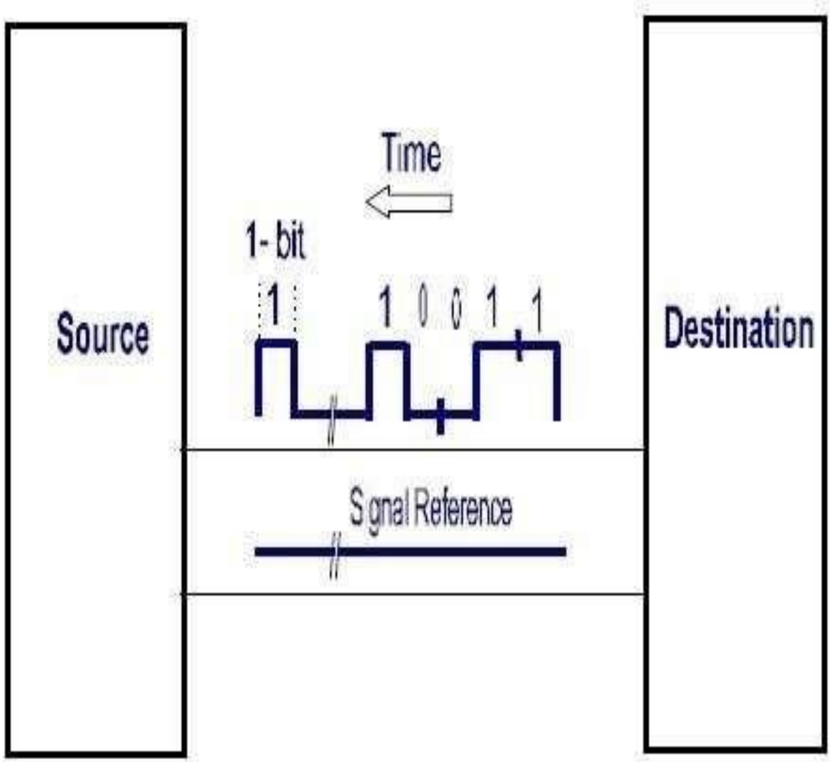
Data Transfer Schemes

Asynchronous and synchronous data transfer schemes

Data Transfer Schemes



Parallel Transmission



Serial Transmission

Data Transfer Schemes

- Even in shorter distance communications, serial computer buses are becoming more common because of a tipping point where the disadvantages of parallel busses (clock skew, interconnect density) outweigh their advantage of simplicity.
- The serial port on your PC is a full-duplex device meaning that it can send and receive data at the same time. In order to be able to do this, it uses separate lines for transmitting and receiving data.

Data Transfer Schemes

Advantages of serial communications:

- Requires fewer interconnecting cables and hence occupies less space.
- "Cross talk" is less of an issue, because there are fewer conductors compared to that of parallel communication cables.
- Many IC s and peripheral devices have serial interfaces.
- Clock skew between different channels is not an issue.
- Cheaper to implement.

Data Transfer Schemes

- **SERIAL DATA TRANSMISSION MODES**

  When data is transmittedbetween two pieces ofequipment, three communication modes of operation can be used.

- **Simplex:** In a simple connection, data is transmitted in one direction only. For example, from a computer to printer that cannot send status signals back to the computer.

- **Half-duplex:**In a half-duplex connection,two-way transfer of data is possible, but only in one direction at a time.

- **Full duplex:** In a full-duplex configuration, both ends can send and receive data simultaneously, which technique is common in our PCs.

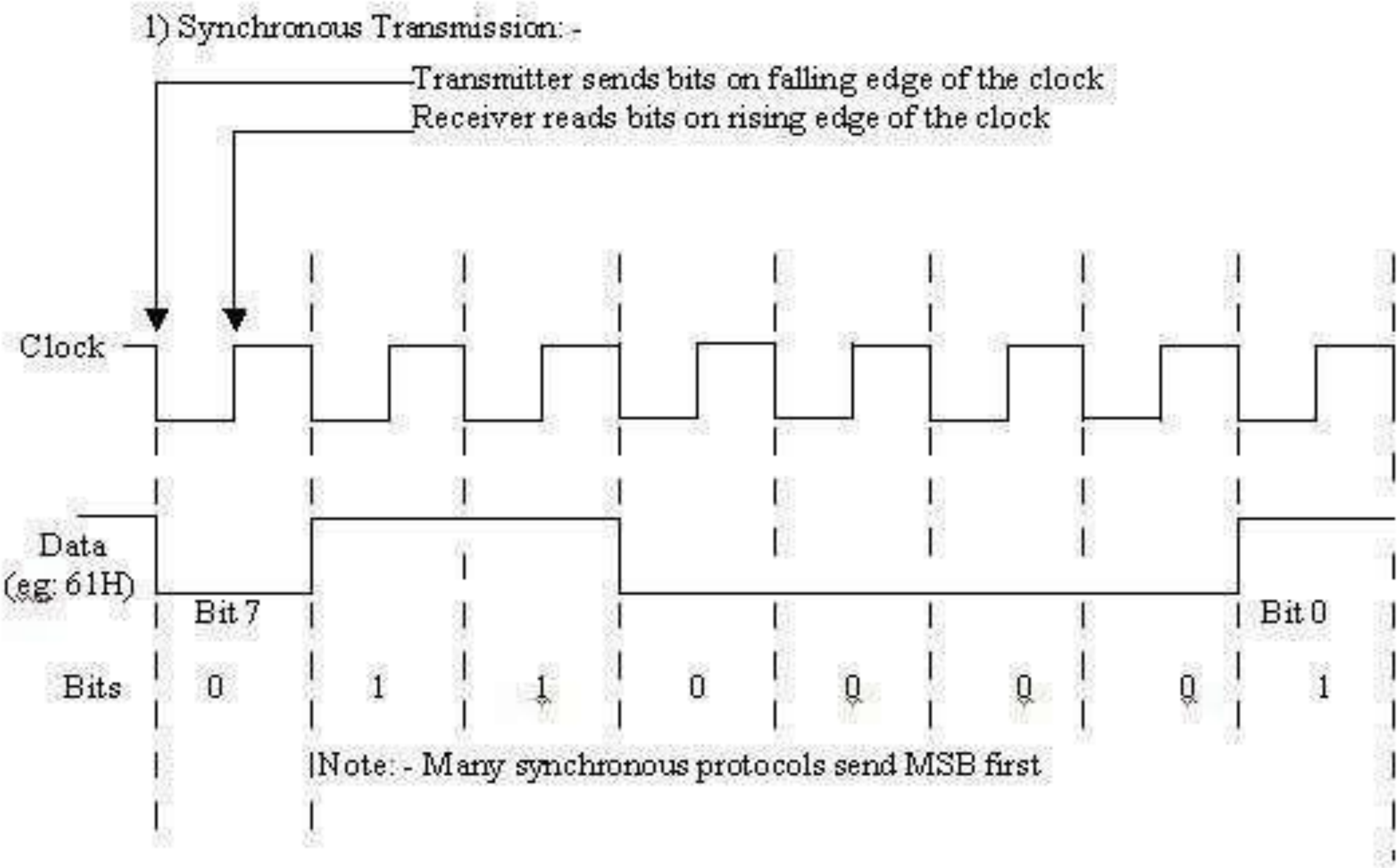
Data Transfer Schemes

- **SERIAL DATA TRANSFER SCHEMS**

- There are two ways to synchronize the two ends of the communication.

  - Synchronous data transmission
  - Asynchronous data transmission

Synchronous Data Transmission


1) Synchronous Transmission: -
Transmitter sends bits on falling edge of the clock
Receiver reads bits on rising edge of the clock
Clock
Data
(eg: 61H)
Bit 7
Bit 0
Bits
0
1
1
0
0
0
0
1
Note: - Many synchronous protocols send MSB first

Data Transfer Schemes

- The synchronous signaling methods use two different signals. A pulse on one signal line indicates when another bit of information is ready on the other signal line.

- In synchronous transmission, the stream of data to be transferred is encoded and sent on one line, and a periodic pulse of voltage which is often called the "clock" is put on another line, that tells the receiver about the beginning and the ending of each bit

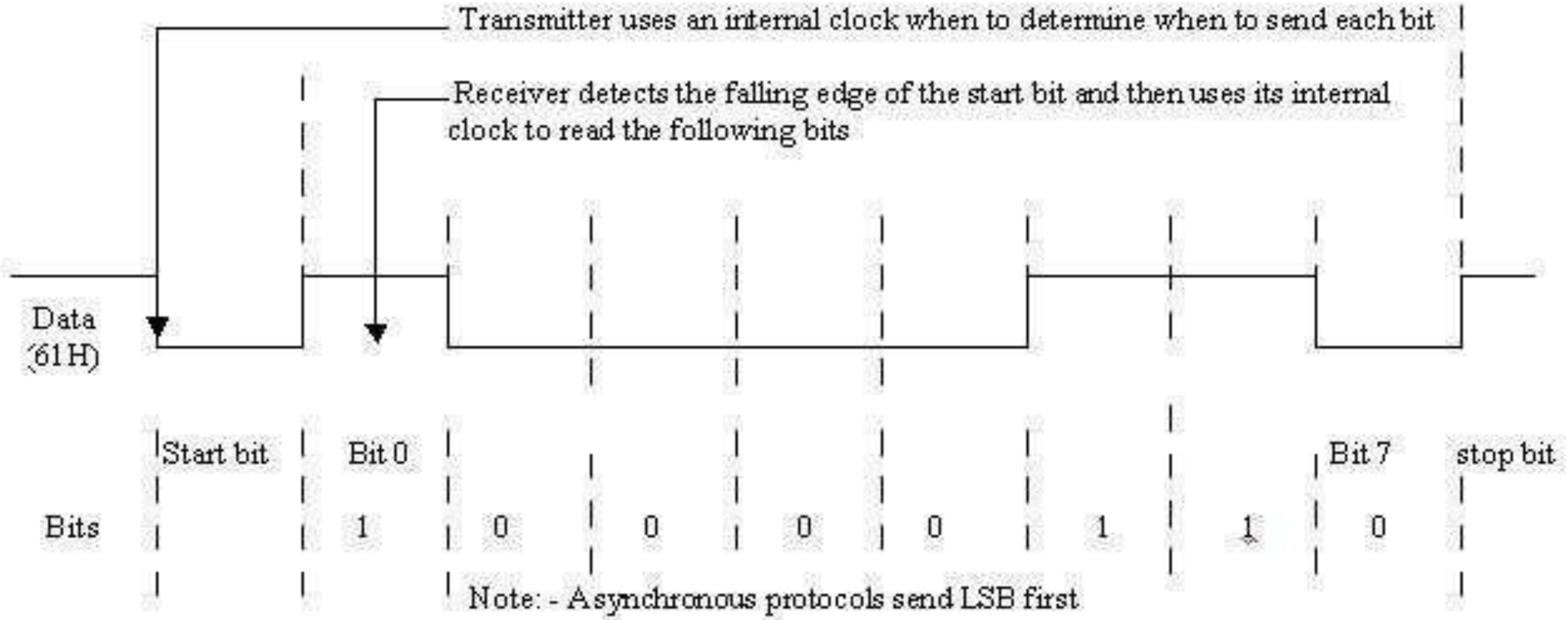
Data Transfer Schemes

- **Advantages:** The only advantage of synchronous data transfer is the Lower overhead and thus, greater throughput, compared to asynchronous one.

- **Disadvantages:**
  - Slightly more complex
  - Hardware is more expensive

Data Transfer Schemes

2) Asynchronous Transmission: -



Transmitter uses an internal clock when to determine when to send each bit

Receiver detects the falling edge of the start bit and then uses its internal clock to read the following bits

Note: - Asynchronous protocols send LSB first

Data Transfer Schemes

- The asynchronous signaling methods use only one signal. The receiver uses transitions on that signal to figure out the transmitter bit rate (known as auto baud) and timing.
- A pulse from the local clock indicates when another bit is ready. That means synchronous transmissions use an external clock, while asynchronous transmissions are use special signals along the transmission medium.

Data Transfer Schemes

Asynchronous communication is the commonly prevailing communication method in the personal computer industry, due to the reason that it is easier to implement and has the unique advantage that bytes can be sent whenever they are ready, no need to wait for blocks of data to accumulate.

Data Transfer Schemes

Advantages:

- Simple and doesn't require much synchronization on both communication sides. The timing is not as critical as for synchronous transmission; therefore hardware can be made cheaper.

- Set-up is very fast, so well suited for applications where messages are generated at irregular intervals, for example data entry from the keyboard.

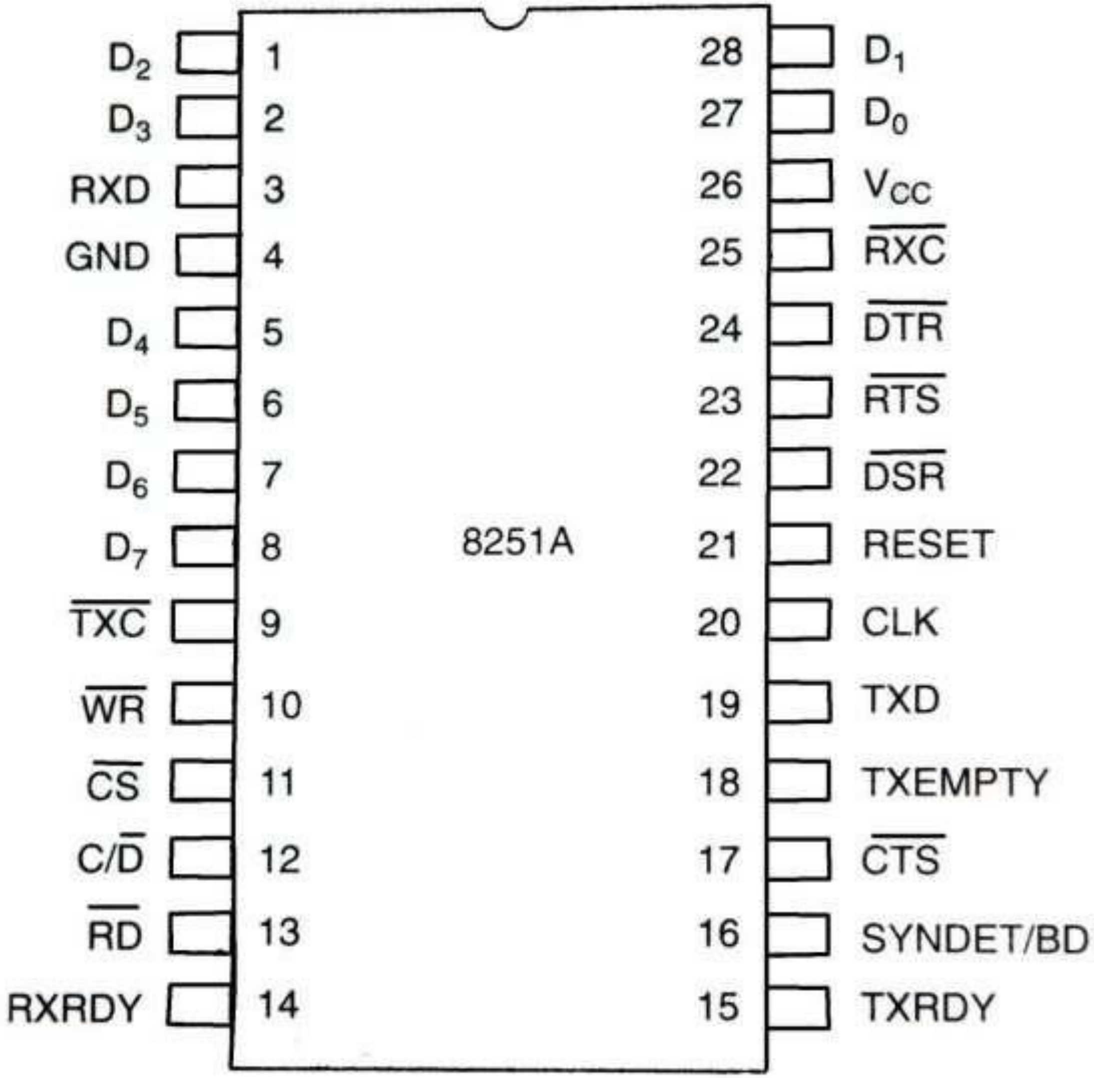
Data Transfer Schemes

Disadvantages:

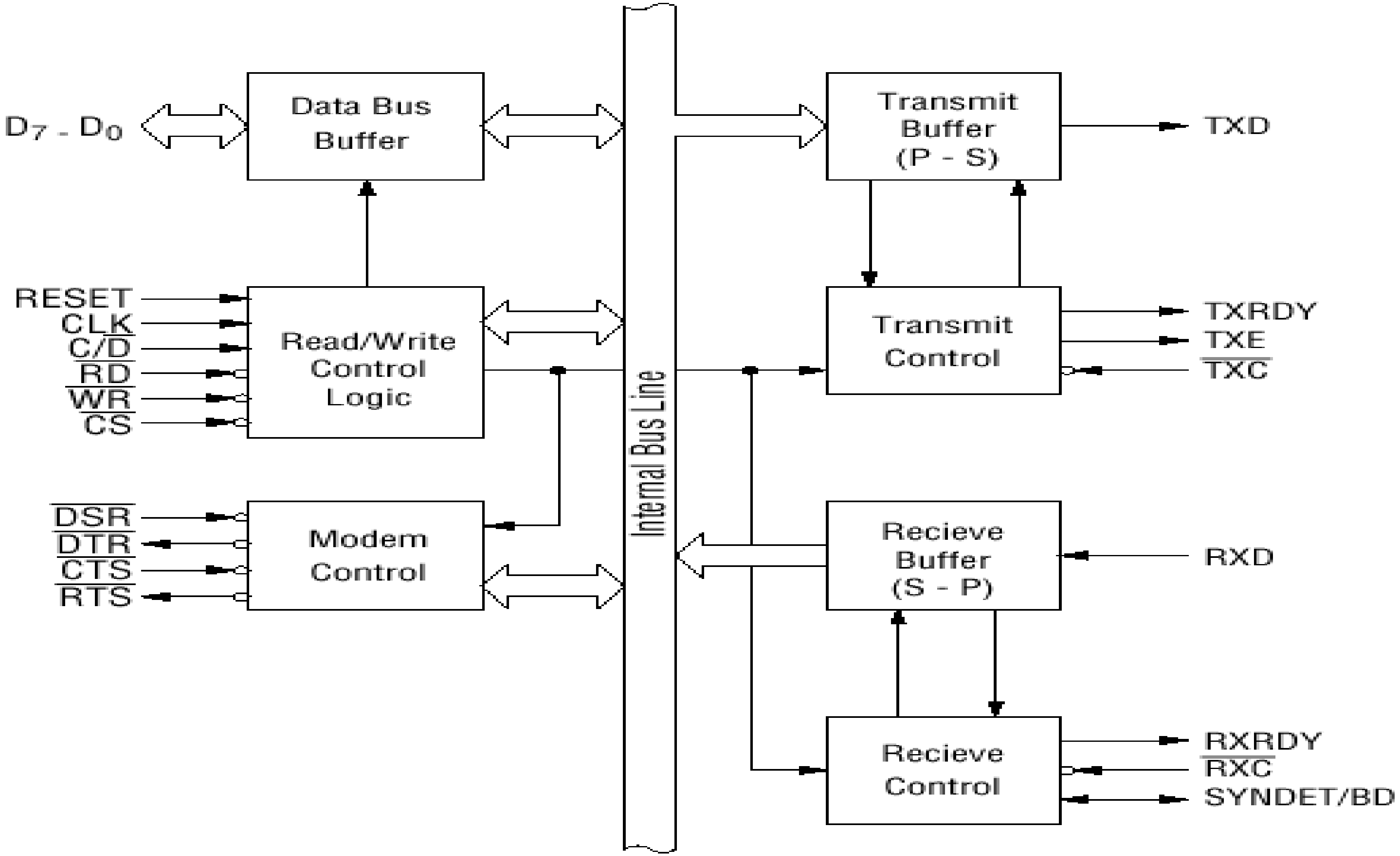
- One of the main disadvantages of asynchronous technique is the large relative overhead, where a high proportion of the transmitted bits are uniquely for control purposes and thus carry no useful information.

8251 USART architecture and interfacing

Pin diagram of 8251



Block diagram of 8251



Sections of 8251A

- Data Bus buffer
- Read/Write Control Logic
- Modem Control
- Transmitter
- CS – Chip Select
- Receiver

Data Bus Buffer

D0-D7 : 8-bit data bus used to read or write status, command word or data

Read/Write Control logic

- C/D – Control/Data
- WR: When signal is low, the MPU either writes.
- RD : When signal goes low, the MPU either reads.
- RESET : A high on this signal reset 8252A.

Control Register

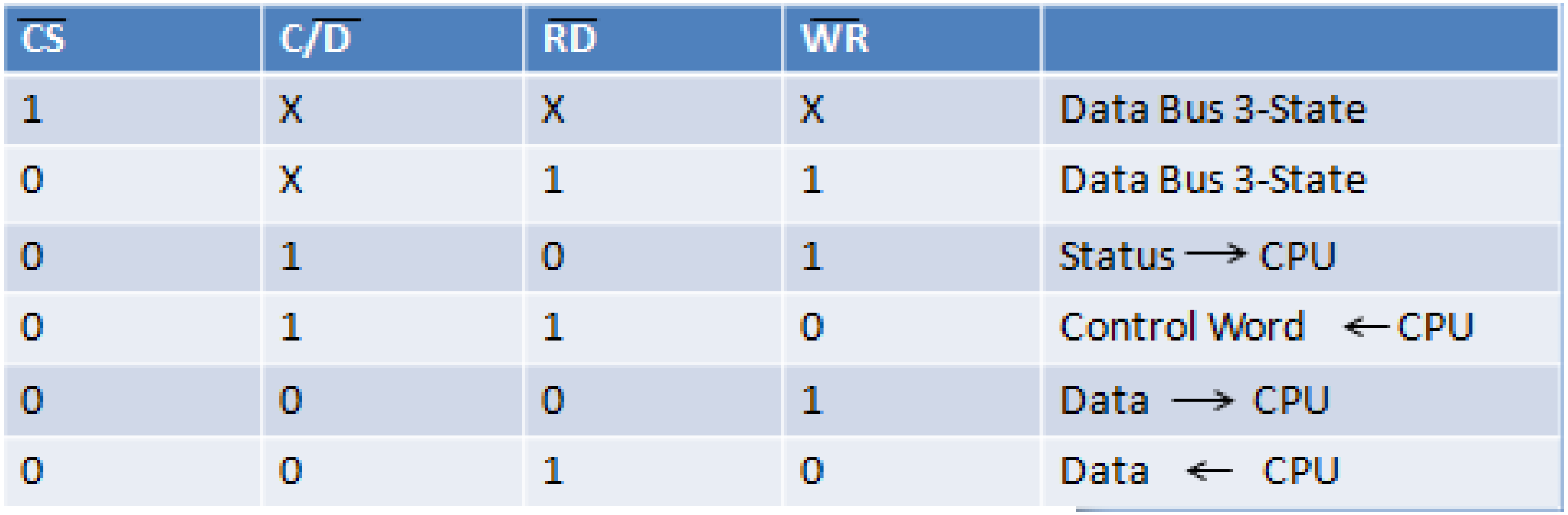
- 16-bit register for a control word consist of two independent bytes namely mode word & command word.
- Mode word : Specifies the general characteristics of operation such as baud, parity, number of bits etc.
- Command word : Enables the data transmission and reception.
- Register can be accessed as an output port when the Control/Data pin is high.

Status register

- Checks the ready status of theperipheral.
- Status word register provides the information concerning register status and transmission errors.

Dataregister

- Used as an input and output port when the C/D is low.



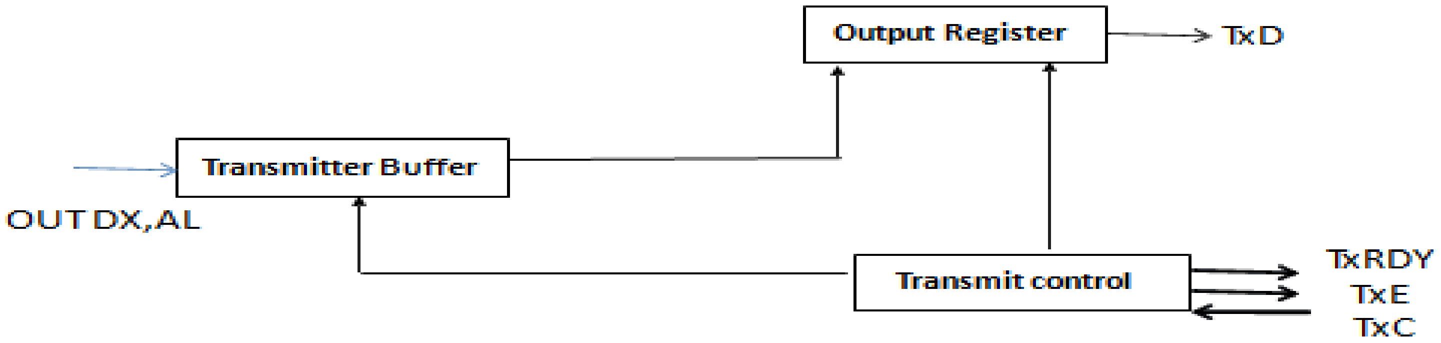
$\overline{CS}$	C/$\overline{D}$	$\overline{RD}$	$\overline{WR}$	
1	X	X	X	Data Bus 3-State
0	X	1	1	Data Bus 3-State
0	1	0	1	Status → CPU
0	1	1	0	Control Word ← CPU
0	0	0	1	Data → CPU
0	0	1	0	Data ← CPU

Modem Control

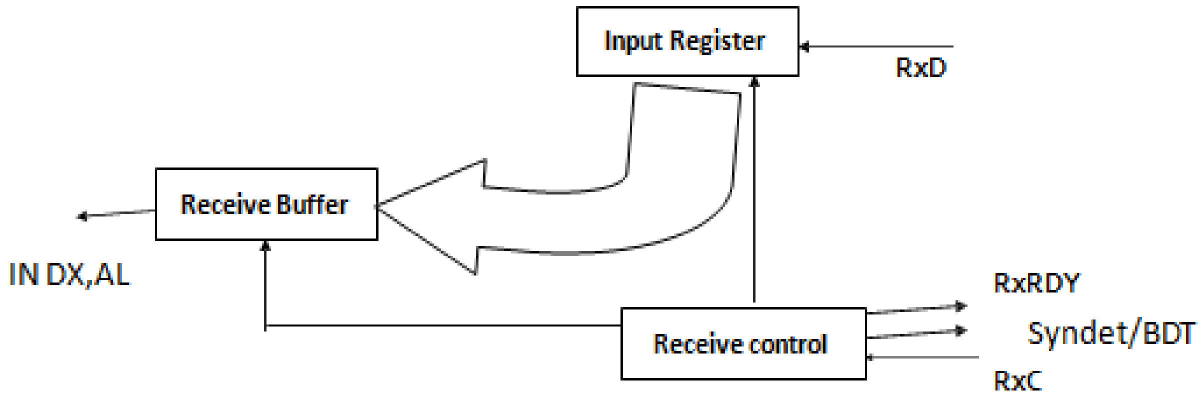
- DSR - Data Set Ready : Checks if the Data Set is ready when communicating with a modem.
- DTR - Data Terminal Ready : Indicates that the device is ready to accept data when the 8251 is communicating with a modem.
- CTS - Clear to Send : If its low, the 8251A is enabled to transmit the serial data provided the enable bit in the command byte is set to'1'.
- RTS - Request to Send Data : Low signal indicates the modem that the receiver is ready to receive a data byte from the modem.

Transmitter section

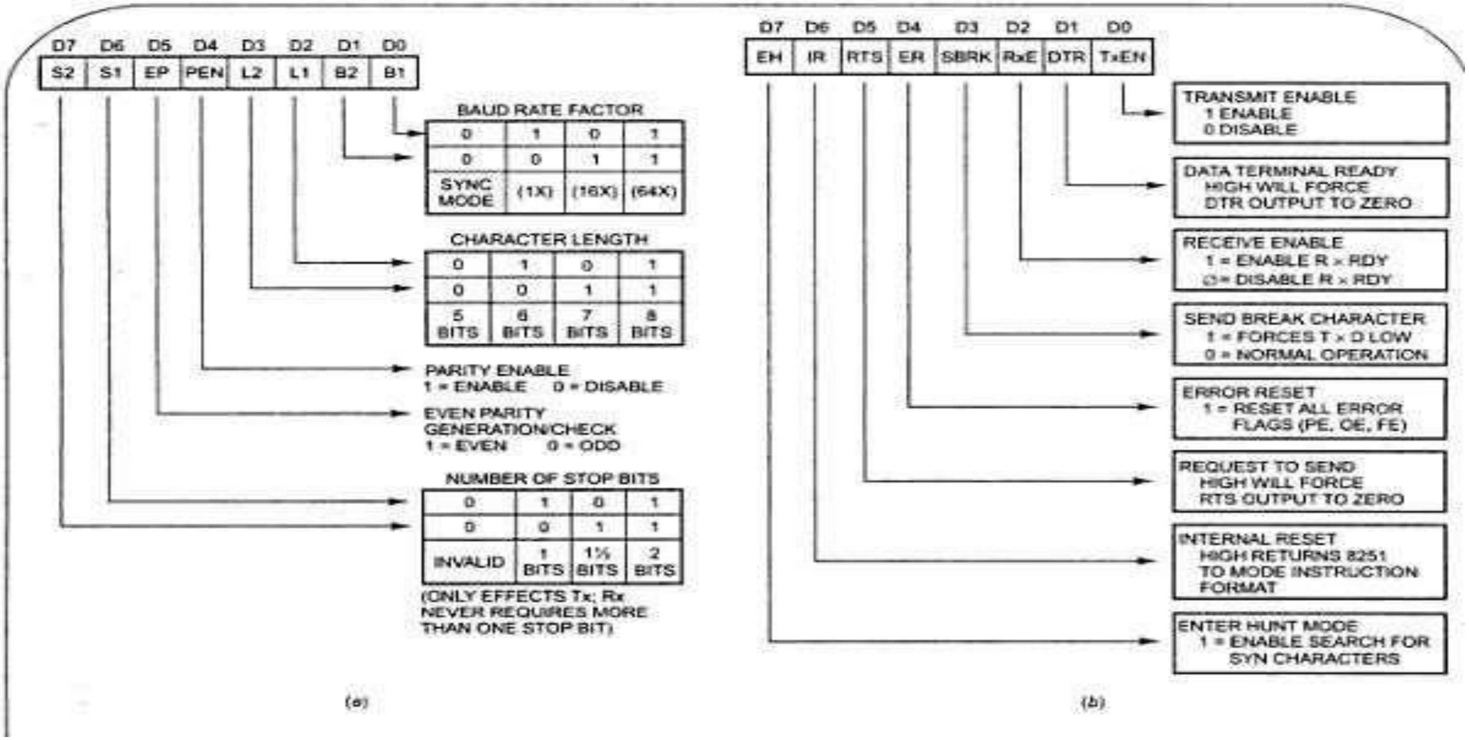
- Accepts parallel data from MPU & converts them into serial data.
- Has two registers:
- Buffer register : To hold eight bits
- Output register : To convert eight bits into a stream of serial bits.



ReceiverSection



Mode word & command word for 8251



D7	D6	D5	D4	D3	D2	D1	D0
S2	S1	EP	PEN	L2	L1	B2	B1

BAUD RATE FACTOR

0	1	0	1
0	0	1	1
SYNC MODE	(1X)	(16X)	(64X)

CHARACTER LENGTH

0	1	0	1
0	0	1	1
5 BITS	6 BITS	7 BITS	8 BITS

PARITY ENABLE
1 = ENABLE 0 = DISABLE

EVEN PARITY
GENERATION/CHECK
1 = EVEN 0 = ODD

NUMBER OF STOP BITS

0	1	0	1
0	0	1	1
INVALID	1 BITS	1½ BITS	2 BITS

(ONLY EFFECTS Tx; Rx NEVER REQUIRES MORE THAN ONE STOP BIT)

(a)

D7	D6	D5	D4	D3	D2	D1	D0
EH	IR	RTS	ER	SBRK	RxE	DTR	TxEN

TRANSMIT ENABLE
1 ENABLE
0 DISABLE

DATA TERMINAL READY
HIGH WILL FORCE
DTR OUTPUT TO ZERO

RECEIVE ENABLE
1 = ENABLE R × RDY
∅= DISABLE R × RDY

SEND BREAK CHARACTER
1 = FORCES T × D LOW
0 = NORMAL OPERATION

ERROR RESET
1 = RESET ALL ERROR
FLAGS (PE, OE, FE)

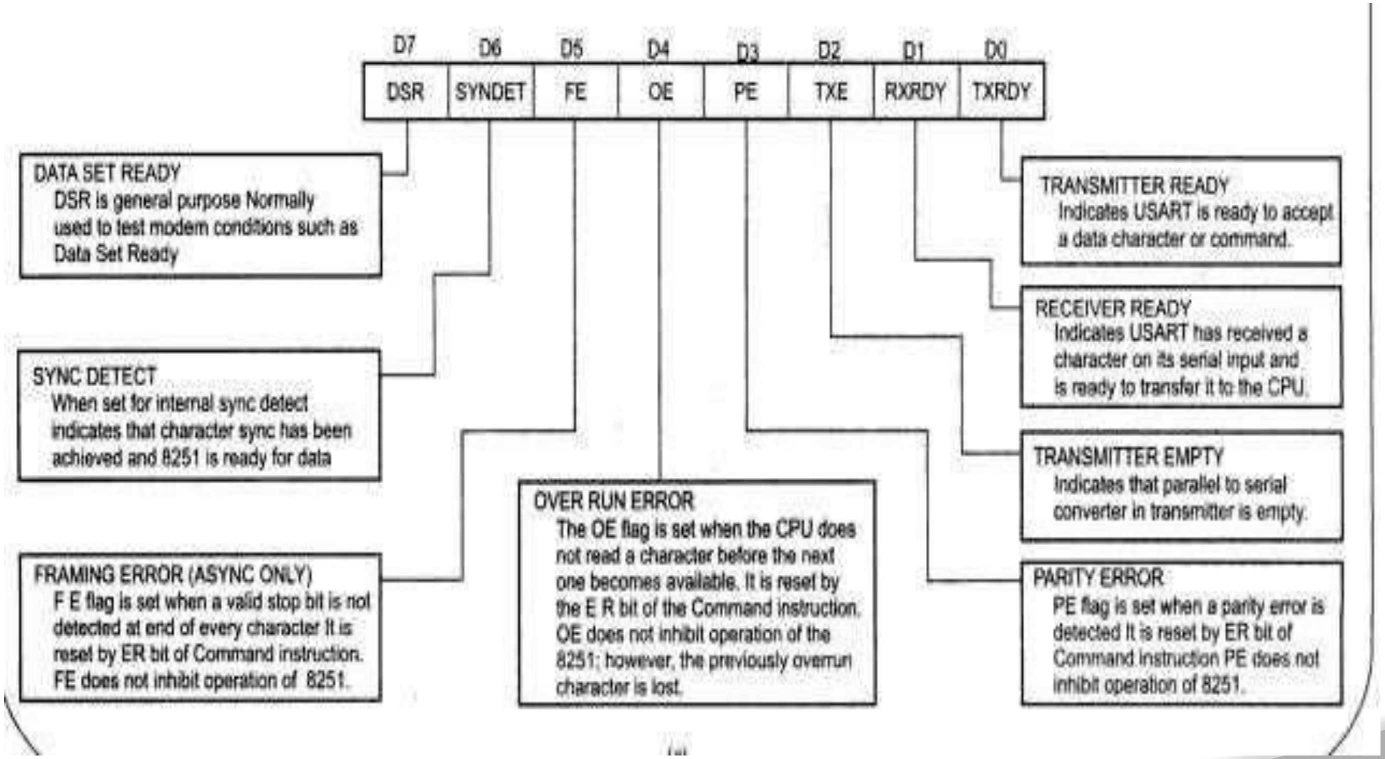
REQUEST TO SEND
HIGH WILL FORCE
RTS OUTPUT TO ZERO

INTERNAL RESET
HIGH RETURNS 8251
TO MODE INSTRUCTION
FORMAT

ENTER HUNT MODE
1 = ENABLE SEARCH FOR
SYN CHARACTERS

(b)

Status word register of 8251



D7	D6	D5	D4	D3	D2	D1	D0
DSR	SYNDET	FE	OE	PE	TXE	RXRDY	TXRDY

DATA SET READY
DSR is general purpose Normally used to test modem conditions such as Data Set Ready

SYNC DETECT
When set for internal sync detect indicates that character sync has been achieved and 8251 is ready for data

FRAMING ERROR (ASYNC ONLY)
F E flag is set when a valid stop bit is not detected at end of every character It is reset by ER bit of Command instruction. FE does not inhibit operation of 8251.

OVER RUN ERROR
The OE flag is set when the CPU does not read a character before the next one becomes available. It is reset by the E R bit of the Command instruction. OE does not inhibit operation of the 8251; however, the previously overrun character is lost.

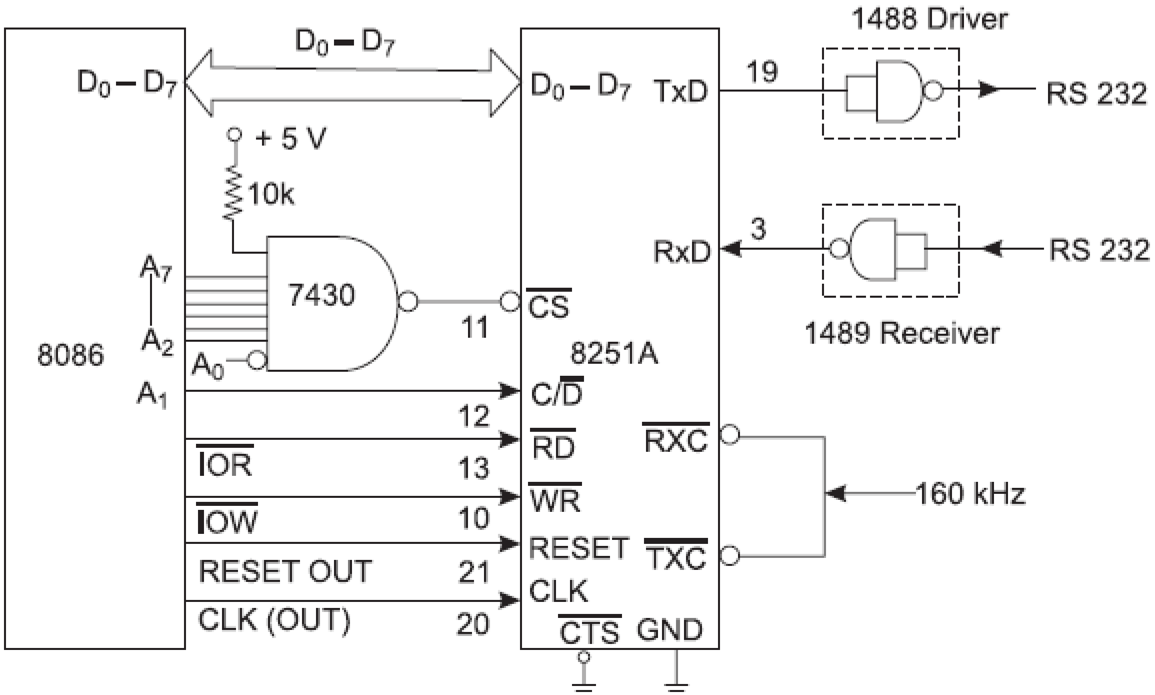
TRANSMITTER READY
Indicates USART is ready to accept a data character or command.

RECEIVER READY
Indicates USART has received a character on its serial input and is ready to transfer it to the CPU.

TRANSMITTER EMPTY
Indicates that parallel to serial converter in transmitter is empty.

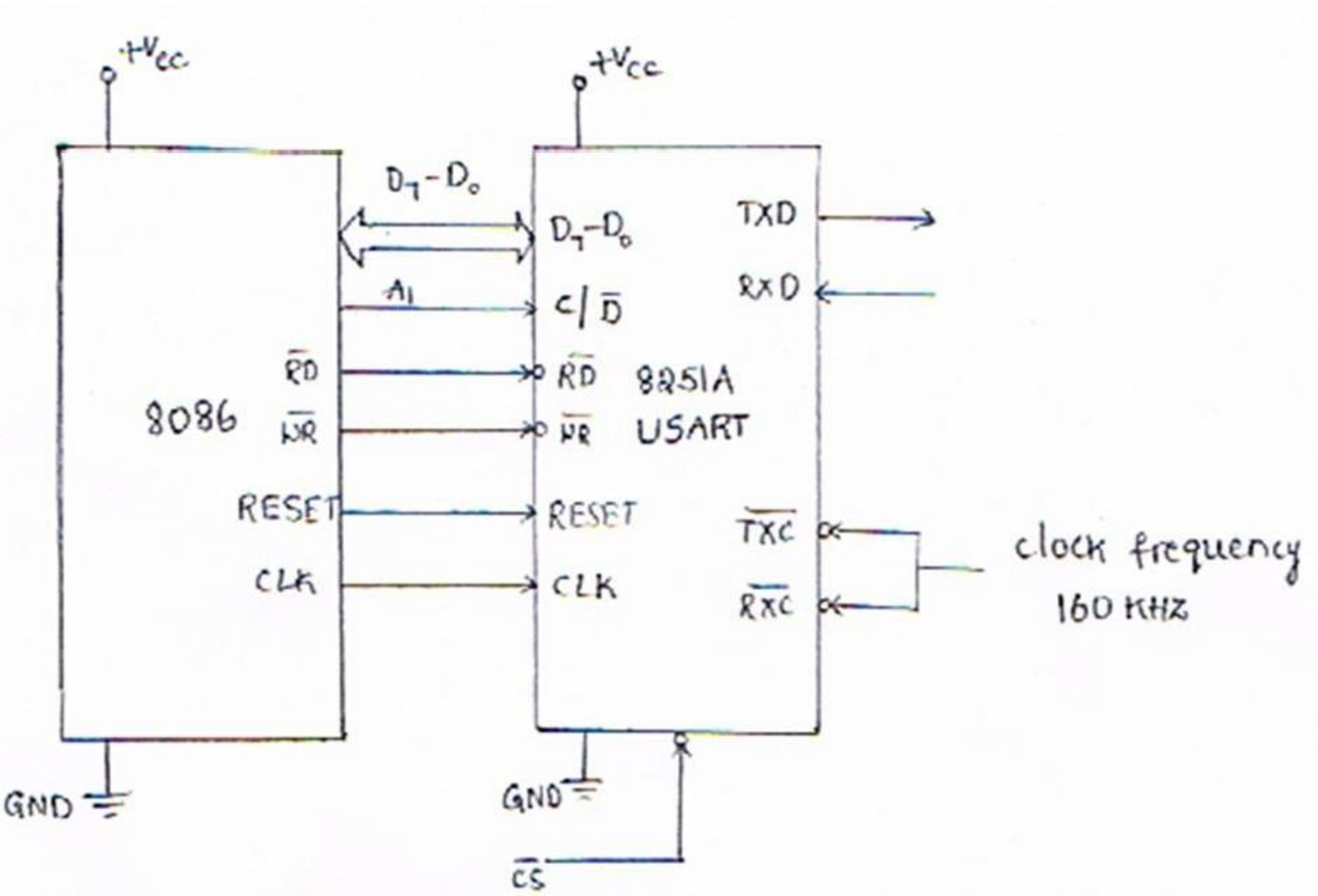
PARITY ERROR
PE flag is set when a parity error is detected It is reset by ER bit of Command Instruction PE does not inhibit operation of 8251.

interfacing 8251 USART with 8086



8251 Interfacing with 8086 microprocessor

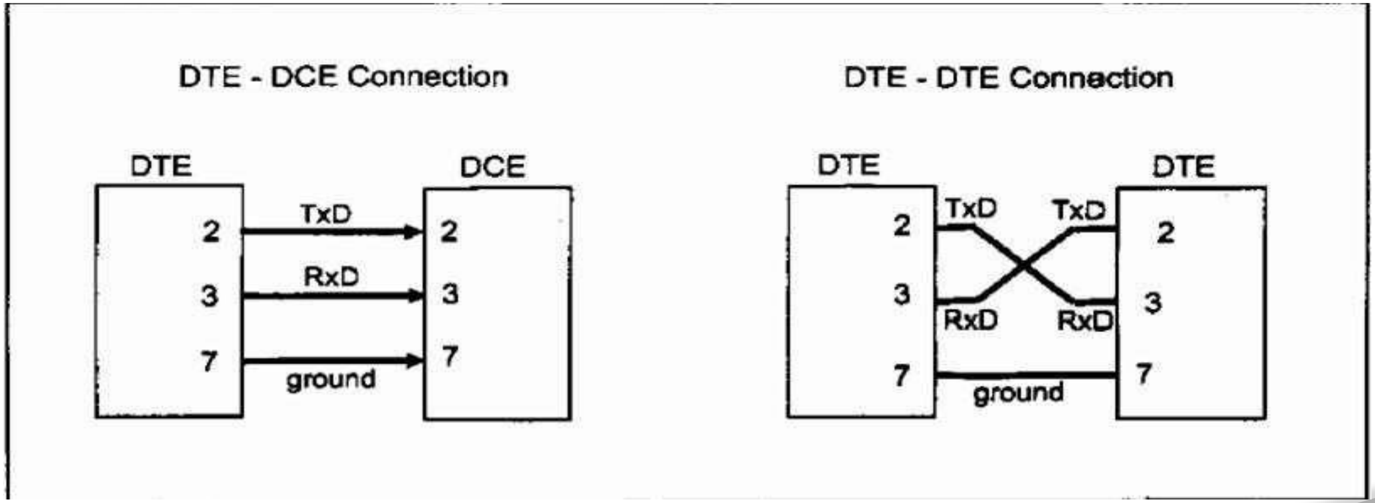
interfacing 8251 USART with 8086



TTL to RS 232C and RS232C to TTL conversion

RS-232 defines serial, asynchronouscommunication

- Serial - bits are encoded and transmitted one at a time (as opposed to parallel transmission)
- Asynchronous - characters can be sent at any time and bits are not individually synchronized

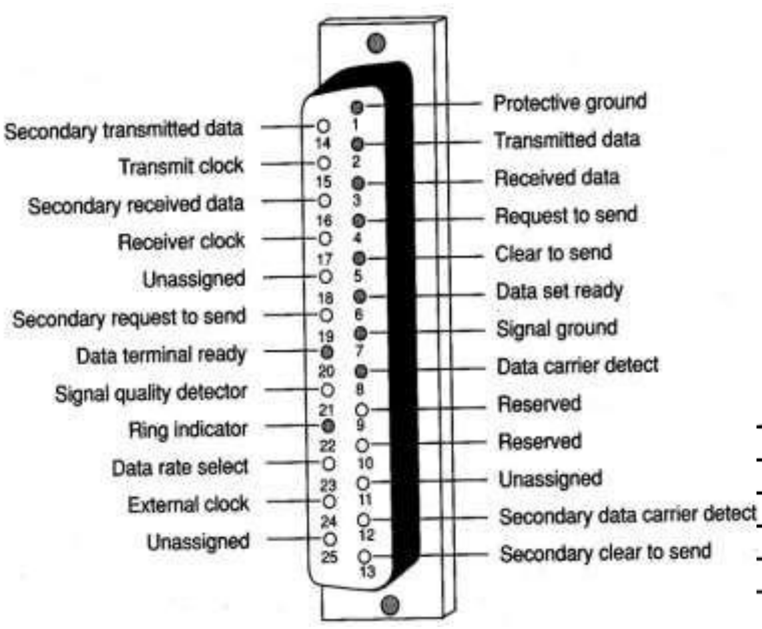


Electrical Characteristics

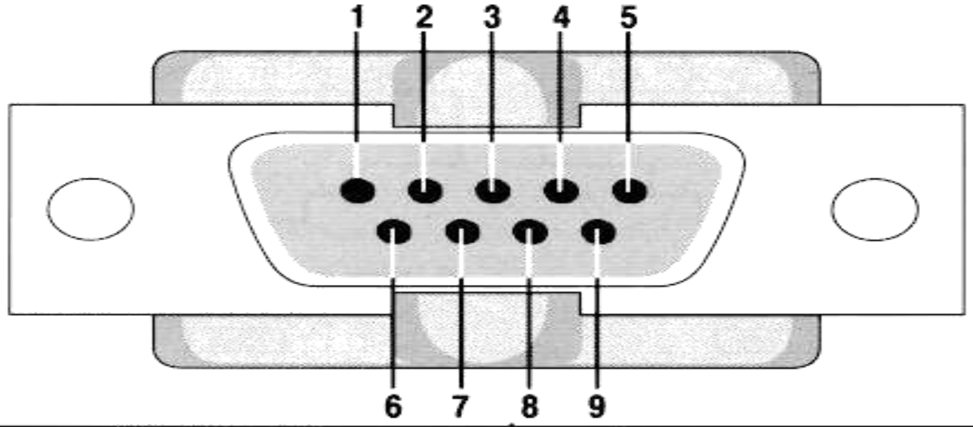
- Single-ended
  - One wire per signal, voltage levels are with respect to system common (i.e. signalground)
- Mark: –3V to –15V
  - represent Logic 1, Idle State (OFF)
- Space: +3 to +15V
  - represent Logic 0, Active State (ON)
- Usually swing between –12V to+12V
- Recommended maximum cable lengthis 15m, at 20kbps

Mechanical Characteristics

- 25-pin connector
- Use male connector on DTE and female connector on DCE.



25-Pin RS232 Connector



Pin	Signal	Pin	Signal
1	Data Carrier Detect	6	Data Set Ready
2	Received Data	7	Request to Send
3	Transmitted Data	8	Clear to Send
4	Data Terminal Ready	9	Ring Indicator
5	Signal Ground		

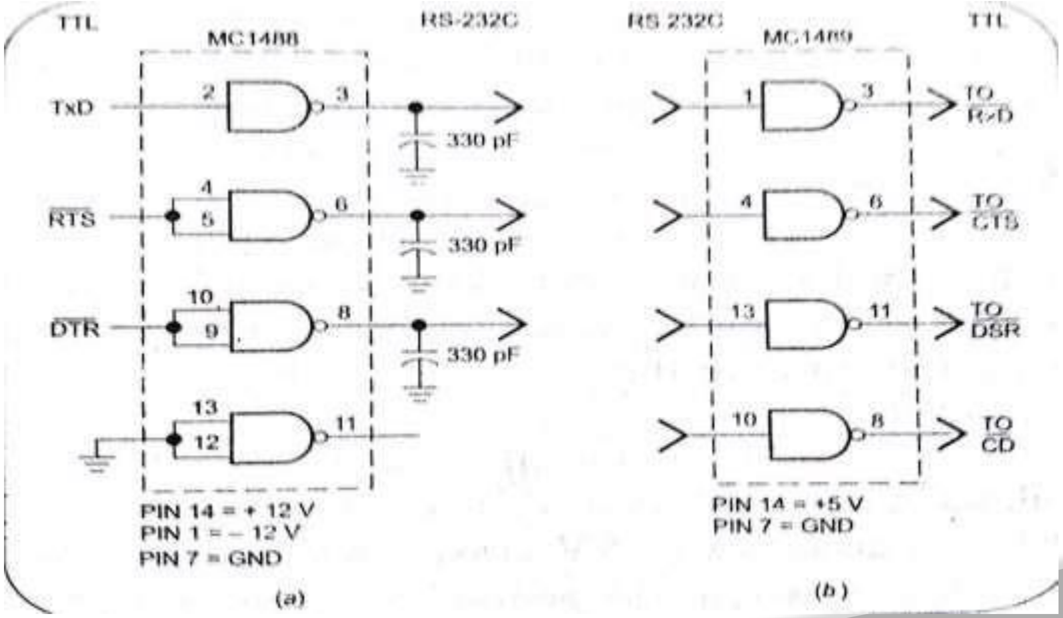
9-Pin RS232 Connector

Function of Signals

- TD: transmitted data
- RD: receiveddata
- DSR: data set ready
    - indicate whether DCE is poweredon.
- DTR: data terminalready
    - indicate whether DTR is powered on
    - turning off DTR causes modem to hang up the line
- RI: ring indicator
    - ON when modem detects phonecall.
- DCD: data carrier detect
    - ON when two modems have negotiated successfully and the carrier signal is established on thephoneline.

- RTS: request to send
  - ON when DTE wants to send data
  - Used to turn on and off modem's carrier signal in multi-point (i.e. multi-drop) lines
  - Normally constantly ON in point-to-point lines
- CTS: clear to send
  - ON when DCE is ready to receive data.
  - SG: signal ground

- Voltage levels, slew rate, and short-circuit behavior are typically controlled by a line driver(MC 1488) that converts from the USART's logic levels (TTL levels) to RS-232 compatible signal levels, and a receiver (MC 1489) that converts RS-232 compatible signal levels to the USART's logic levels (TTLlevels).



MODULE-IV
ADVANCED MICROPROCESSORS

COURSE OUTCOMES

CO8	Outline the salient features of 80286, 80386 and RISC processors in relation to basic 8086 microprocessor.
CO9	Illustrate the paging operation and segmentation of advanced microprocessors for memory management.

Introduction to 80286

Salient features of 80286

- High performance microprocessor with memory management and protection
- 80286 is the first member of the family of advanced microprocessors with built-in/on-chip memory management and protection abilities primarily designed for multi-user/multitasking systems
- Available in 8 MHz, 10 MHz & 12.5 MHz clock frequencies
- 80286 is upwardly compatible with 8086 in terms of instruction set.
- 80286 have two operating modes namely real address mode and virtual address mode.

Salient features of 80286:

- In real address mode, the 80286 can address up to 1Mb of physical memory address like 8086.
- In virtual address mode, it can address up to 16 Mb of physical memory address space and 1 GB of virtual memory address space.
- 80286 has some extra instructions to support operating system and memory management.
- In protected virtual address mode, it is source code compatible with 8086.
- The performance of 80286 is five times faster than the standard 8086.

Bus and memory sizes

- The 80286 CPU, with its 24-bit address bus is able toaddress 16MB of physical memory.
- 1GB of virtual memory for each task

Microprocessor	Data bus width	Address bus width	Memory size
8086	16	20	1M
80186	16	20	1M
80286	16	24	16M

Operating Modes:

Intel 80286 has 2 operating modes:

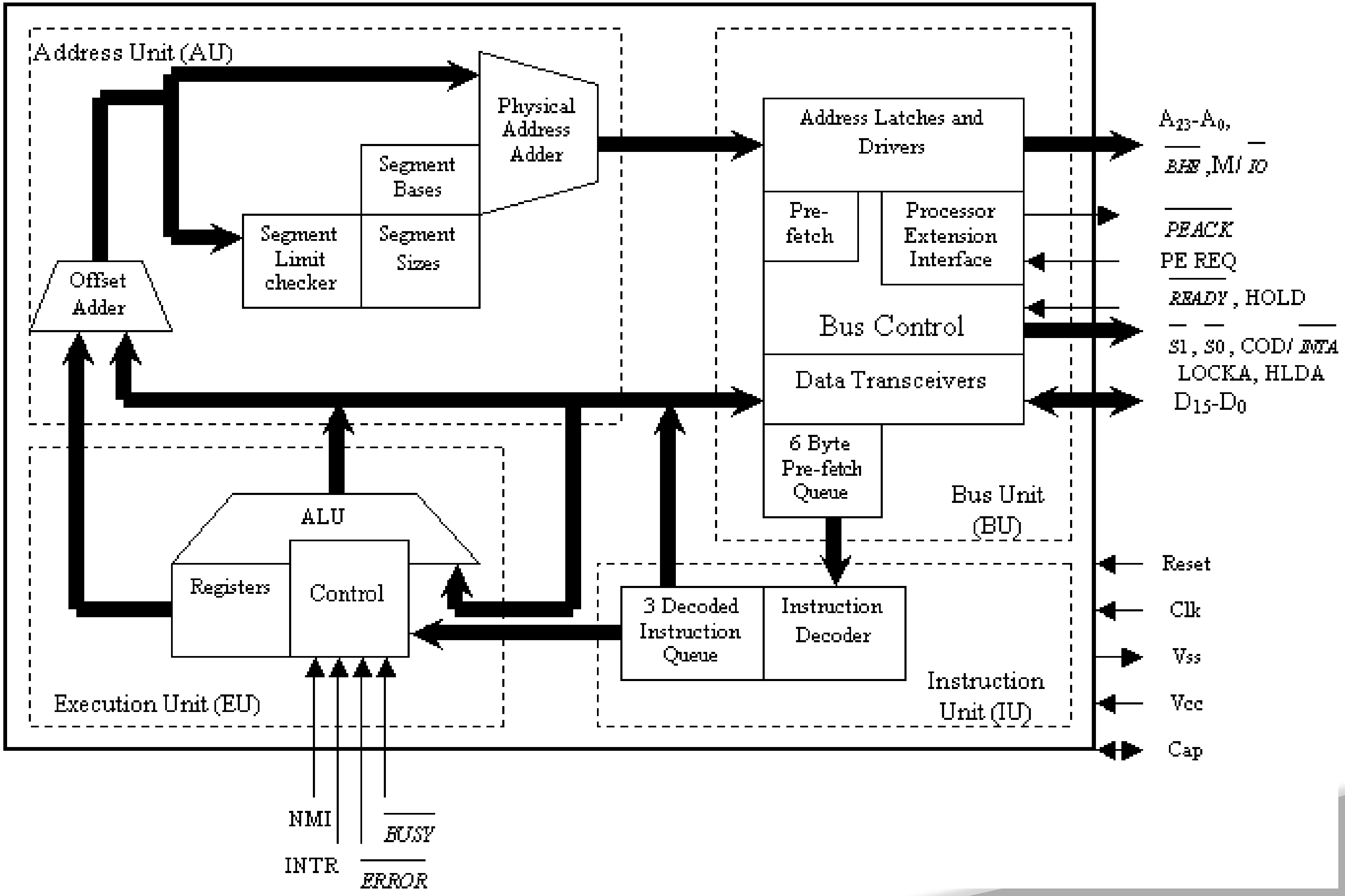
Real Address Mode :

- 80286 is just a fast 8086 --- up to 6 timesfaster
- All memory management and protection mechanisms are disabled
- 286 is object code compatible with 8086

Protected Virtual Address Mode

- 80286 works with all of its memory management and protection capabilities with the advanced instructionset.
- it is source code compatible with 8086

80286 Architecture:



Functional Parts:

1. Bus Interface unit
2. Instruction unit
3. Execution unit
4. Address unit

Bus Interface Unit

- Performs all memory and I/O read and write operations.
- Take care of communication between CPU and a coprocessor.
- Transmit the physical address over address bus $A_0 – A_{23.}$
- Prefetcher module in the bus unit performs this task of prefetching.
- Bus controller controls the prefetcher module.
- Fetched instructions are arranged in a 6 – byte prefetch queue.

Instruction Unit

- Receive arranged instructions from 6 byte prefetch queue.
- Instruction decoder decodes up to 3 prefetched instruction and are latched them onto a decoded instruction queue.
- Output of the decoding circuit drives a control circuit in the Execution unit.

Execution unit

- EU executes the instructions received from the decoded

  instruction queue sequentially.

- Contains Register Bank.

- contains one additional special register called Machine status

  word (MSW) register --- lower 4 bits are only used.

- ALU is the heart of execution unit.

- After execution ALU sends the result either over data bus or back

  to the register bank.

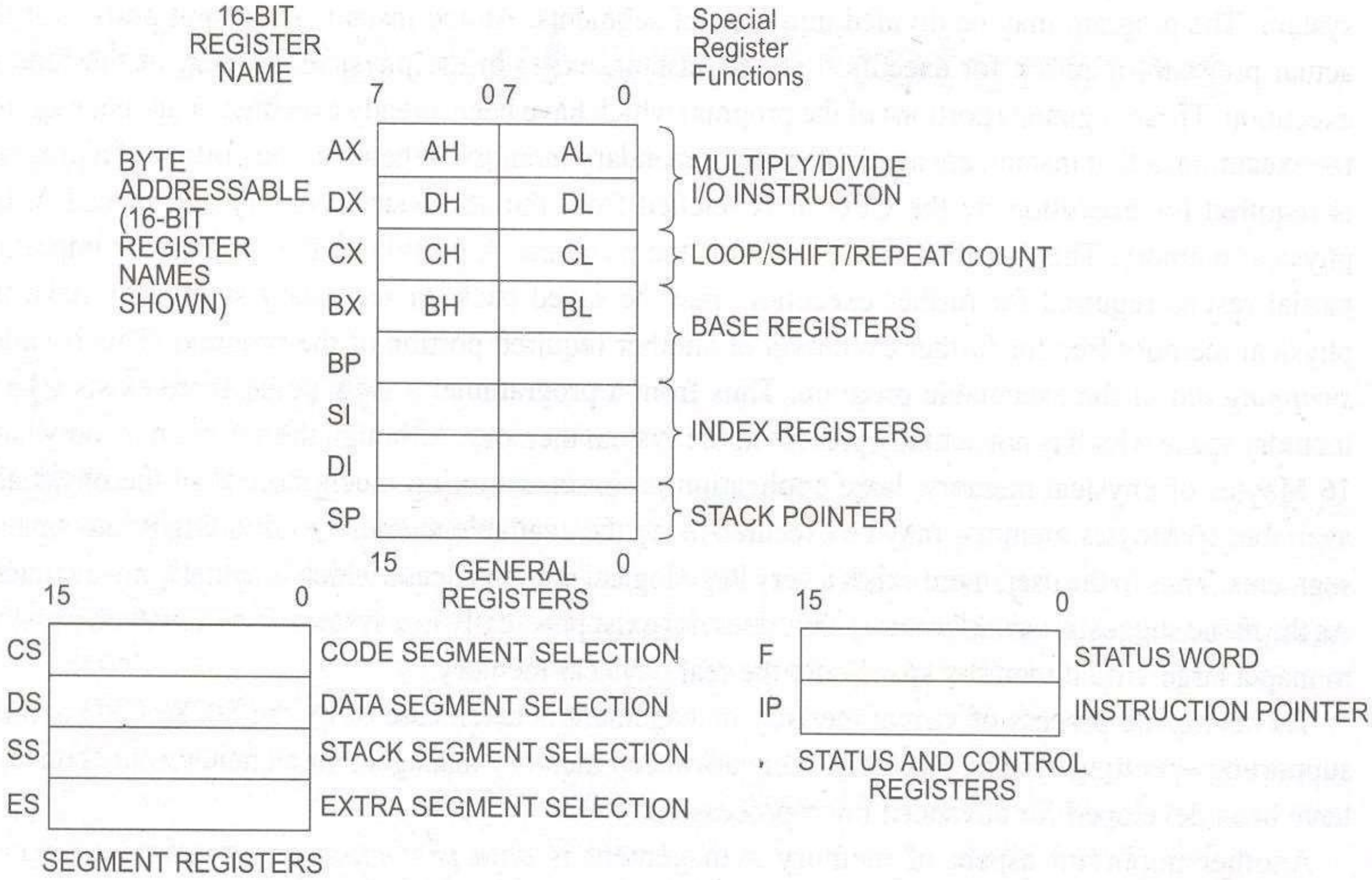
Address Unit

- Calculate the physical addresses of the instruction and data that the CPU want to access
- Address lines derived by this unit may be used to address different peripherals.
- Physical address computed by the address unit is handed over to the BUS unit.

REGISTER ORGANIZATION OF 80286:

The 80286 CPU contains almost the same set of registers, as in 8086, namely

- ➢ Eight 16-bit general purpose registers (AX, BX, CX, DX)
- ➢ Four 16-bit segment registers (CS, SS, DS, ES)
- ➢ Status and control registers (SP, BP, SI, DI)
- ➢ Instruction Pointer (IP)
- ➢ Two 16-bit register - FLAGS, MSW
- ➢ Two 16-bit register - LDTR and TR
- ➢ Two 48-bit register - GDTR and IDTR



16-BIT REGISTER NAME

Special Register Functions

BYTE ADDRESSABLE (16-BIT REGISTER NAMES SHOWN)

7 0 7 0

AX	AH	AL	MULTIPLY/DIVIDE I/O INSTRUCTON
DX	DH	DL	
CX	CH	CL	LOOP/SHIFT/REPEAT COUNT
BX	BH	BL	BASE REGISTERS
BP			
SI			INDEX REGISTERS
DI			
SP			STACK POINTER

15 GENERAL REGISTERS 0

15 0

CS		CODE SEGMENT SELECTION
DS		DATA SEGMENT SELECTION
SS		STACK SEGMENT SELECTION
ES		EXTRA SEGMENT SELECTION

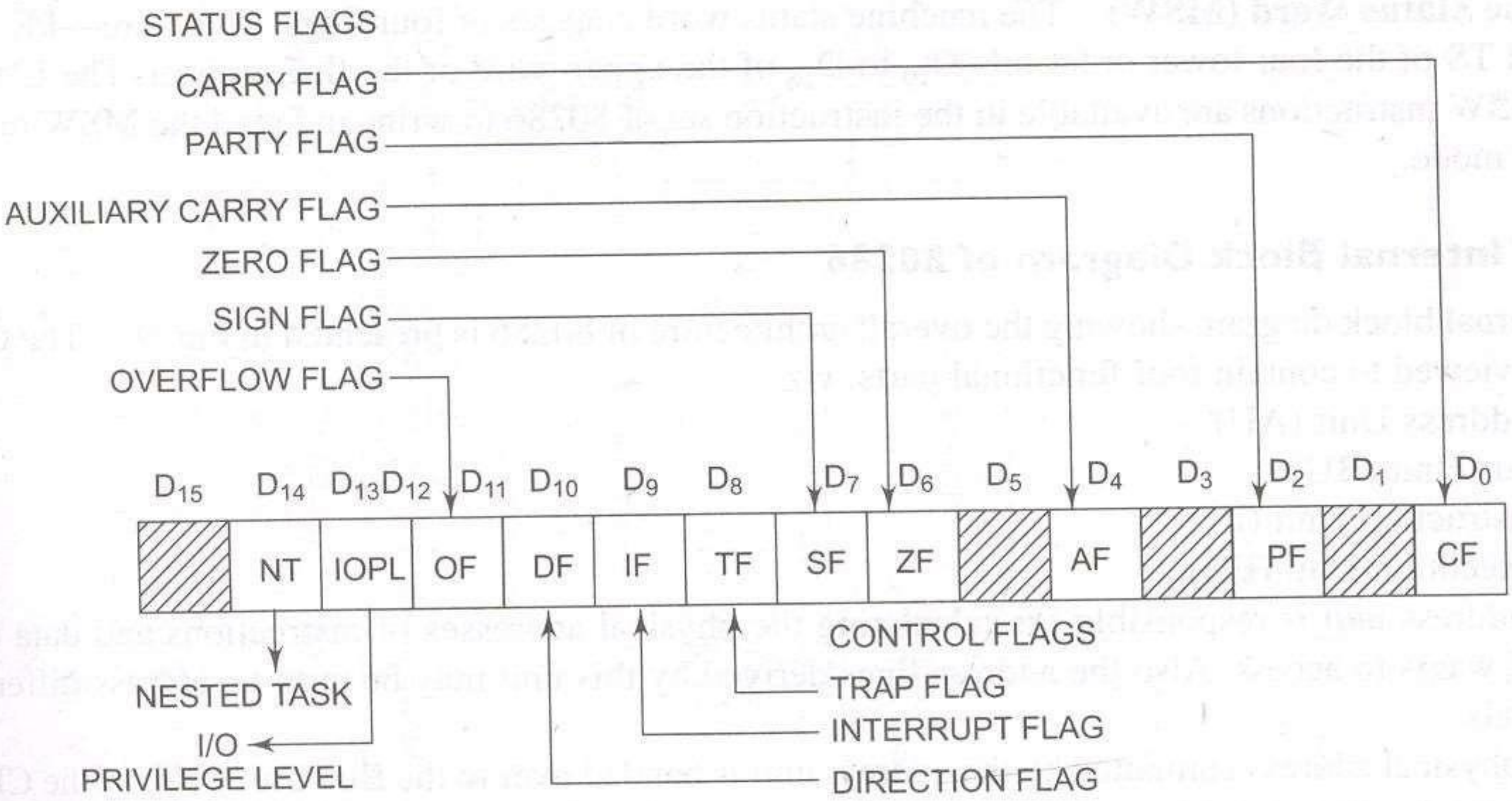
SEGMENT REGISTERS

15 0

F		STATUS WORD
IP		INSTRUCTION POINTER

STATUS AND CONTROL REGISTERS

Flag Register



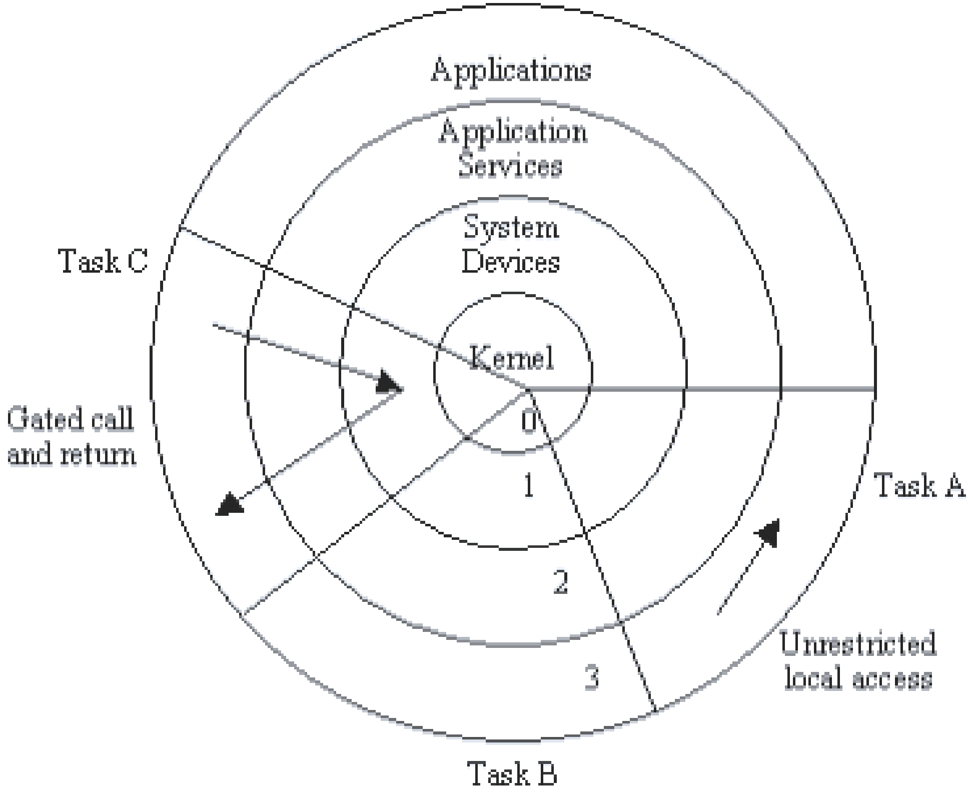
D15	D14	D13 D12	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0
▨	NT	IOPL	OF	DF	IF	TF	SF	ZF	▨	AF	▨	PF	▨	CF

- The initial protected mode, released with the 286, was not widely used;
- for example, it was used by Microsoft xenix (around 1984),coherent and minix. Several shortcomings such as the inability to access the BIOS or DOS calls due to inability to switch back to real mode without resetting the processor prevented widespread usage.
- Acceptance was additionally hampered by the fact that the 286 only allowed memory access in 16 bit segments via each of four segment registers, meaning only 4*2 bytes, equivalent to 256 kilobytes, could be accessed at a time Because changing a segment register in protected mode caused a 6-byte segment descriptor to be loaded into the CPU from memory

- The segment register load instruction took many tens of processor cycles, making it much slower than on the 8086; therefore, the strategy of computing segment addresses on-the-fly in order to access data structures larger than
  128 kilobytes (the combined size of the two data segments) became impractical, even for those few programmers who had mastered it on the 8086/8088

There are four types of privilege levels

- 00 - kernel level (highest privilege level)
- 01 - OS services
- 10 - OS extensions
- 11 - Applications (lowest privilege level)
- Each task assigned a privilege level, which indicates thepriority or privilege of that task.
- It can only changed by transferring the control, using gate descriptors, to a new segment.
- A task executing at level 0, the most privileged level, can access all the data segment defined in GDT and LDT of the task.
- A task executing at level 3, the least privileged level, will havethe most limited access to data and other descriptors.


Applications
Application Services
System Devices
Kernel
Task C
Gated call and return
0
1
2
3
Task A
Unrestricted local access
Task B

Base Address

- 32 bit starting memory address of the segment Segment Limit

- 20 bit length of the segment. (More specifically, the address of the last accessible data, so the length is one more that the value stored here.) How exactly this should be interpreted depends on other bits of the segment descriptor.

  G=Granularity

- If clear, the limit is in units of bytes, with a maximum of 2^{20} bytes. If set, the limit is in units of 4096-byte pages, for a maximum of 2^{32} bytes.

Base Address

- D=Default operand size

If clear, this is a 16-bit code segment; if set, this is a 32-bit segment

- L=Long-mode segment

If set, this is a 64-bit segment (and D must be zero), and code in this segment uses the 64-bit instruction encoding

- AVL=Available

For software use, not used by hardware

- D=Default operand size

If clear, this is a 16-bit code segment; if set, this is a 32-bit segment

- L=Long-mode segment

If set, this is a 64-bit segment (and D must be zero), and code in this segment uses the 64-bit instruction encoding

- AVL=Available

For software use, not used by hardware

P=Present

- If clear, a "segment not present" exception is generated on any reference to this segment

DPL=Descriptor privilege level

- Privilege level required to access this descriptor

C=Conforming

- Code in this segment may be called from less-privileged levels

R=Readable

- If clear, the segment may be executed but not read from

A=Accessed

- This bit is set to 1 by hardware when the segment is accessed, and cleared by software

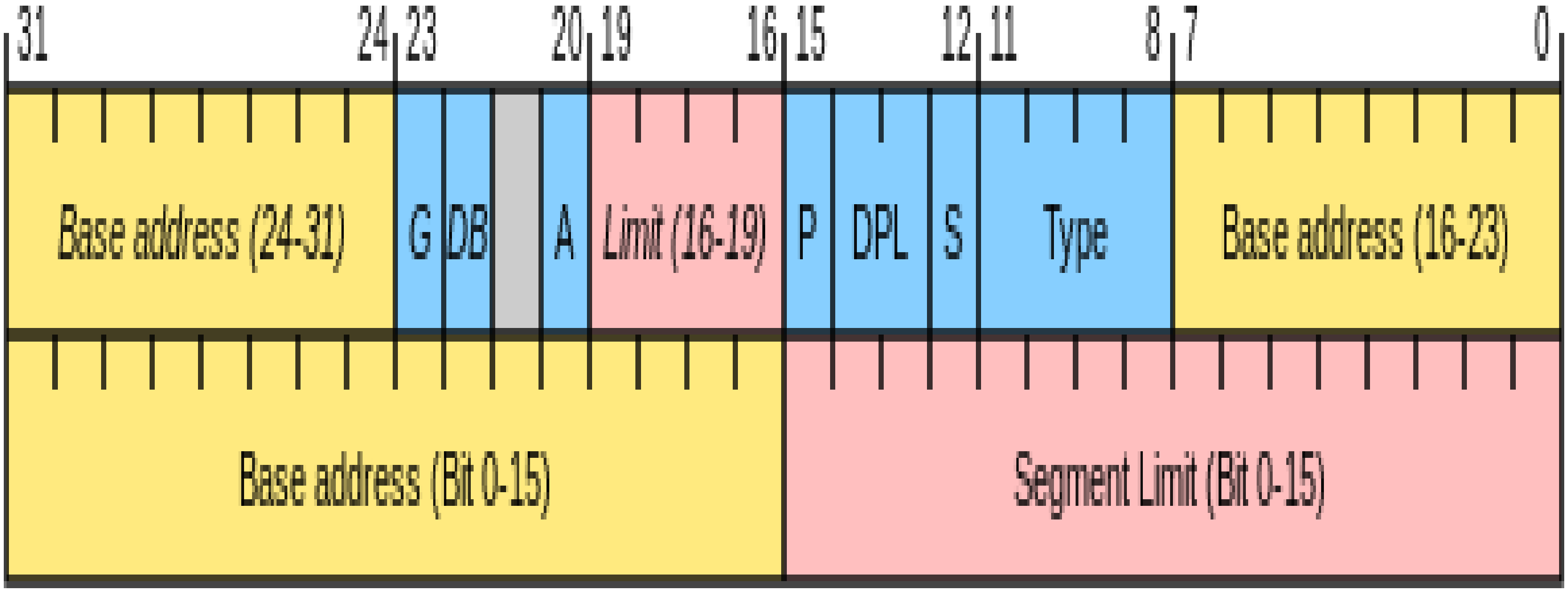
Memory access in GDT and LDT

- The Global Descriptor Table or GDT is a data structure used by Intel x86- family processors starting with the 80286 in order to define the characteristics of the various memory areas used during program execution, including the base address, the size and access privileges like execute- ability and write-ability.

Memory access in GDT and LDT

- There is also a Local Descriptor Table (LDT). While the LDT contains memory segments which are private to a specific program, the GDT contains global segments.
- The x86 processors have facilities for automatically switching the current LDT on specific machine events, but no facilities for automatically switching the GDT.

Memory access in GDT and LDT

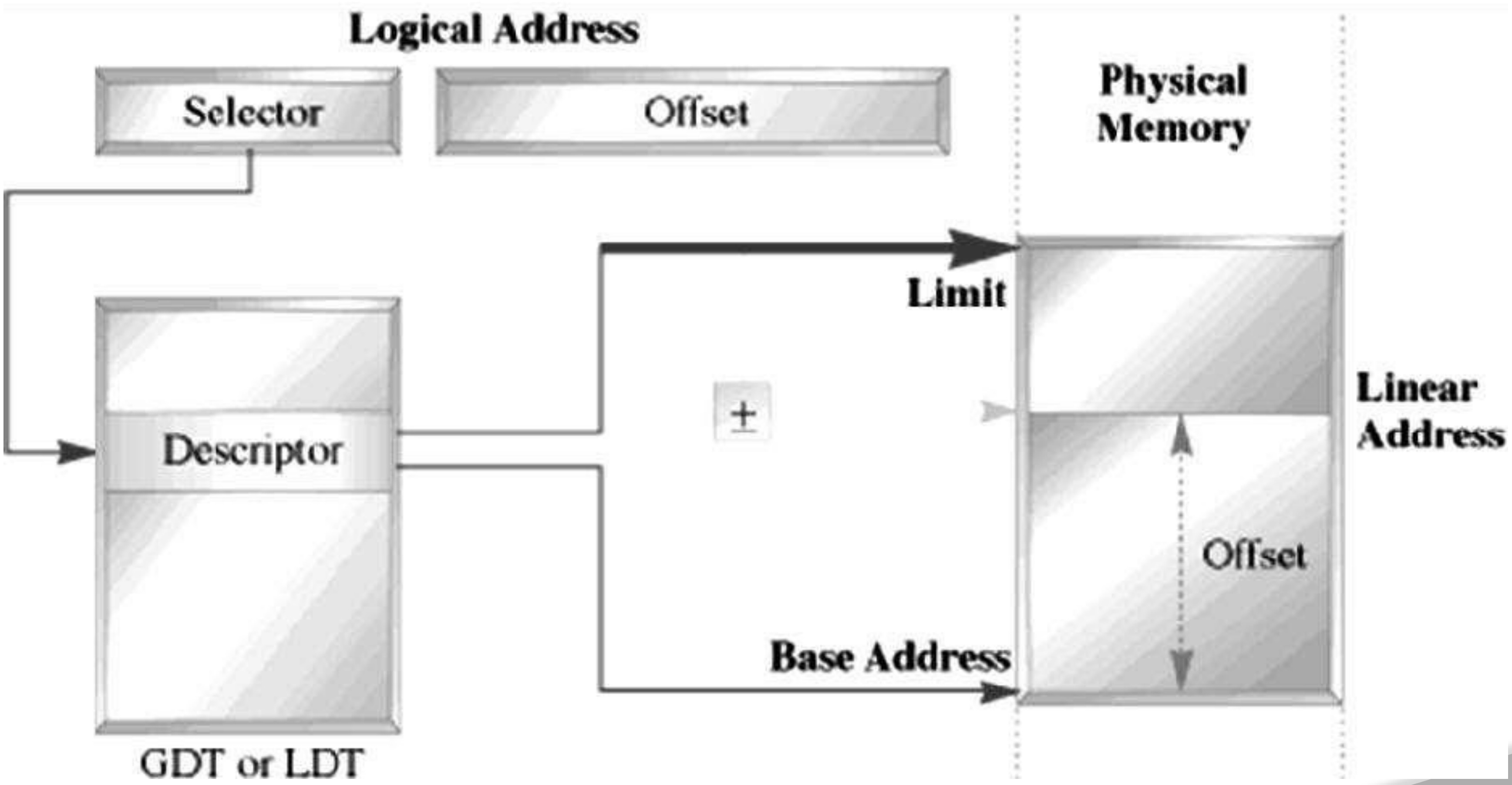


31 24 23 20 19 16 15 12 11 8 7 0

Base address (24-31) G DB A Limit (16-19) P DPL S Type Base address (16-23)

Base address (Bit 0-15) Segment Limit (Bit 0-15)

Memory access in GDT and LDT



GDT or LDT

Memory access in GDT and LDT

Memory Accessing In GDT or LDT

- A segment cannot be accessed, if its descriptor does not exist in either LDT or GDT.
- Set of descriptor (descriptor table) arranged in a proper sequence describes the complete program.

Memory access in GDT and LDT

- The descriptor is a block of contiguous memory location containing information of a segment, like
  - Segment base address
  - Segment limit
  - Segment type
  - Privilege level – prevents unauthorized access
  - Segment availability in physical memory
  - Descriptor type
  - Segment use by another task

Memory access in GDT and LDT

- The Global Descriptor Table or GDT is a data structure used by Intel x86-family processors starting with the 80286 in order to define the characteristics of the various memory areas used during program execution, including the base address, the size and access privileges like execute- ability and write-ability.

Memory access in GDT and LDT

- Local Descriptor Table (LDT). While the LDT contains memory segments which are private to a specific program, the GDT contains global segments. The x86 processors have facilities for automatically switching the current LDT on specific machine events, but no facilities for automatically switching the GDT.

Memory access in GDT and LDT

Differentiate between GDT and LDT.

- LDT is actually defined by a descriptor inside the GDT, while the GDT is directly defined by a linear address.The lack of symmetry between both tables is underlined by the fact that the current LDT can be automatically switched on certain events, notably if TSS-based multitasking is used, while this is not possible for the GDT.
- The LDT also cannot store certain privileged types of memory segments.

Memory access in GDT and LDT

- The LDT is the sibling of the Global Descriptor Table (GDT) and similarly defines up to 8191 memory segments accessible to programs.
- LDT (and GDT) entries which point to identical memory areas are called *aliases*.
- Instruction to load GDT is LGDT(Load Global Descriptor Table) and instruction to load LDT is LLDT(Load Global Descriptor Table). Both are privileged instructions.

Multitasking

Multitasking

- multitasking is the concurrent execution of multiple tasks (also known as processes) over a certain period of time. New tasks can interrupt already started ones before they finish, instead of waiting for them to end.
- As a result, a computer executes segments of multiple tasks in an interleaved manner, while the tasks share common processing resources such as central processing unit (CPUs) and main memory.

Multitasking

context switch

- Multitasking automatically interrupts the running program, saving its state (partial results, memory contents and computer register contents) and loading the saved state of another program and transferring control to it.
- This “context switch" may be initiated at fixed time intervals (pre-emptive multitasking), or the running program may be coded to signal to the supervisory software when it can be interrupted (cooperative multitasking).

Multitasking

Features of Multitasking

- It allows more efficient use of the computer hardware; where a program is waiting for some external event such as a user input or an input/output transfer with a peripheral to complete, the central processor can still be used with another program.
- In a time sharing system, multiple human operators use the same processor as if it was dedicated to their use, while behind the scenes the computer is serving many users by multitasking their individual programs.

Multitasking

- In multiprogramming systems, a task runs until it must wait for an external event or until the operating system's scheduler forcibly swaps the running task out of the CPU.

Multitasking

Applications :

- Real-time systems such as those designed to control industrial robots, require timely processing;
- single processor might be shared between calculations of machine movement, communications, and user interface.

Multitasking

Advantages

- Often multitasking operating systems include measures to change the priority of individual tasks, so that important jobs receive more processor time than those considered less significant.
- Depending on the operating system, a task might be as large as an entire application program, or might be made up of smaller threads that carry out portions of the overall program.

Addressing Modes

Multitasking

- multitasking is the concurrent execution of multiple tasks (also known as processes) over a certain period of time. New tasks can interrupt already started ones before they finish, instead of waiting for them to end.
- As a result, a computer executes segments of multiple tasks in an interleaved manner, while the tasks share common processing resources such as central processing unit (CPUs) and main memory.

Addressing Modes

context switch

- Multitasking automatically interrupts the running program, saving its state (partial results, memory contents and computer register contents) and loading the saved state of another program and transferring control to it.
- This “context switch" may be initiated at fixed time intervals (pre-emptive multitasking), or the running program may be coded to signal to the supervisory software when it can be interrupted (cooperative multitasking).

Features of Multitasking

- It allows more efficient use of the computer hardware; where a program is waiting for some external event such as a user input or an input/output transfer with a peripheral to complete, the central processor can still be used with another program.
- In a time sharing system, multiple human operators use the same processor as if it was dedicated to their use, while behind the scenes the computer is serving many users by multitasking their individual programs.

- In multiprogramming systems, a task runs until it must wait for an external event or until the operating system's scheduler forcibly swaps the running task out of the CPU.

Addressing Modes

Applications :

- Real-time systems such as those designed to control industrial robots, require timely processing;
- A single processor might be shared between calculations of machine movement, communications, and user interface.

Advantages

- Often multitasking operating systems include measures to change the priority of individual tasks, so that important jobs receive more processor time than those considered less significant.
- Depending on the operating system, a task might be as large as an entire application program, or might be made up of smaller threads that carry out portions of the overall program.

Addressing Modes

Direct addressing mode:

- In the direct addressing mode, a 16-bit memory address (offset)

  directly specified in the instruction as a part of it.

  Example: MOV AX, [5000H].

Register addressing mode:

- In the register addressing mode, the data is stored in a register and it is referred using the particular register. All the registers, except IP, may be used in this mode.

  Example: MOV BX, AX

Addressing Modes

Register indirect addressing mode:

- Sometimes, the address of the memory location which contains data or operands is determined in an indirect way, using the offset registers. The mode of addressing is known as register indirect mode.
- In this addressing mode, the offset address of data is in either BX or SI

  or DI Register. The default segment is either DS or ES.

Example: MOV AX, [BX].

Addressing Modes

Indexed addressing mode:

- In this addressing mode, offset of the operand is stored one of the index registers. DS & ES are the default segments for index registers SI & DI respectively.

  Example: MOV AX, [SI]

- Here, data is available at an offset address stored in SI in DS.

Register relative addressing mode:

- In this addressing mode, the data is available at an effective address formed by adding an 8-bit or 16-bit displacement with the content of any one of the register BX, BP, SI & DI in the default (either in DS & ES) segment.

  Example: MOV AX, 50H [BX]

Addressing Modes

Based indexed addressing mode:

- The effective address of data is formed in this addressing mode, by adding content of a base register (any one of BX or BP) to the content of an index register (any one of SI or DI). The default segment register may be ES or DS.

  Example: MOV AX, [BX][SI]

Relative based indexed:

- The effective address is formed by adding an 8 or 16-bit displacement with the sum of contents of any of the base registers (BX or BP) and any one of the index registers, in a default segment.

  Example: MOV AX, 50H [BX] [SI]

Addressing Modes

Addressing Modes for control transfer instructions:

- Intersegment
  - Intersegment direct
  - Intersegment indirect

- Intrasegment
  - Intrasegment direct
  - Intrasegment indirect

Addressing Modes

Intersegment direct:

- In this mode, the address to which the control is to be transferred is in a different segment. This addressing mode provides a means of branching from one code segment to another code segment. Here, the CS and IP of the destination address are specified directly in the instruction.

Example: JMP 5000H: 2000H;

- Jump to effective address 2000H in segment 5000H.

Addressing Modes

Intersegment indirect:

- In this mode, the address to which the control is to be transferred lies in a different segment and it is passed to the instruction indirectly, i.e. contents of a memory block containing four bytes,
  i.e. IP(LSB), IP(MSB), CS(LSB) and CS(MSB) sequentially. The starting address of the memory block may be referred using any of the addressing modes, except immediate mode.

Example: JMP [2000H].

Jump to an address in the other segment specified at effective
address 2000H in DS.

Addressing Modes

Intrasegment direct mode:

- In this mode, the address to which the control is to be transferred lies in the same segment in which the control transfers instruction lies and appears directly in the instruction as an immediate displacement value.
- In this addressing mode, the displacement is computed relative to the content of the instruction pointer.

Addressing Modes

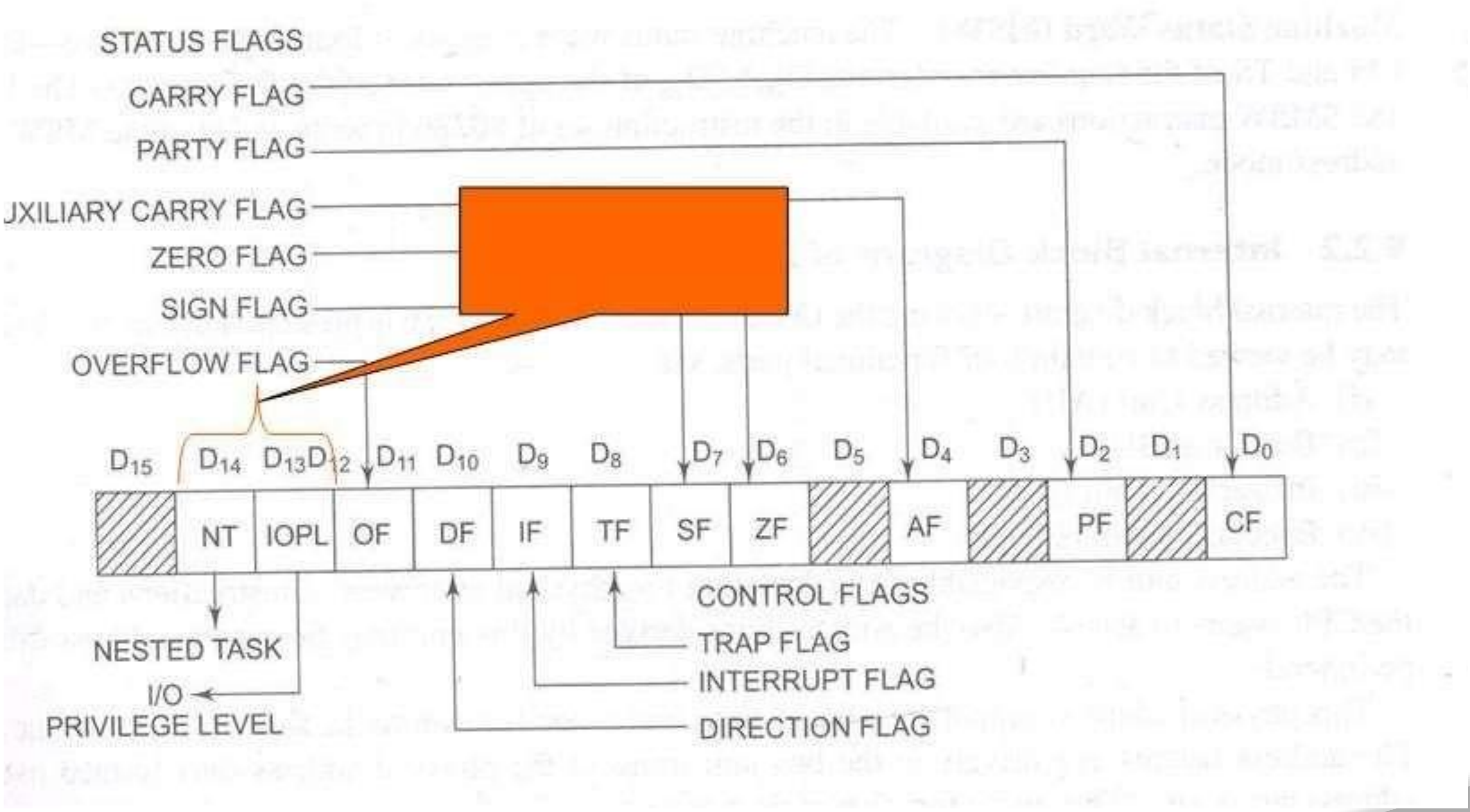
Intrasegment indirect mode:

- In this mode, the displacement to which the control is to be transferred is in the same segment in which the control transfer instruction lies, but it is passed to the instruction directly. Here, the branch address is found as the content of a register or a memory location.

- This addressing mode may be used in unconditional branch instructions.

- Example: JMP [BX]; Jump to effective address stored in BX.

Flag Register of 80286



Flag Register of 80286

IOPL – Input Output Privilege Level flags (bit D12 and D13

- 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 executed without hindrance.
- If the IOPL is lover than the current privilege level, an interrupt occurs, causing execution to suspend.Note that IPOL 00 is the highest or more trusted; and IOPL 11 is the lowest or least

Flag Register of 80286

- NT – Nested task flag (bit D14)
- When set, it indicates that one system task has invoked another through a CALL instruction as opposed to a JMP.
- For multitasking this can be manipulated to our advantage

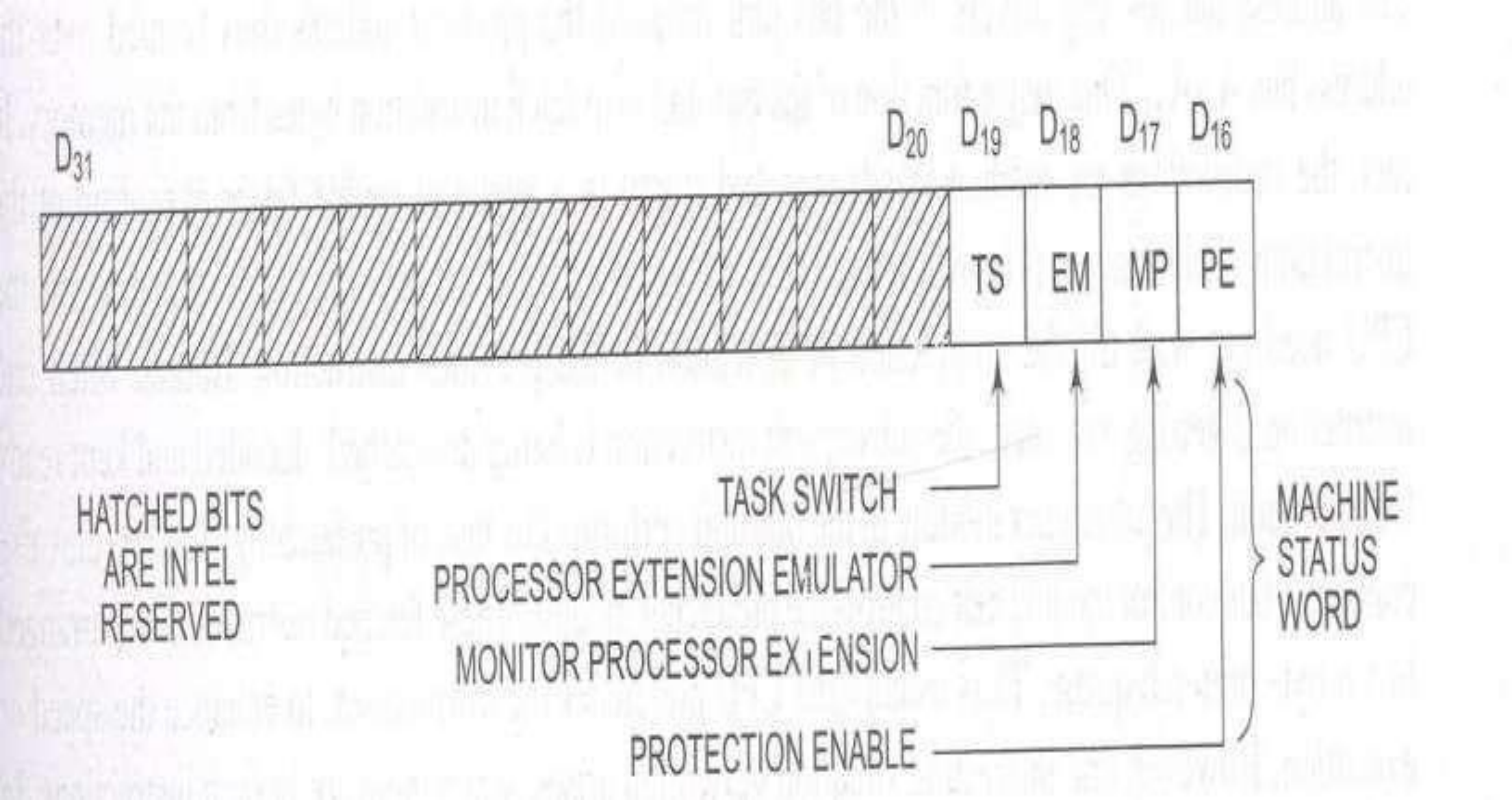
Flag Register of 80286

Machine Status Word Register

- Consist of four flags
  - PE,
  - MP,
  - EM and
  - TS are for the most part used toindicate whether a processor extension (co-processor) is present in the system or not

Flag Register of 80286

- **Word Machine Status...**


D31
D20
D19
D18
D17
D16
TS
EM
MP
PE
HATCHED BITS ARE INTEL RESERVED
TASK SWITCH
PROCESSOR EXTENSION EMULATOR
MONITOR PROCESSOR EXTENSION
PROTECTION ENABLE
MACHINE STATUS WORD

Flag Register of 80286

- PE - Protection enable

  Protection enable flag places the 80286 in protected mode, if set. this can only be cleared by resetting the CPU.

- MP – Monitor processor extension

  flag allows WAIT instruction to generate a processor extension.

- Emulate processor extension flag,

  if set , causes a processor extension absent exception and permits the emulation of processor extension by CPU.

Salient Features of 80386

Features of 80386

- The 80386 microprocessor is an enhanced version of the 80286 microprocessor and includes a memory-management unit is enhanced to provide memory paging
- The 80386 also includes 32-bit extended registers and a 32-bit address and data bus
- The 80386 has a physical memory size of 4GBytes that can be addressed as a virtual memory with up to 64TBytes
- The 80386 is operated in the pipelined mode, it sends the address of the next instruction or memory data to the memory system prior to completing the execution of the current instruction

Features of 80386

- This allows the memory system to begin fetching the next instruction or data before the current is completed
- This increases access time, thus reducing the speed of the memory
- The I/O structure of the 80386 is almost identical to the 80286, except that I/O can be inhibited when the 80386 is operated in the protected mode through the I/O bit protection map
- The register set of the 80386 contains extended versions of the registers introduced on the 80286 microprocessor. These extended registers include EAX, EBX, ECX, EDX, EBP, ESP, EDI, ESI, EIP and EFLAGS
- The instruction set of the 80386 is enhanced to include instructions that address the 32-bit extended register set

Features of 80386

- Interrupts, in the 80386 microprocessor, have been expanded to include additional predefined interrupts in the interrupt vector table
- The 80386 memory manager is similar to the 80286, except the physical addresses generated by the MMU are 32-bits wide instead of 24-bits
- The 80386 is also capable of paging
- The 80386 is operated in the real mode (i.e. 8086 mode) when it is reset

Features of 80386

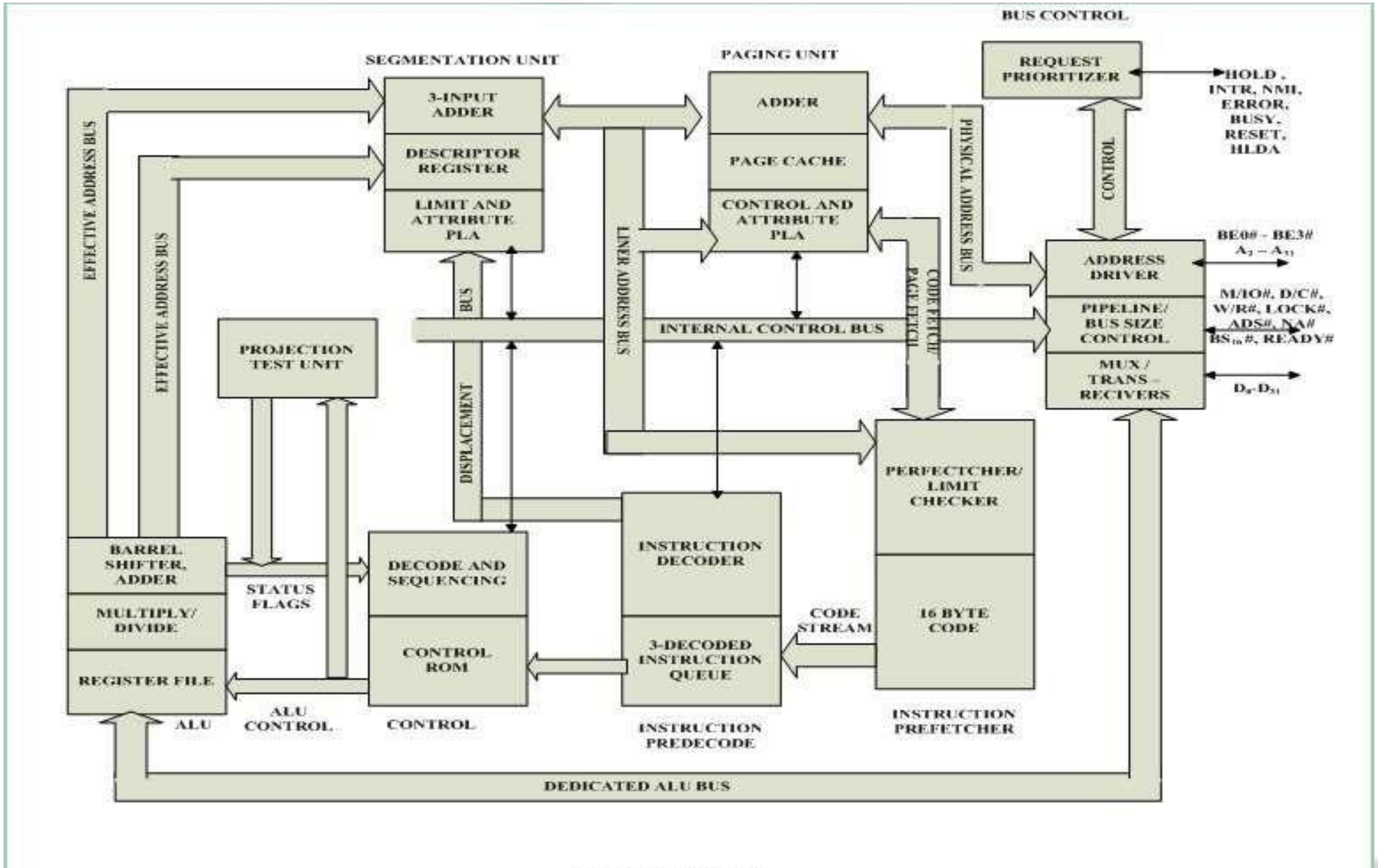
- The real mode allows the microprocessor to address data in the first 1MByte of memory
- In the protected mode, 80386 addresses any location in its 4G bytes of physical address space

Architecture of 80386

The Internal Architecture of 80386 is divided into 3sections.

- Central processing unit
- Memory management unit
- Bus interface unit
- Central processing unit is further divided into Execution unit and Instruction unit
- Execution unit has 8 General purpose and 8 Special purpose registers which are either used for handling data or calculating offset addresses.

Architecture of 80386



80386 ARCHITECTURE

- The Instruction unit decodes the opcode bytes received from the 16- byte instruction code queue and arranges them in a 3- instruction decoded instruction queue.

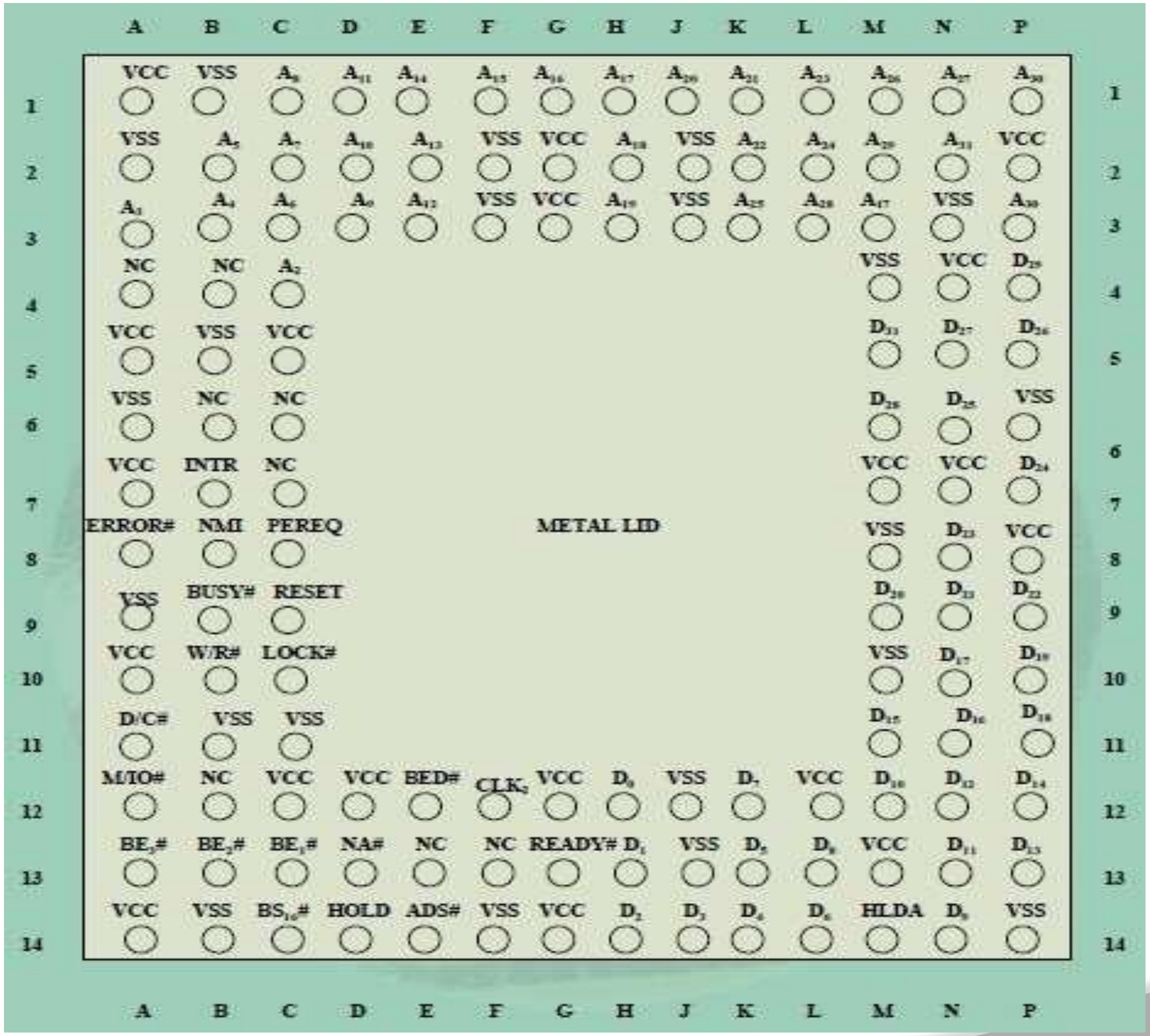
- After decoding them pass it to the control section for deriving the necessary control signals. The barrel shifter increases the speed of all shift and rotate operations.

- The multiply / divide logic implements the bit-shift-rotate algorithms to complete the operations in minimum time.

- Even 32- bit multiplications can be executed within one microsecond by the multiply / divide logic.

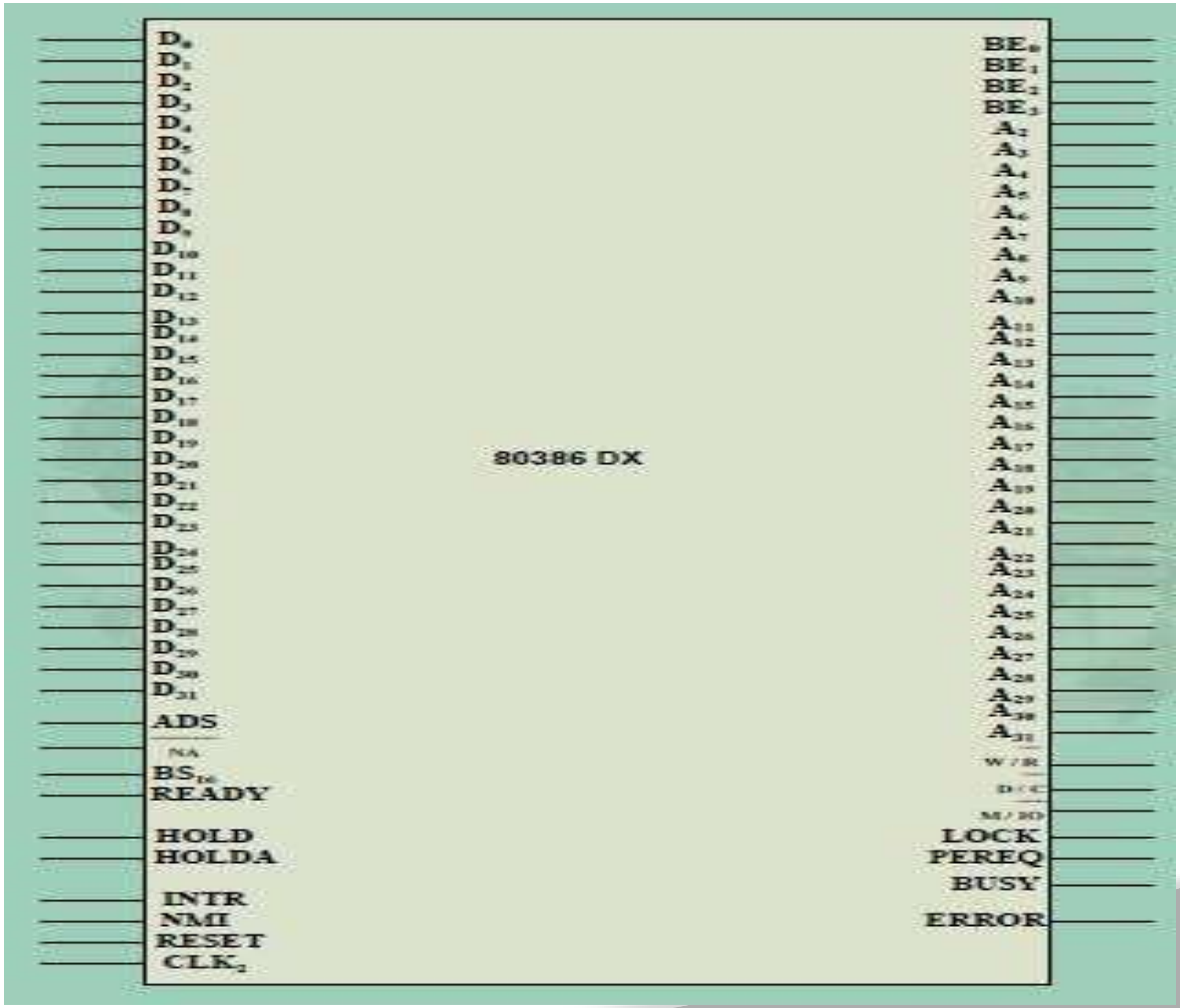
- The Memory management unit consists of a Segmentation unit and a Paging unit.

Pin diagram of 80386



	A	B	C	D	E	F	G	H	J	K	L	M	N	P	
1	VCC	VSS	A_8	A_{11}	A_{14}	A_{15}	A_{16}	A_{17}	A_{20}	A_{21}	A_{23}	A_{26}	A_{27}	A_{30}	1
2	VSS	A_5	A_7	A_{10}	A_{13}	VSS	VCC	A_{18}	VSS	A_{22}	A_{24}	A_{29}	A_{31}	VCC	2
3	A_3	A_4	A_6	A_9	A_{12}	VSS	VCC	A_{19}	VSS	A_{25}	A_{28}	A_{27}	VSS	A_{30}	3
4	NC	NC	A_2									VSS	VCC	D_{29}	4
5	VCC	VSS	VCC									D_{31}	D_{27}	D_{26}	5
6	VSS	NC	NC									D_{28}	D_{25}	VSS	6
7	VCC	INTR	NC									VCC	VCC	D_{24}	7
8	ERROR#	NMI	PEREQ				METAL LID					VSS	D_{23}	VCC	8
9	VSS	BUSY#	RESET									D_{20}	D_{21}	D_{22}	9
10	VCC	W/R#	LOCK#									VSS	D_{17}	D_{19}	10
11	D/C#	VSS	VSS									D_{15}	D_{16}	D_{18}	11
12	M/IO#	NC	VCC	VCC	BED#	CLK_2	VCC	D_0	VSS	D_7	VCC	D_{10}	D_{12}	D_{14}	12
13	BE_3#	BE_2#	BE_1#	NA#	NC	NC	READY#	D_1	VSS	D_5	D_8	VCC	D_{11}	D_{13}	13
14	VCC	VSS	BS_{16}#	HOLD	ADS#	VSS	VCC	D_2	D_3	D_4	D_6	HLDA	D_9	VSS	14
	A	B	C	D	E	F	G	H	J	K	L	M	N	P	

Pin diagram of 80386



Signal Descriptions of 80386

- CLK2 :The input pin provides the basic system clock timing for the operation of 80386.

- D0 – D31:These 32 lines act as bidirectional data bus during different access cycles.

- A31 – A2: These are upper 30 bit of the 32- bit address bus.

- BE0 toBE3 : The 32- bit data bus supported by 80386 and the memory system of 80386 can be viewed as a 4- byte wide memory access mechanism.

- ADS: The address status output pin indicates that the address bus and bus cycle definition pins(W/R#, D/C#, M/IO#, BE0# to BE3#) are carrying the respective validsignals.

Signal Descriptions of 80386

- VCC: These are system power supply lines.
- VSS: These return lines for the power supply.
- BS16: The bus size – 16 input pin allows the interfacing of 16 bit devices with the 32 bit wide 80386 databus.
- HOLD: The bus hold input pin enables the other bus masters to gain control of the system bus if it is asserted.
- HLDA: The bus hold acknowledge output indicates that a valid bus hold request has been received and the bus has been relinquished by the CPU.

Signal Descriptions of 80386

- ERROR: The error input pin indicates to the CPU that the coprocessor has encountered an error while executing its instruction.

- PEREQ: The processor extension request output signal indicates to the CPU to fetch a data word for the coprocessor.

- INTR: This interrupt pin is a maskable interrupt, that can be masked using the IF of the flag register.

- NMI: A valid request signal at the non-maskable interrupt request input pin internally generates a non- maskable interrupt of type2.

Signal Descriptions of 80386

- READY: The ready signals indicates to the CPU that the previous bus cycle has been terminated and the bus is ready for the next cycle.

- BUSY: The busy input signal indicates to the CPU that the coprocessor is busy with the allocated task.

- RESET: A high at this input pin suspends the current operation

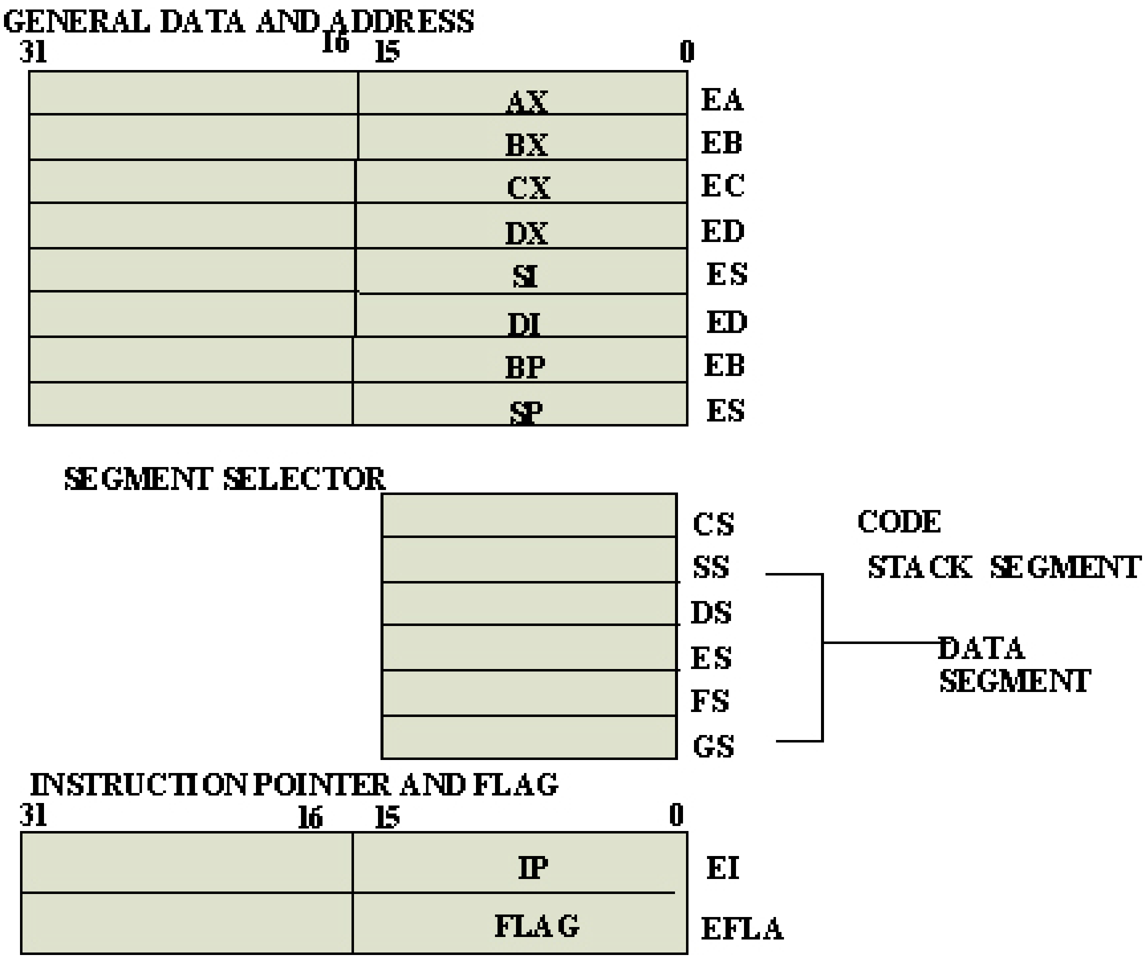
  and restart the execution from the starting location.

- N / C : No connection pins are expected to be left open.

80386 Register Organization

- The 80386 has eight 32 - bit general purpose registers which may be used as either 8 bit or 16 bit registers.
- A 32 - bit register known as an extended register, is represented by the register name with prefix E.
- The six segment registers available in 80386 are CS, SS, DS, ES, FS and GS.
- The CS and SS are the code and the stack segment registers respectively, while DS, ES, FS, GS are 4 data segment registers.
- A 16 bit instruction pointer IP is available along with 32 bit counterpart EIP.

80386 Register Organization



GENERAL DATA AND ADDRESS

31 16 15 0

	AX	EA
	BX	EB
	CX	EC
	DX	ED
	SI	ES
	DI	ED
	BP	EB
	SP	ES

SEGMENT SELECTOR

	CS
	SS
	DS
	ES
	FS
	GS

CODE

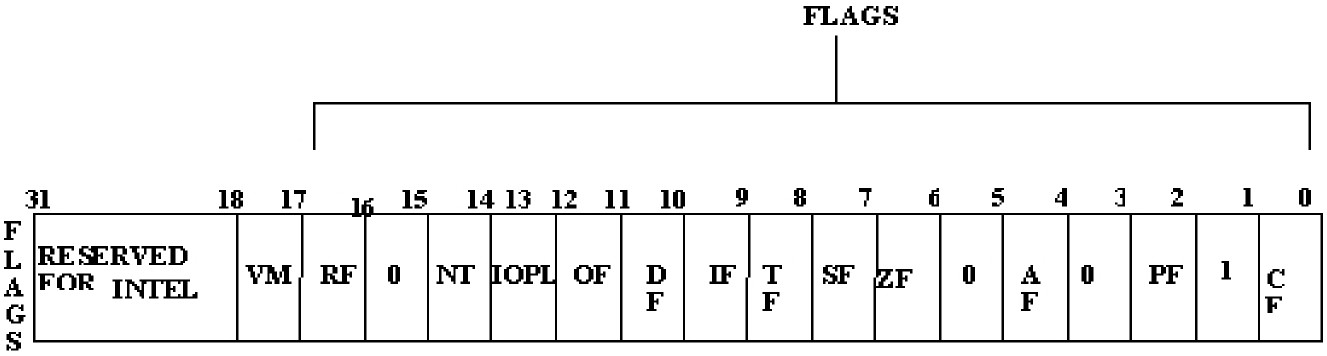
STACK SEGMENT

DATA SEGMENT

INSTRUCTION POINTER AND FLAG

31 16 15 0

	IP	EI
	FLAG	EFLA



	31 18	17	16	15	14	13 12	11	10	9	8	7	6	5	4	3	2	1	0
FLAGS	RESERVED FOR INTEL	VM	RF	0	NT	IOPL	OF	DF	IF	TF	SF	ZF	0	AF	0	PF	1	CF

FLAG REGISTER OF 80386

- The Flag register of 80386 is a 32 bit register. Out of the 32 bits, Intel has reserved bits D18 to D31, D5 and D3, while D1 is always set at 1.

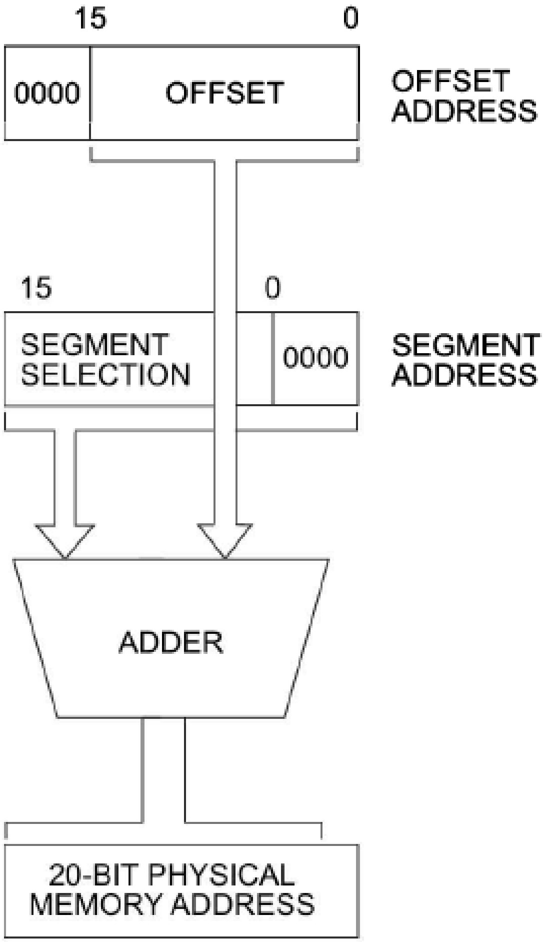
- Two extra new flags are added to the 80286 flag to derive the flag register of 80386. They are VM and RF flags.

- VM - Virtual Mode Flag: If this flag is set, the 80386 enters the virtual 8086 mode within the protection mode.
- RF- Resume Flag: This flag is used with the debug register breakpoints.
- Segment Descriptor Registers: This registers are not available for programmers, rather they are internally used to store the descriptor information, like attributes, limit and base addresses of segments

- Control Registers: The 80386 has three 32 bit control registers CR0, CR2 and CR3 to hold global machine status
- System Address Registers: Four special registers are defined to refer to the descriptor tables supported by 80386.
- Debug and Test Registers: Intel has provide a set of 8 debug registers for hardware debugging.

Real and Protected Mode

Real Address Mode ADDRESS CALCULATION



Real Address Mode ADDRESS CALCULATION

- Act as a fast 8086
- It addresses only 1 M byte of physical memory using A0-A19.
- In real addressing mode of operation of 80286, it just acts as a fast 8086. The instruction set is upward compatible with that of 8086.
- The 80286 addresses only 1Mbytes of physical memory using A0- A19. The lines A20-A23 are not used by the internal circuit of 80286 in this mode.
- In real address mode, while addressing the physical memory, the 80286 uses BHE along with A0- A19. The 20-bit physical address is again formed in the same way as that in 8086.

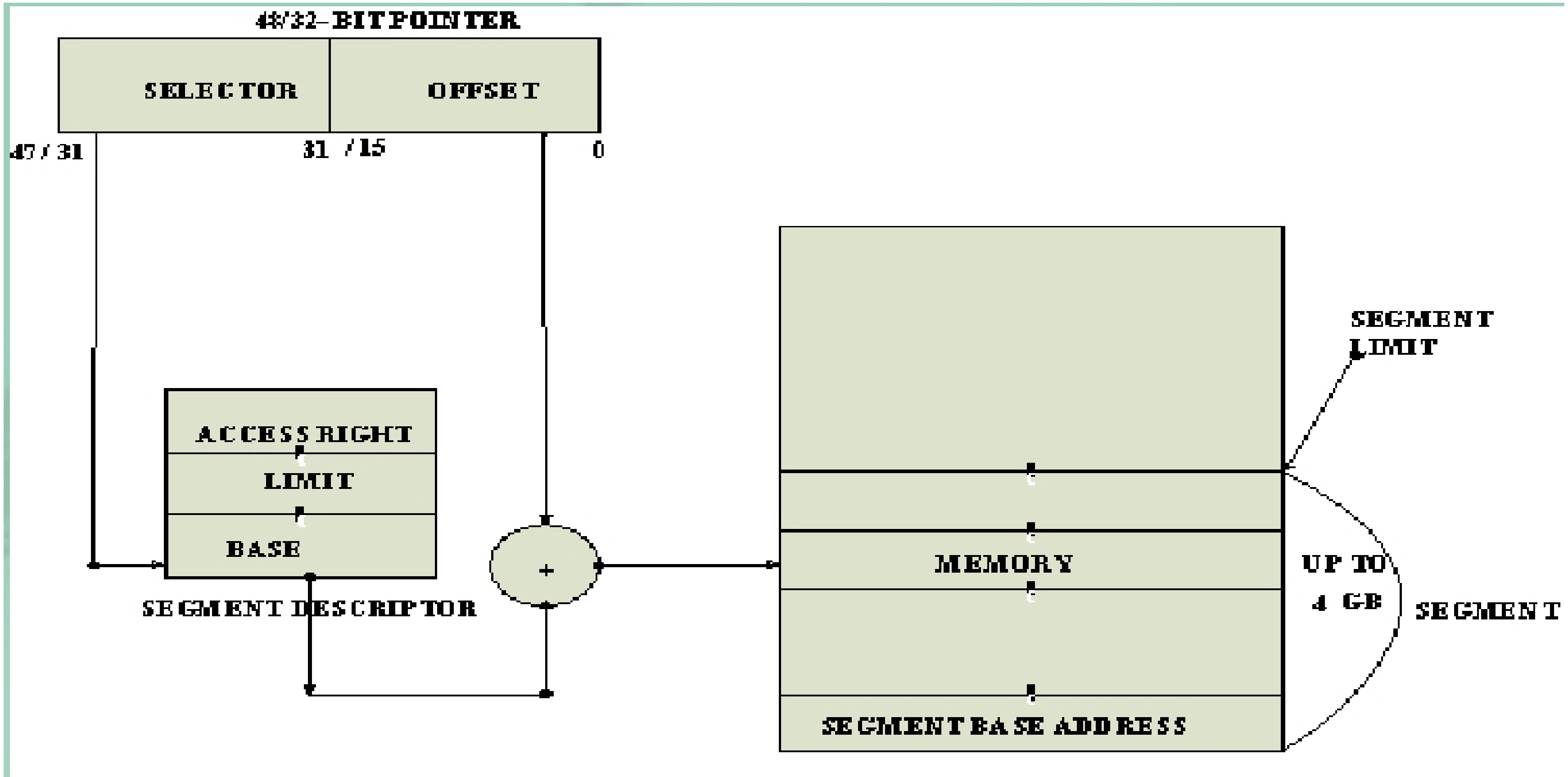
Protected Mode of 80386:

- All the capabilities of 80386 are available for utilization in its protected mode of operation.

- The 80386 in protected mode support all the software written for 80286 and 8086 to be executed under the control of memory management and protection abilities of 80386.

- The protected mode allows the use of additional instruction, addressing

modes and capabilities of 80386.



Protected Mode Addressing Without Paging Unit

Addressing in protected mode

- In this mode, the contents of segment registers are used as selectors to address descriptors which contain the segment limit, base address and access rights byte of the segment.

- The effective address (offset) is added with segment base address to calculate linear address.

- This linear address is further used as physical address, if the paging unit is disabled, otherwise the paging unit converts the linear address into physical address.

Addressing in protected mode

- The paging unit is a memory management unit enabled only in protected mode.
- The paging mechanism allows handling of large segments of memory in terms of pages of 4Kbyte size.
- The paging unit operates under the control of segmentation unit.
- The paging unit if enabled converts linear addresses into physical address, in protected mode.

Paging

Paging Unit:

- The paging unit of 80386 uses a two level table mechanism to convert a linear address provided by segmentation unit into physicaladdresses.

- The paging unit converts the complete map of a task into pages, each of size 4K. The task is further handled in terms of its page, rather than segments.

- The paging unit handles every task in terms of three components namely page directory, page tables and page itself.

Paging Unit:

- The Paging unit organizes the physical memory in terms of pages of 4kbytes size each.

- Paging unit works under the control of the segmentation unit, i.e. each segment is further divided into pages.

- The virtual memory is also organizes in terms of segments and pages by the memory management unit.

- Paging unit converts linear addresses into physical addresses.

Paging Unit

- The control and attribute PLA checks the privileges at the page level.

- Each of the pages maintains the paging information of the task.

- The limit and attribute PLA checks segment limits and attributes at segment level to avoid invalid accesses to code and data in the memory segments.

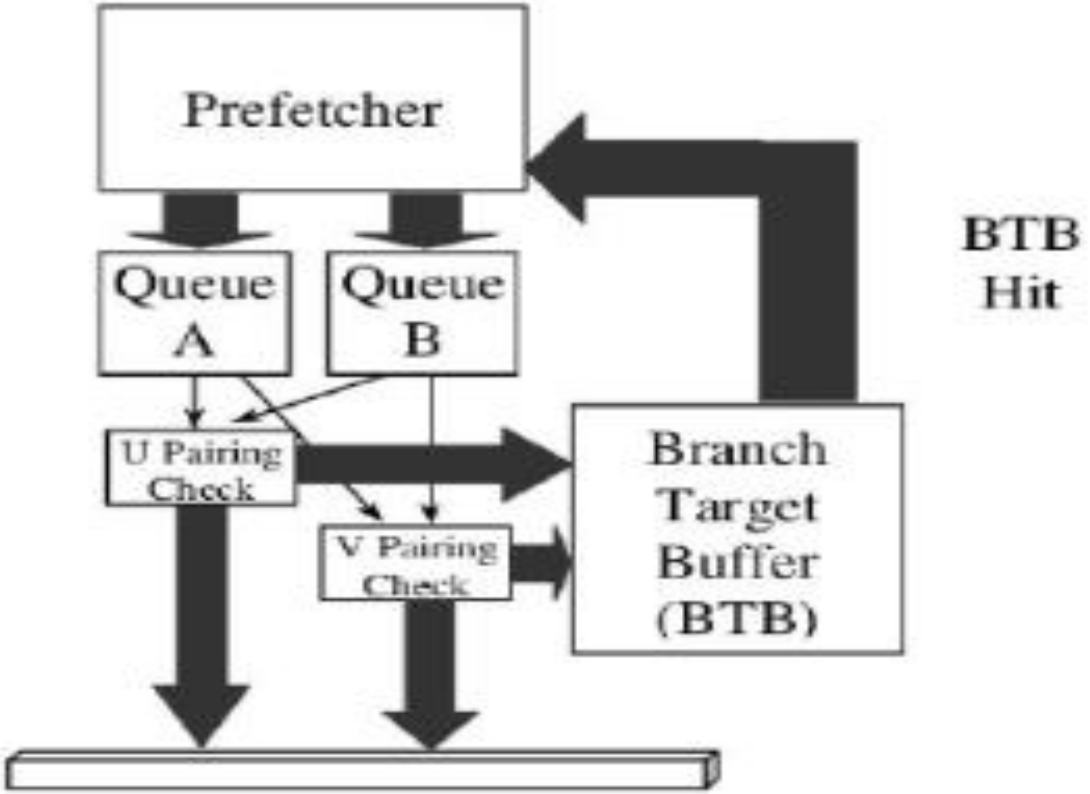
Salient Features of Pentium

Features of Pentium

- 64 bit data bus
- Instruction cache
- Data cache
- Two parallel integer execution units
- Floating point unit
- Branch Prediction Logic
- Data Integrity and Error Detection
- Dual Integer Processor
- Functional redundancy check
- Superscalar architecture

Branch Prediction

Branch Prediction

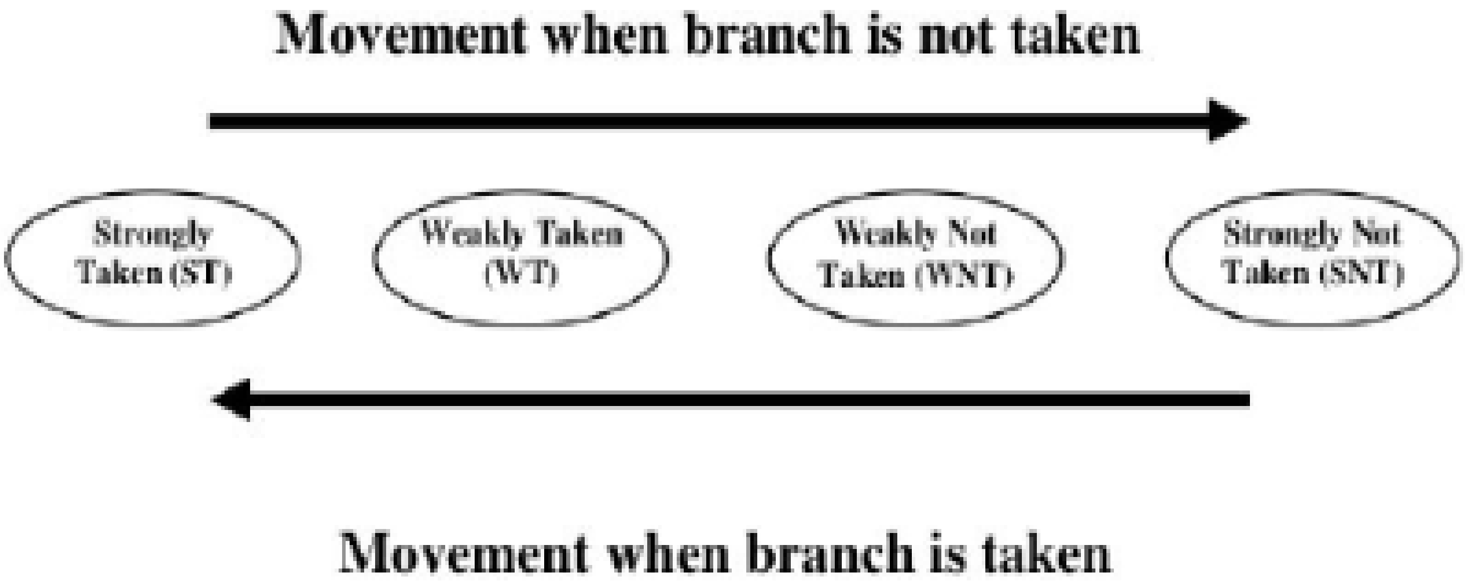


The history bits can indicate one of four possible states.

1. **Strongly Taken**: The history bits are initialized to this state when the entry is first made. In addition, if a branch marked weakly taken is taken again, it is upgraded to strongly taken stage. When a branch marked strongly taken is not taken the next time, it is downgraded to weakly taken.
2. **Weakly Taken:** It is upgraded to the strongly taken state when a branch marked weakly taken is taken again. When the corresponding marked branch is not taken, then it is downgraded to weakly not taken state. In D1 stage, a hit on strongly or weakly taken entry will result in a positive prediction. (i.e., the branch is predicted taken)

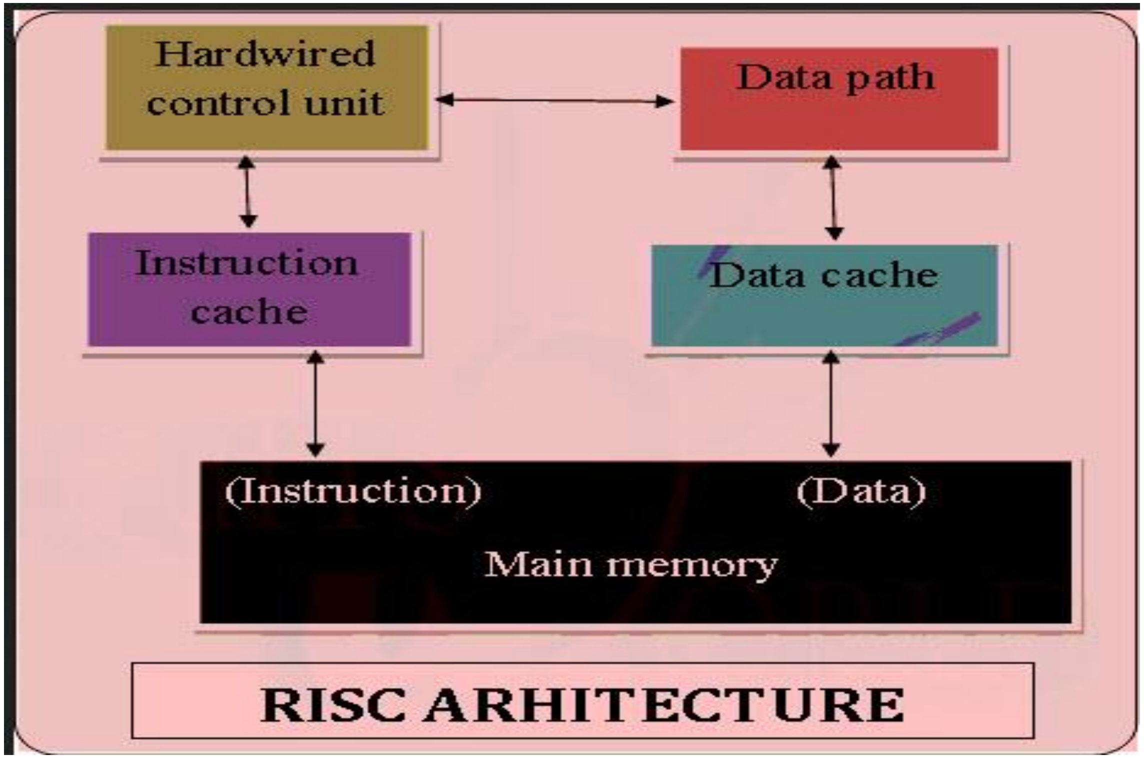
3. **Weakly Not Taken**: If a branch marked weakly not taken is taken again, it is upgraded to the weakly taken state. When a branch marked weakly not taken is not taken the next time, it is downgraded to strongly not taken.

4. **Strongly Not Taken**: If a branch marked strongly not taken is taken again it is upgraded to the weakly not taken state. When a branch marked strongly not taken is not taken the next time, it remains in the strongly not taken state. In D1 Stage, a hit on weakly not taken or Strongly not taken entry will result in a negative prediction (i.e., the branch is predicted not taken)


Movement when branch is not taken
Strongly Taken (ST)
Weakly Taken (WT)
Weakly Not Taken (WNT)
Strongly Not Taken (SNT)
Movement when branch is taken

Overview of RISC Processors

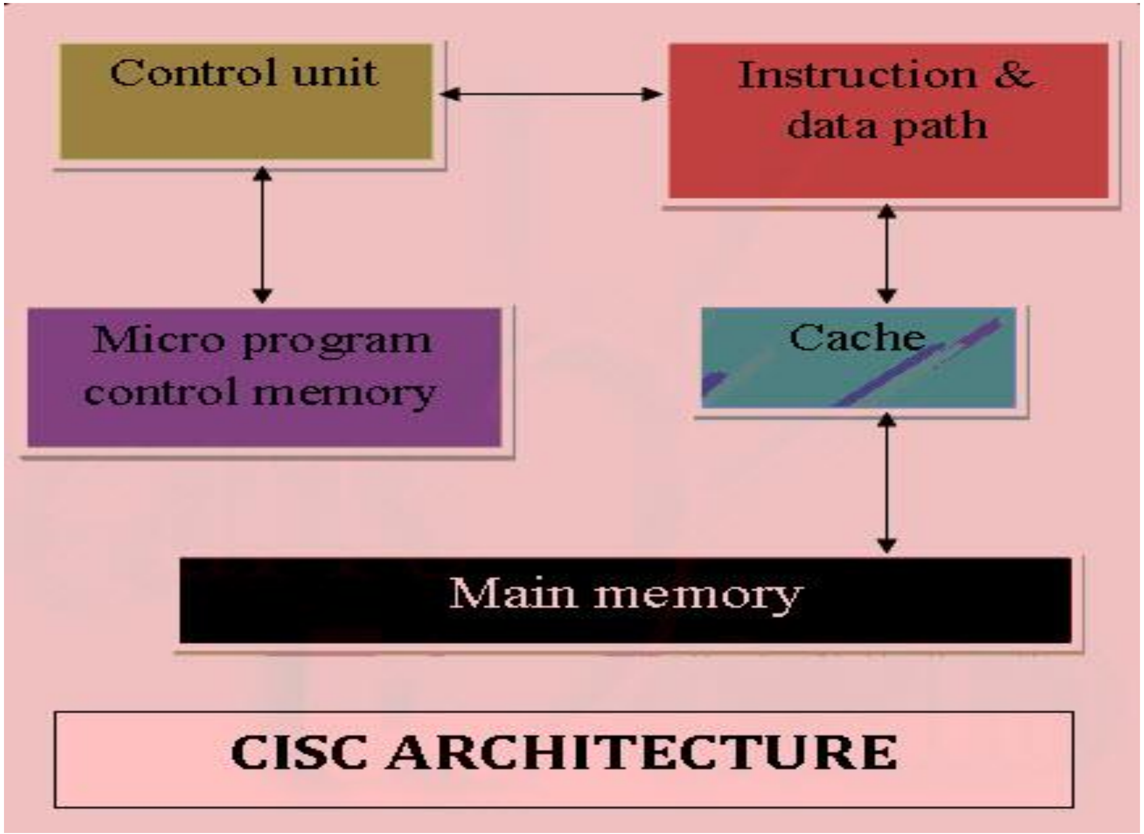
RISC Architecture



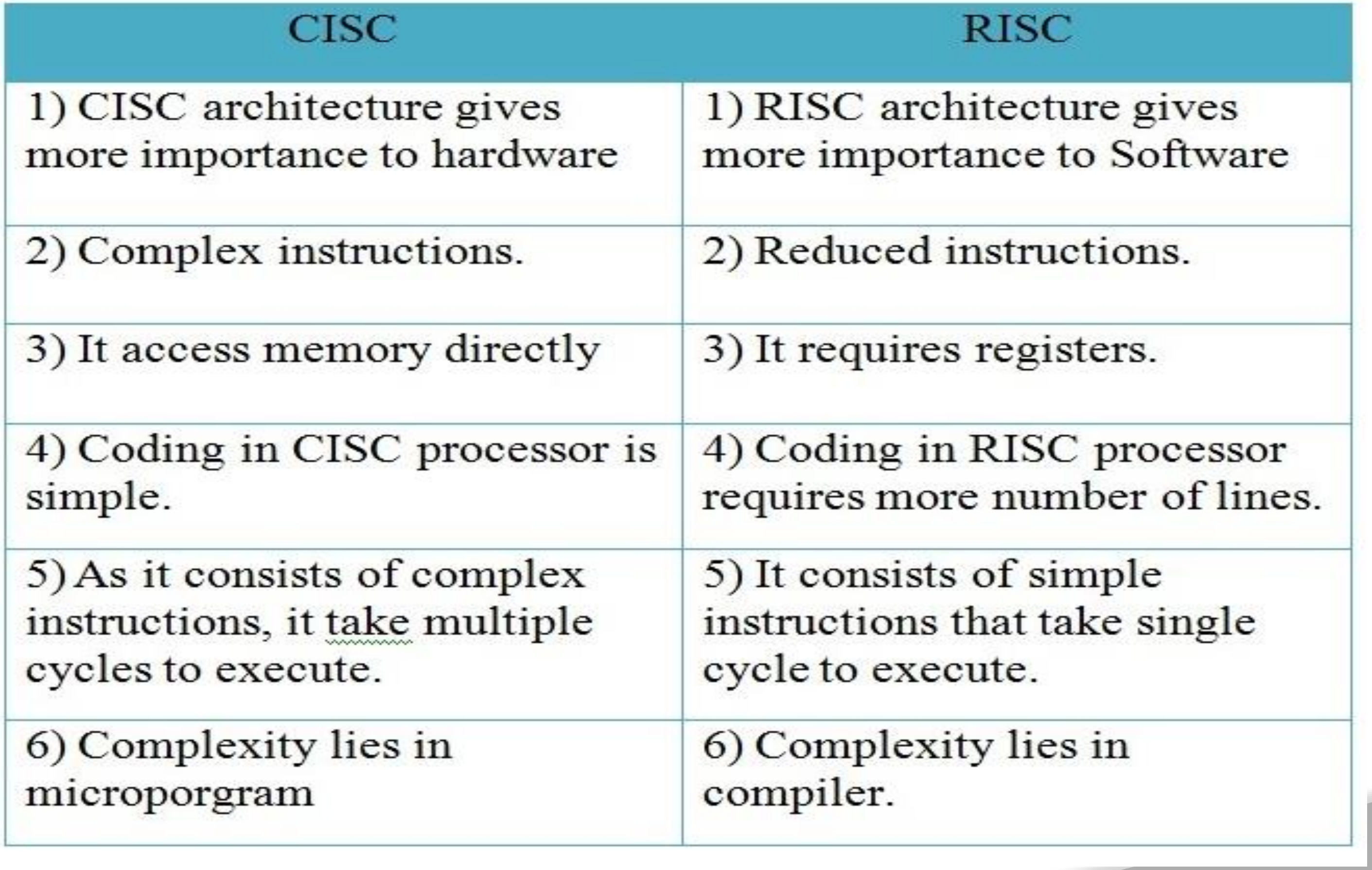
The features of RISC

- The demand of decoding is less
- Few data types in hardware
- General purpose register Identical
- Uniform instruction set
- Simple addressing nodes

CISC Architecture



Comparison between CISC and RISC



CISC	RISC
1) CISC architecture gives more importance to hardware	1) RISC architecture gives more importance to Software
2) Complex instructions.	2) Reduced instructions.
3) It access memory directly	3) It requires registers.
4) Coding in CISC processor is simple.	4) Coding in RISC processor requires more number of lines.
5) As it consists of complex instructions, it take multiple cycles to execute.	5) It consists of simple instructions that take single cycle to execute.
6) Complexity lies in microporgram	6) Complexity lies in compiler.

The Advantages of RISC architecture

- RISC(Reduced instruction set computing)architecture has a set of instructions, so high-level language compilers can produce more efficient code
- It allows freedom of using the space on microprocessors because of its simplicity.
- Many RISC processors use the registers for passing arguments and holding the local variables.
- RISC functions use only a few parameters, and the RISC processors cannot use the call instructions, and therefore, use a fixed length instruction which is easy to pipeline.
- The speed of the operation can be maximized and the execution time can be minimized.

  Very less number of instructional formats, a few numbers of instructions and a few addressing modes are needed.

The Disadvantages of RISC architecture

- Mostly, the performance of the RISC processors depends on the programmer or compiler as the knowledge of the compiler plays a vital role while changing the CISC code to a RISC code
- While rearranging the CISC code to a RISC code, termed as a code expansion, will increase the size. And, the quality of this code expansion will again depend on the compiler, and also on the machine’s instruction set.
- The first level cache of the RISC processors is also a disadvantage of the RISC, in which these processors have large memory caches on the chip itself. For feeding the instructions, they require very fast memory systems.

Advantages of CISC architecture

- Microprogramming is easy assembly language to implement, and less expensive than hard wiring a control unit.
- The ease of micro coding new instructions allowed designers to make CISC machines upwardly compatible:
- As each instruction became more accomplished, fewer instructions could be used to implement a given task.

Disadvantages of CISC architecture

- The performance of the machine slows down due to the amount of clock time taken by different instructions will be dissimilar
- Only 20% of the existing instructions is used in a typical programming event, even though there are various specialized instructions in reality which are not even used frequently.
- The conditional codes are set by the CISC instructions as a side effect of each instruction which takes time for this setting – and, as the subsequent instruction changes the condition code bits – so, the compiler has to examine the condition code bits before this happens.

MODULE-V

8051 MICROCONTROLLER ARCHITECTURE

COURSE OUTCOMES

CO1	Outline the internal architecture of 8085, 8086 and 8051 microcomputers to study the functionality.
CO5	Interpret the functionality of various types of interrupts and their structure for controlling the processor or controller and program execution flow.
CO10	Interpret the internal building blocks and registers of 8051 microcontroller used to perform serial data transfer, timer operation, interfacing of memory and I/O devices.
CO11	Build necessary hardware and software interface using microcomputer based systems to provide solution for real world problems.

8051 Microcontroller Architecture

Disadvantages of Microprocessor

- The overall system cost is high.
- A large sized PCB is required for assembling all the components.
- Overall product design requires more time.
- Physical size of the product is big.
- A discrete components are used, the system is not reliable.

Advantages of Microcontroller based System

- As the peripherals are integrated into a single chip, the overall system cost is very less.

- As the peripherals are integrated with a microprocessor the system is more reliable.

- Though microcontroller may have on chip ROM,RAM and I/O ports, addition ROM, RAM I/O ports may be interfaced externally if required.

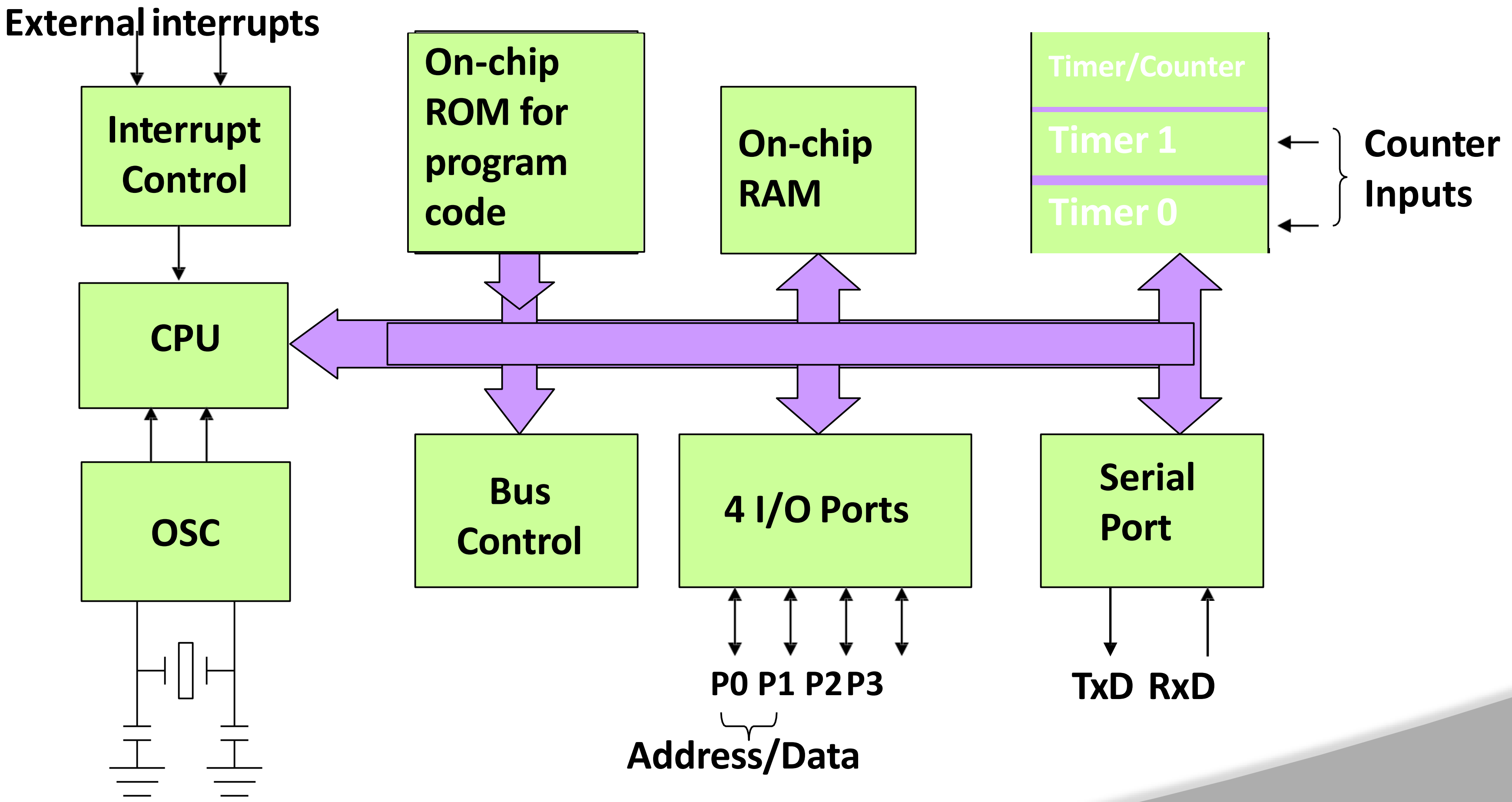
- On chip ROM provide a software security.

8051 Basic Component

- 4K bytes internal ROM
- 128 bytes internal RAM
- Four 8-bit I/O ports (P0 - P3).
- Two 16-bit timers/counters
- One serial interface
- 64k external memory for code
- 64k external memory for data
- 210 bit addressable

- Microcontroller

Block Diagram

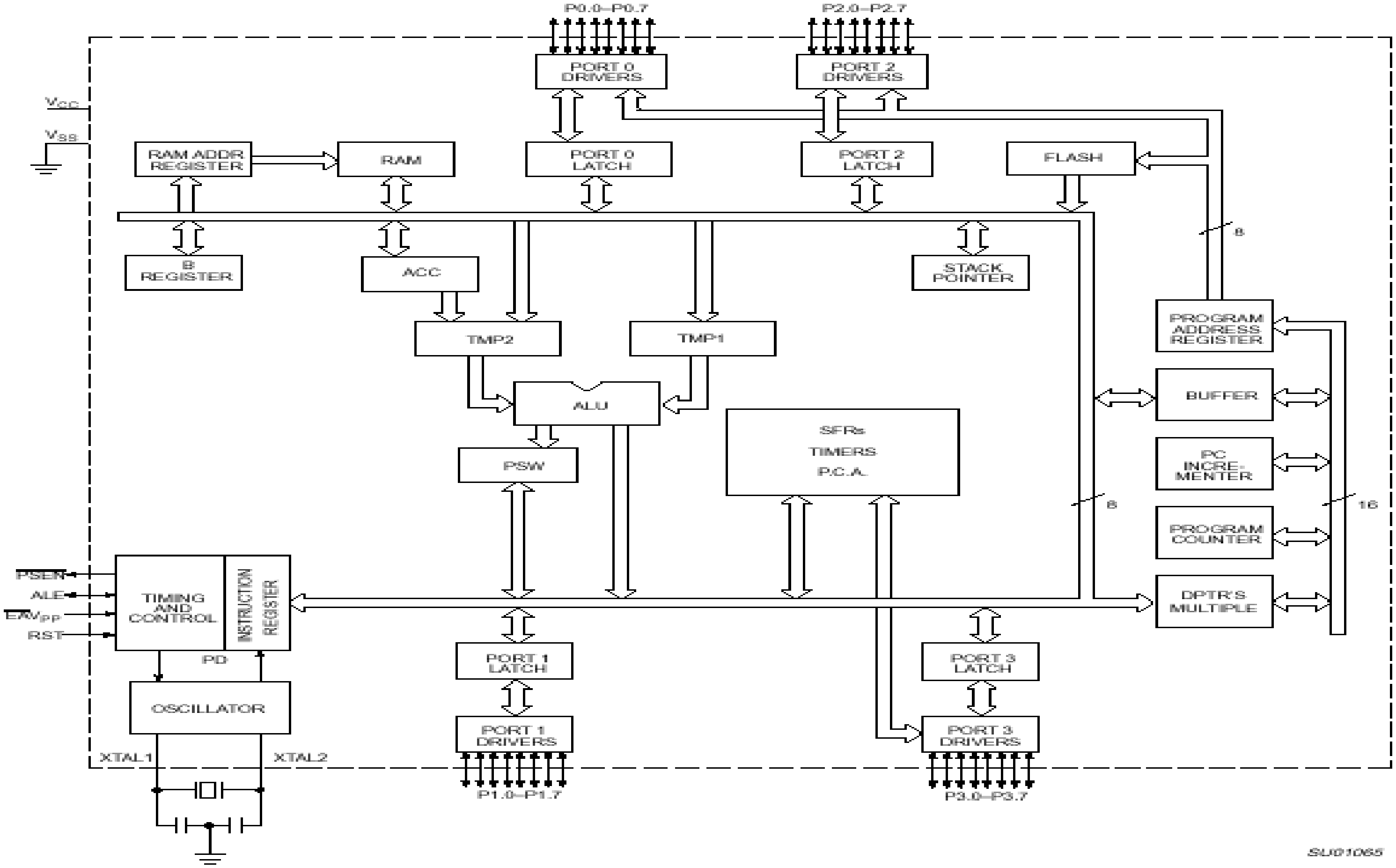


- The system bus connects all the support devices to the CPU.
- The system bus consists of

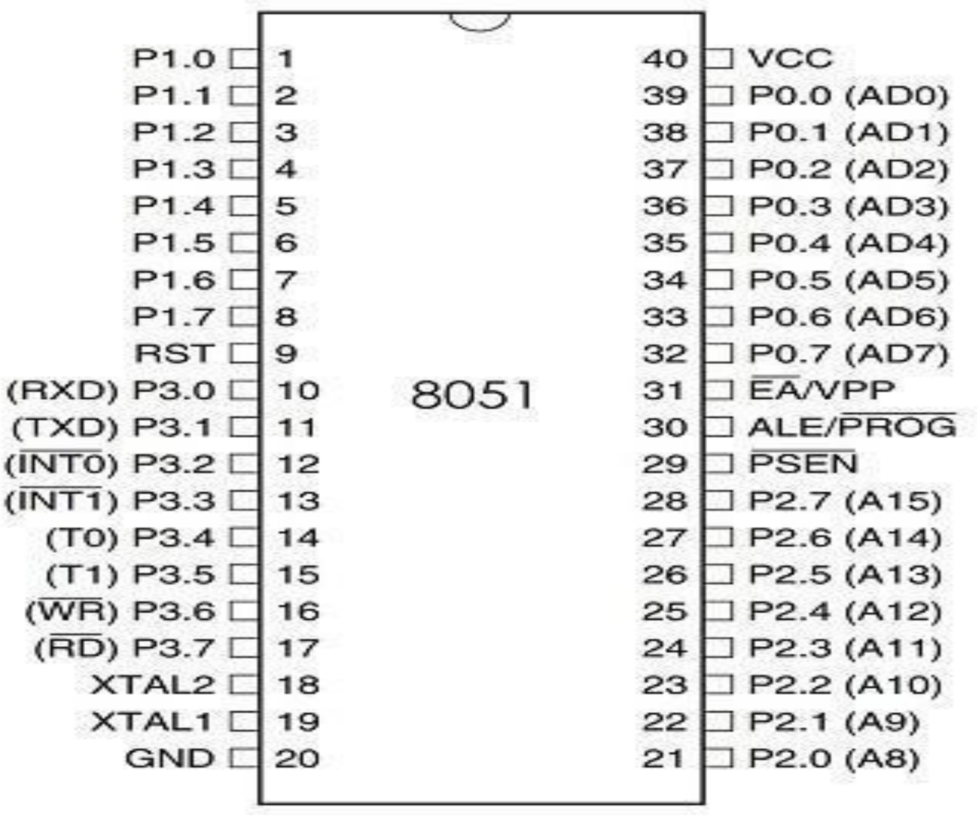
  8-bit data bus

  16-bit address bus and bus control signals.
- All other devices like program memory, ports, data memory, serial interface, interrupt control, timers, and the CPU are all interfaced together through the system bus.

Internal Block Diagram of 8051



Pin Diagram of 8051



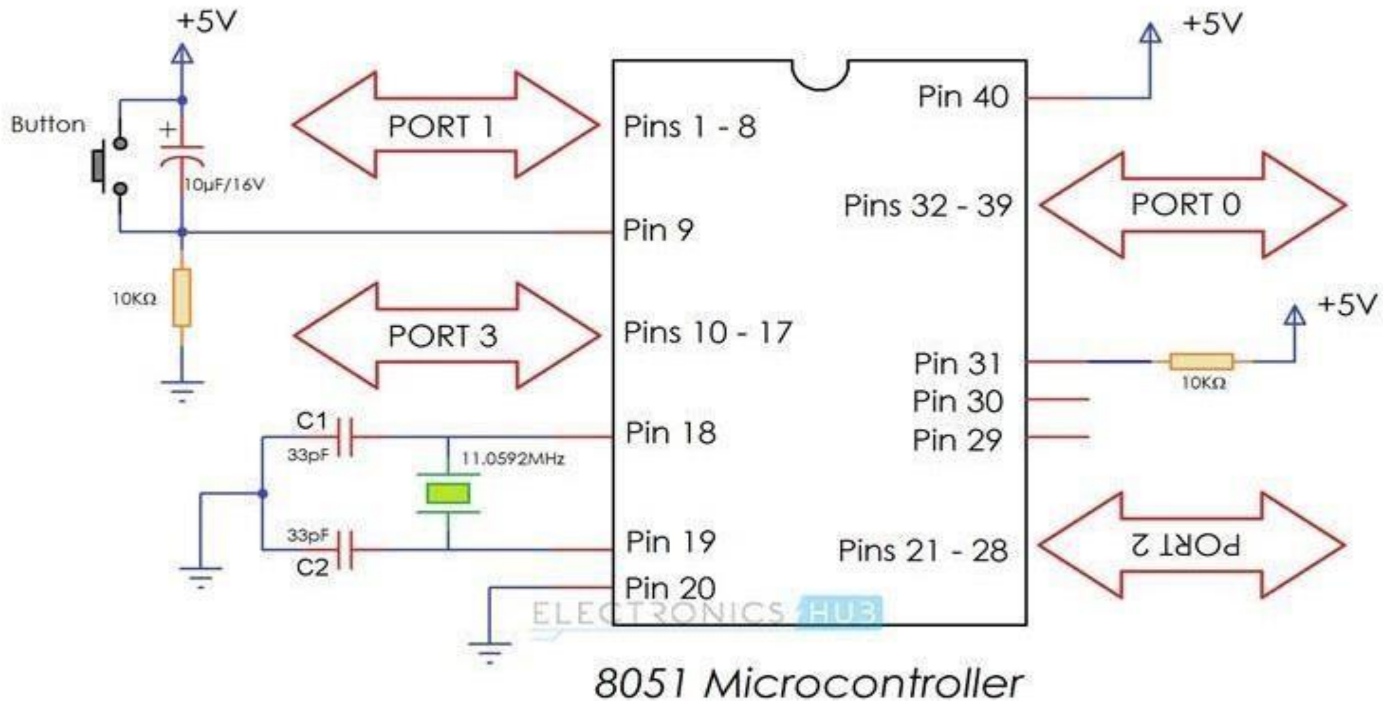
40 - PIN DIP

- 8051 microcontrollers have 4 I/O ports each of 8-bit, which can be configured as input or output. Hence, total 32 input/output pins allow the microcontroller to be connected with the peripheral devices.
- **Pin configuration**, i.e. the pin can be configured as 1 for input and 0 for output as per the logic state.

  Input/output (I/O) pin − All the circuits within the microcontroller must be connected to one of its pins except P0 port because it does not have pull-up resistors built-in.

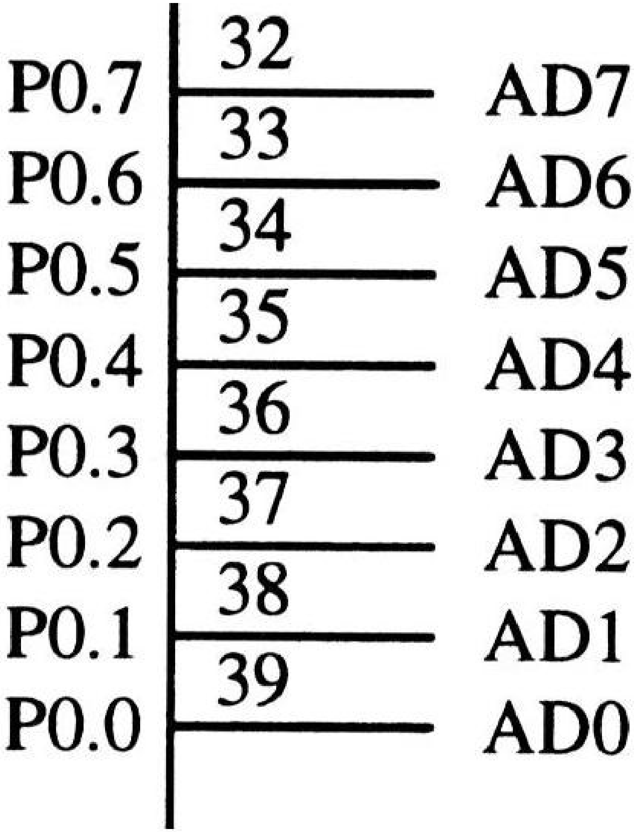
  Input pin − Logic 1 is applied to a bit of the P register. The output FE transistor is turned off and the other pin remains connected to the power supply voltage over a pull-up resistor of high resistance.

Basic circuit of 8051



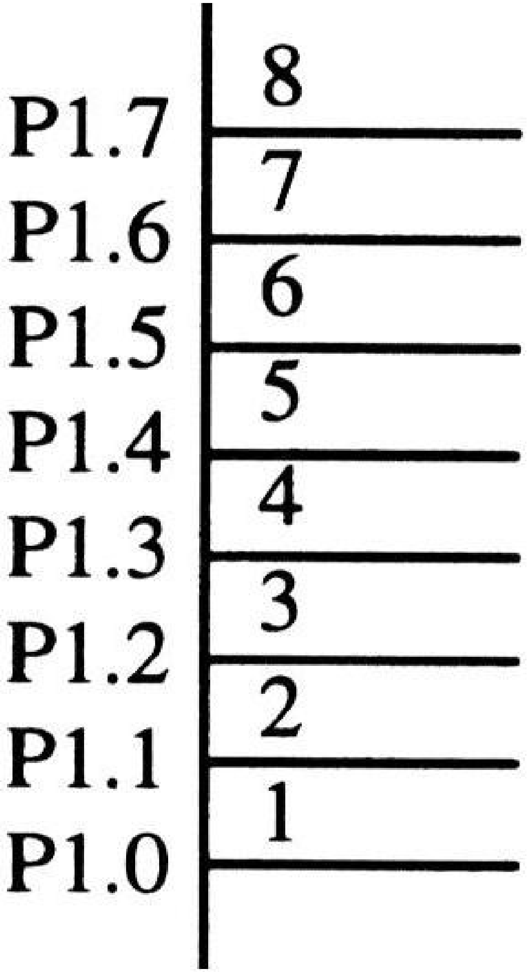
PORT 0-Description

- *8-bit R/W -General Purpose I/O*
- *Or acts as amultiplexed low byte address and data bus for external memory design*



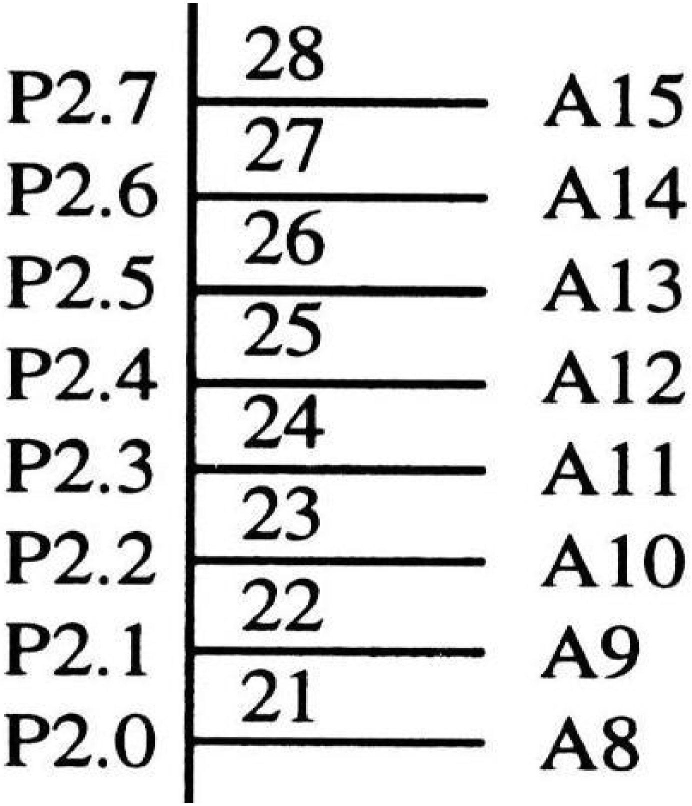
PORT 1 -Description

- ***Only* 8-bit R/W - General Purpose I/O**



PORT 2 -Description

- *8-bit R/W - General Purpose I/O*
- *Or high byte of the address bus for external memory design*



PORT 3 - Description

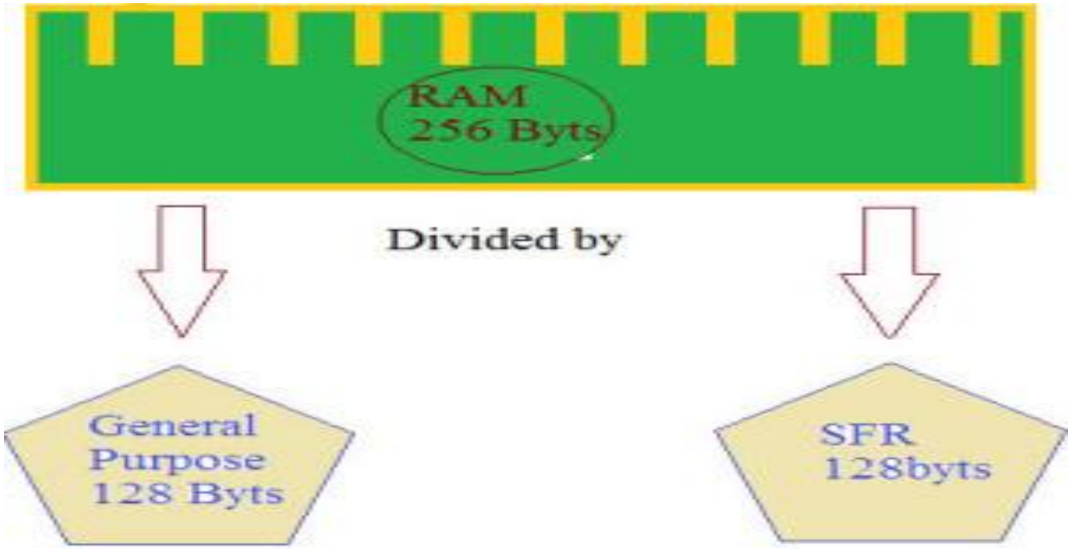
PORT 3 Pin	Function	Description
P3.0	RXD	Serial Input
P3.1	TXD	Serial Output
P3.2	INT0	External Interrupt 0
P3.3	INT1	External Interrupt 1
P3.4	T0	Timer 0
P3.5	T1	Timer 1
P3.6	WR	External Memory Write
P3.7	RD	External Memory Read

Register set of 8051

Types of Registers

The 8051 microcontroller contains mainly two types of registers:

- General-purpose registers (Byte addressable registers)
- Special function registers (Bit addressable registers)



General Purpose Registers

- There are four different bank registers with each bank having 8 addressable 8-bit registers, and only one bank register can be accessed at a time.

- But, by changing the bank register’s number in the flag register, we can access other bank registers

Special Function Registers

- The special function registers including the Accumulator, Register B, Data pointer, PCON, PSW, etc., are designed for a predetermined purpose during manufacturing with the address 80H to FFH, and this area cannot be used for the data or program storage purpose.

- These registers can be implemented by bit address and byte address registers.

- 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

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

PSW Register

CY	CA	F0	RS1	RS0	OV	-	P

CY	PSW.7	Carry Flag
AC	PSW.6	Auxiliary Carry Flag
F0	PSW.5	Flag 0 available to user for general purpose.
RS1	PSW.4	Register Bank selector bit 1
RS0	PSW.3	Register Bank selector bit 0
OV	PSW.2	Overflow Flag
-	PSW.1	User definable FLAG
P	PSW.0	Parity FLAG. Set/ cleared by hardware during instruction cycle to indicate even/odd number of 1 bit in accumulator.

RS1	RS2	Register Bank	Address
0	0	0	00H-07H
0	1	1	08H-0FH
1	0	2	10H-17H
1	1	3	18H-1FH

Modes of timer operation

TIMER/COUNTER

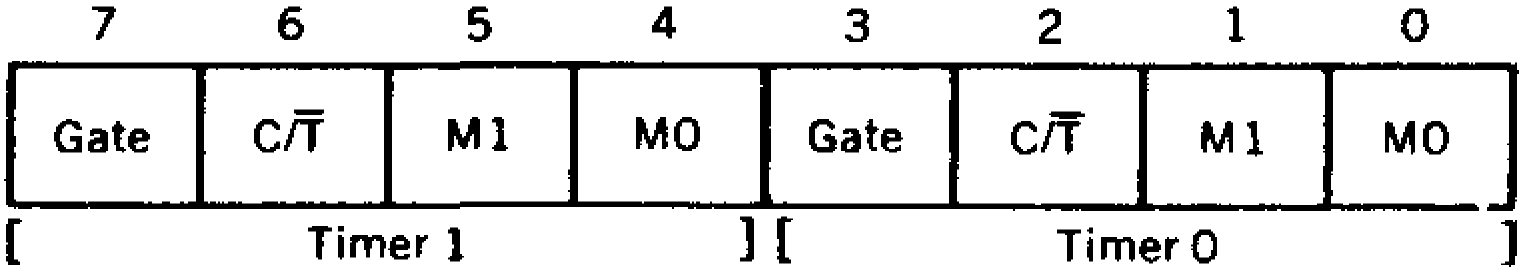
- 8051 has two 16-bit programmable timers/counters. They can be configured to operate either as timers or as event counters. The names of the two counters are T0 and T1 respectively.
- The timer content is available in four 8-bit special function registers, viz, TL0,TH0,TL1 and TH1 respectively.
- In the "timer" function mode, the counter is incremented in every machine cycle. Thus, one can think of it as counting machine cycles. Hence the clock rate is 1/12 th of theoscillatorfrequency.
- In the "counter" function mode, the register is incremented in response to a 1 to 0 transition at its corresponding external input pin (T0 or T1). It requires 2 machine cycles to detect a high to low.

Operation of Timer/Counter

- The operation of the timers/counters is controlled by two special function registers, TMOD and TCON respectively.

Timer Mode control (TMOD) Special Function Register:

- TMOD register is not bit addressable.
- TMOD Address: 89 H



7	6	5	4	3	2	1	0
Gate	C/$\overline{T}$	M1	M0	Gate	C/$\overline{T}$	M1	M0
[Timer 1]				[Timer 0]			

Timer/ Counter control logic:

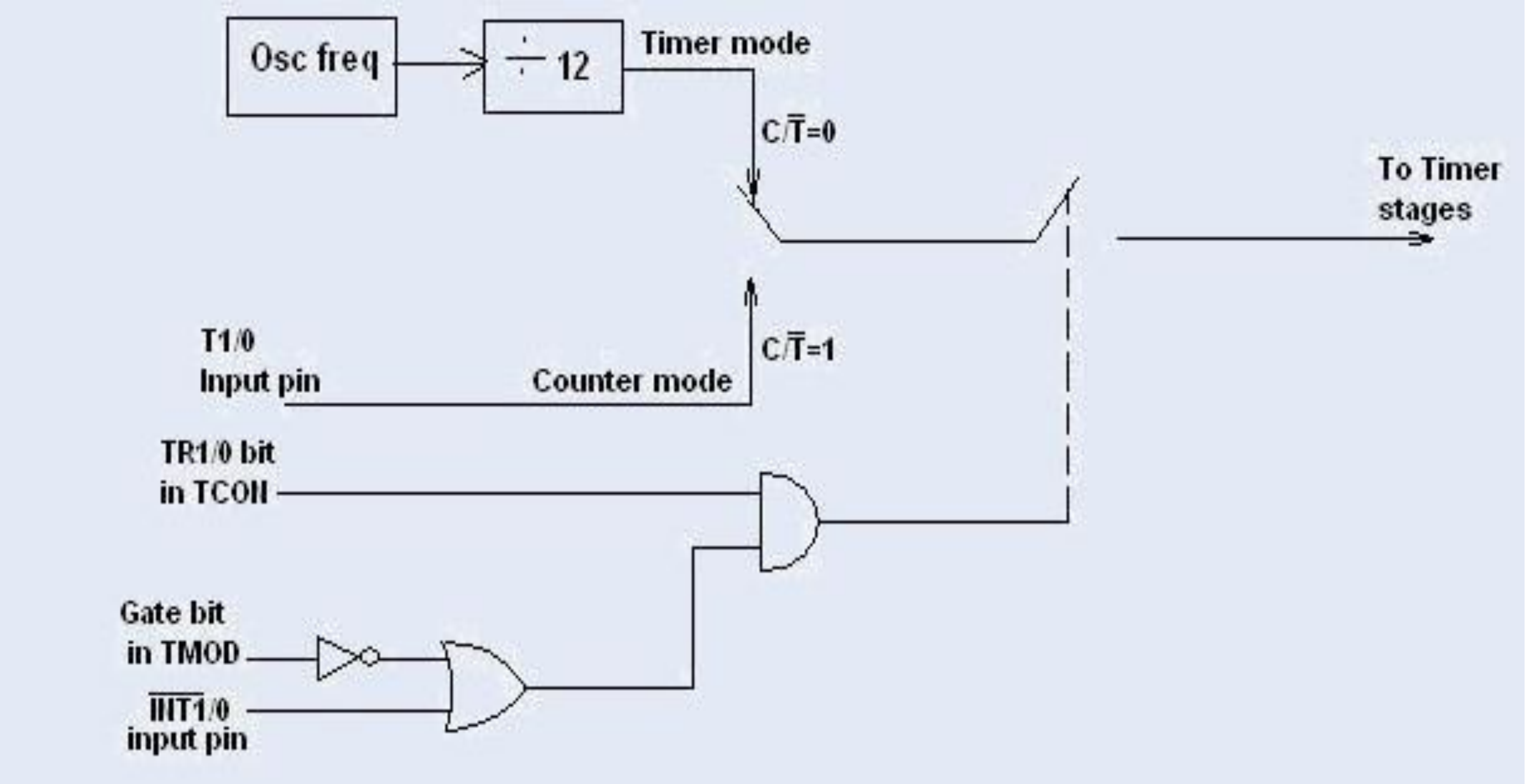
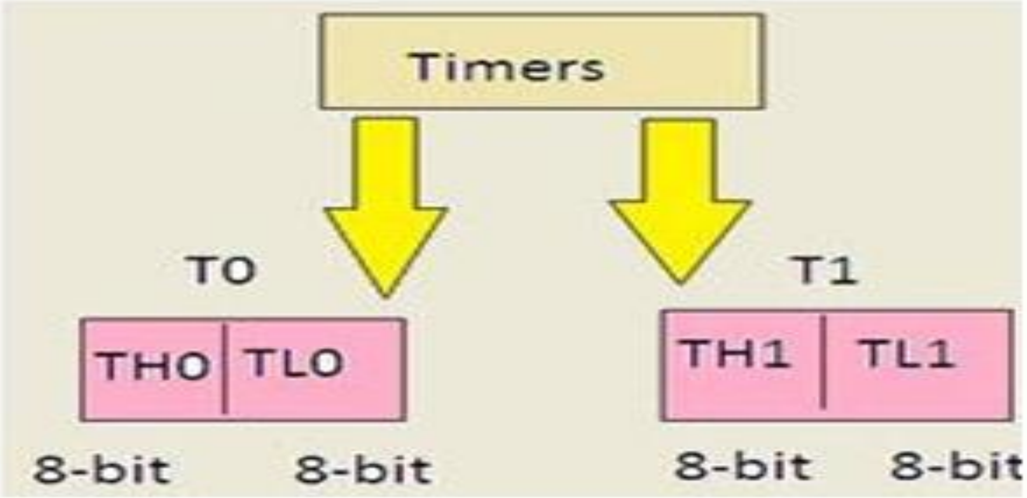


Figure: Timer/ Counter control logicDiagram

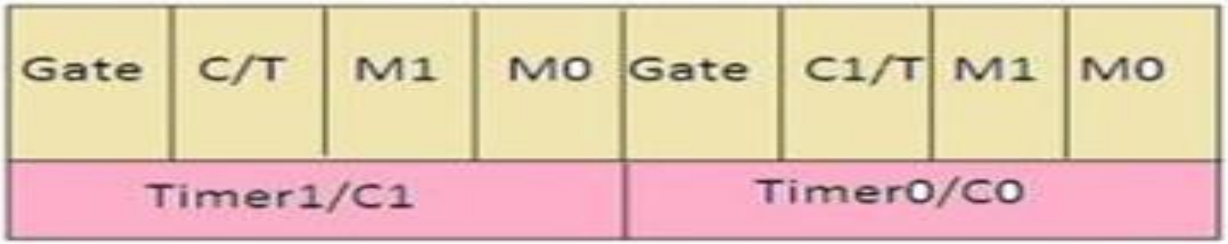
Difference between a Timer and a Counter

Timer	Counter
The register incremented for every machine cycle.	The register is incremented considering 1 to 0 transitions at its corresponding to an external input pin (T0, T1).
Maximum count rate is 1/12 of the oscillator frequency.	Maximum count rate is 1/24 of the oscillator frequency.
A timer uses the frequency of the internal clock, and generates delay.	A counter uses an external signal to count pulses.

Timers & Counters



Timer Mode Control (TMOD):



Gate	C/T	M1	M0	Gate	C1/T	M1	M0
Timer1/C1				Timer0/C0			

Gate: If the gate bit is set to „0“, then we can start and stop the “software” timer in the same way. If the gate is set to „1“, then we can perform hardware timer.

C/T: If the C/T bit is „1“, then it is acting as a counter mode, and similarly when set C+
=/T bit is „0“; it is acting as a timer mode.

Mode select bits: The M1 and M0 are mode select bits, which are used to select the timer operations. There are four modes to operate the timers.

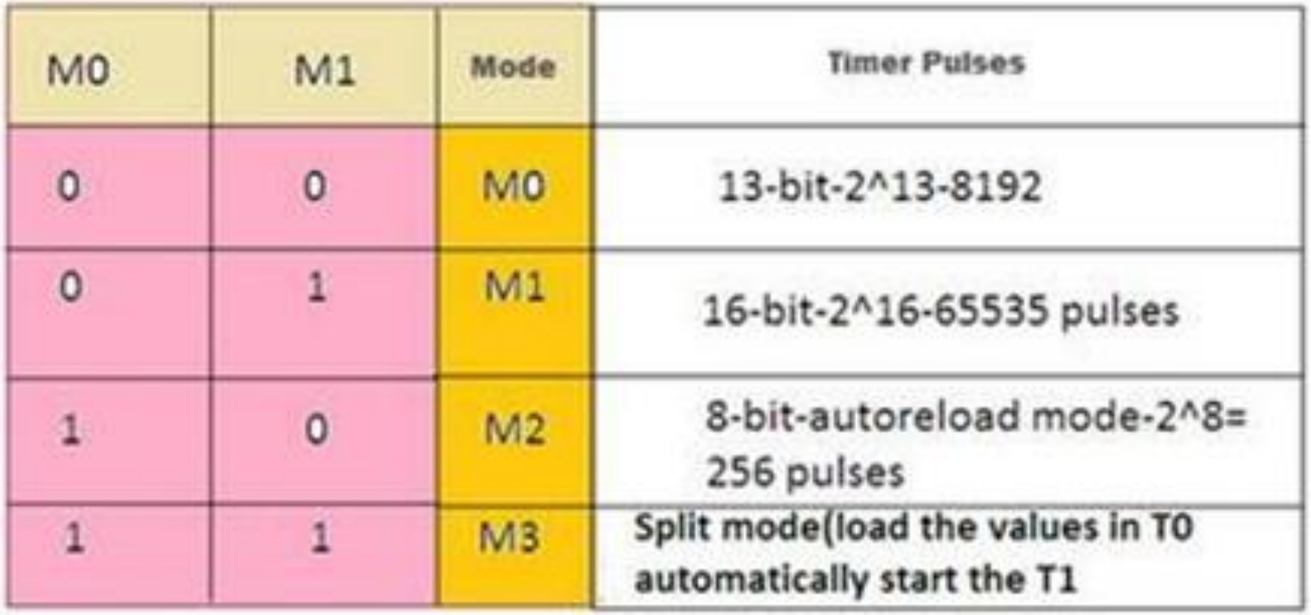
Mode 0: This is a 13-bit mode that means the timer operation completes with “8192” pulses.

Mode 1: This is a16-bit mode, which means the timer operation completes with maximum clock pulses that “65535”.

Mode 2: This mode is an 8-bit auto reload mode, which means the timer operation completes with only “256” clock pulses.

Mode 3: This mode is a split-timer mode, which means the loading values in T0 and automatically starts the T1.

Mode selection Bits



M0	M1	Mode	Timer Pulses
0	0	M0	13-bit-2^13-8192
0	1	M1	16-bit-2^16-65535 pulses
1	0	M2	8-bit-autoreload mode-2^8= 256 pulses
1	1	M3	Split mode(load the values in T0 automatically start the T1

Timer modes of operation

Timer Mode-0:

In this mode, the timer is used as a 13-bit UP counter as follows.

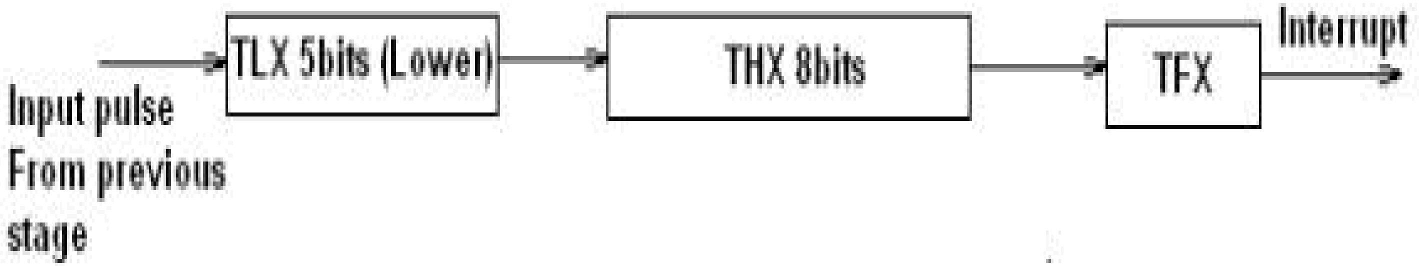
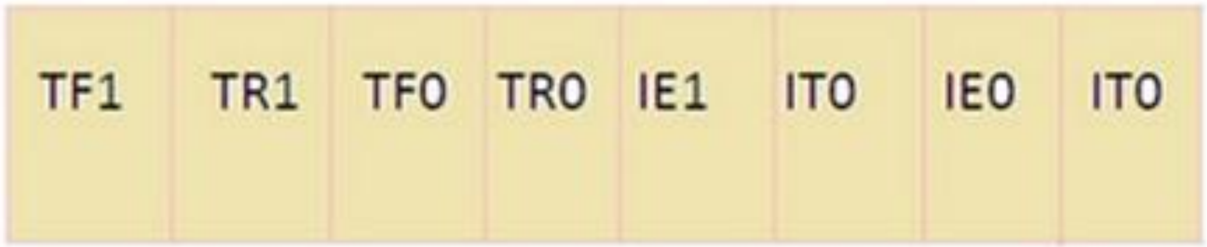


Fig: Operation of Timer in Mode 2

- ➢The lower 5 bits of TLX and 8 bits of THX are used for the 13 bit count. Upper 3 bits of TLX are ignored. When the counter rolls over from all 0's to all 1's, TFX flag is set and an interrupt is generated.

Timer Control Register (TCON):



TF1	TR1	TFO	TRO	IE1	ITO	IEO	ITO

Timer Control Register (TCON)

Timer modes of operation

- The input pulse is obtained from the previous stage. If TR1/0 bit is 1 and Gate bit is 0, the counter continues counting up. If TR1/0 bit is 1 and Gate bit is 1, then the operation of the counter is controlled by input. This mode is useful to measure the width of a given pulse fed to input.

Timer Mode-1:

- This mode is similar to mode-0 except for the fact that the Timer operates in 16-bit mode.

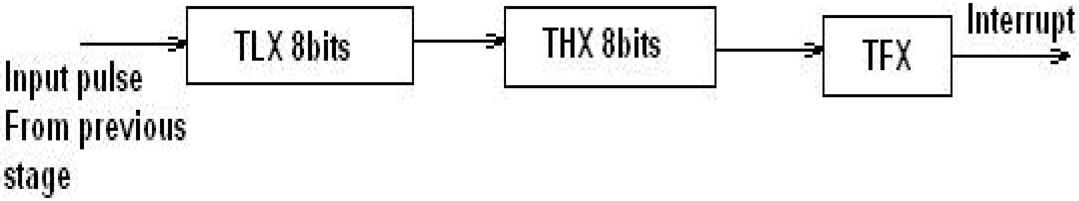


Fig: Operation of Timer in Mode 1

Timer Mode-2: (Auto-Reload Mode)

➢This is a 8 bit counter/timer operation. Counting is performed in TLX while THX stores a constant value. In this mode when the timer overflows i.e. TLX becomes FFH, it is fed with the value stored in THX. For example if we load THX with 50H then the timer in mode 2 will count from 50H to FFH. After that 50H is again reloaded. This mode is useful in applications like fixed time sampling

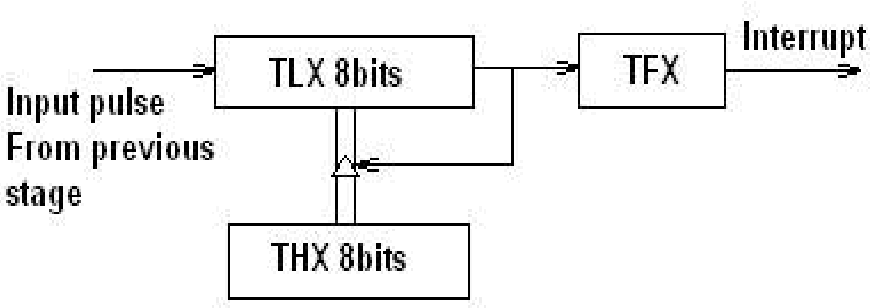


Fig: Operation of Timer in Mode2

Timer Mode-3:

Timer 1 in mode-3 simply holds its count. The effect is same as setting TR1=0.

Timer0 in mode-3 establishes TL0 and TH0 as two separate counters.

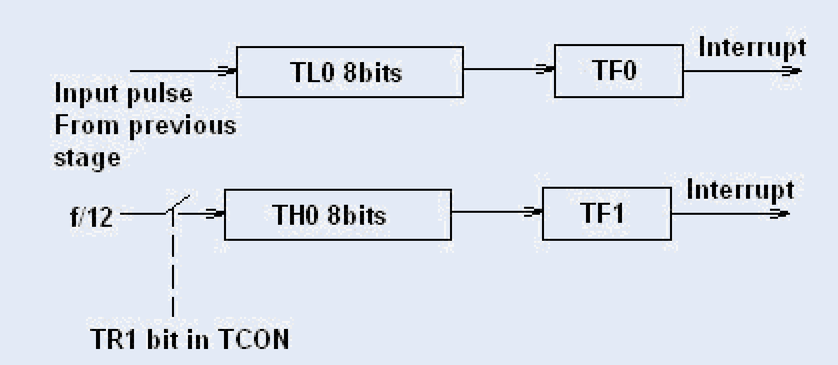


Fig: Operation of Timer in Mode 3

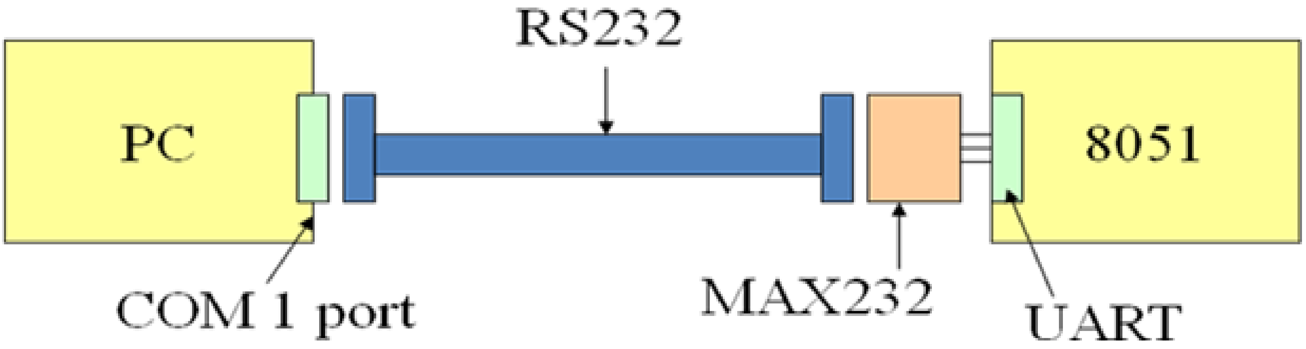
Control bits TR1 and TF1 are used by Timer-0 (higher 8 bits) (TH0) in Mode-3 while TR0 and TF0 are available to Timer-0 lower 8 bits(TL0).

Serial port operation

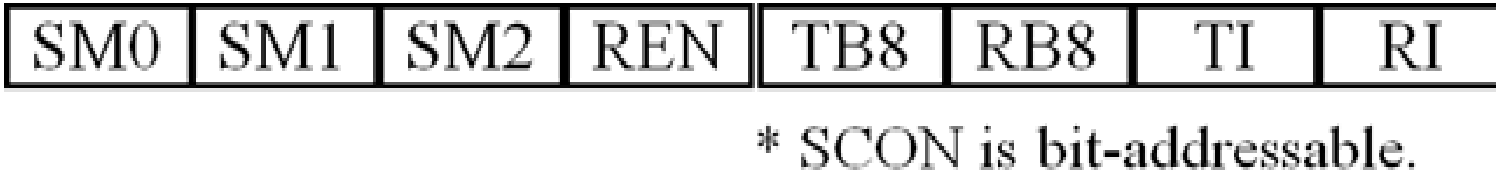
8051 Serial Communication: Sections

1. Basics of serial communication
2. 8051 connection to RS232
3. 8051 serial communication programming

8051 and PC:



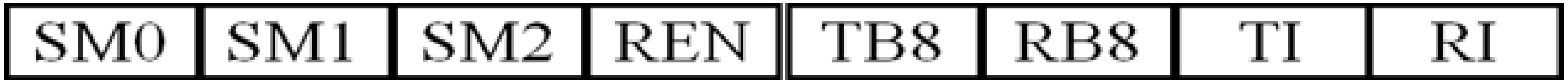
SCON Register:



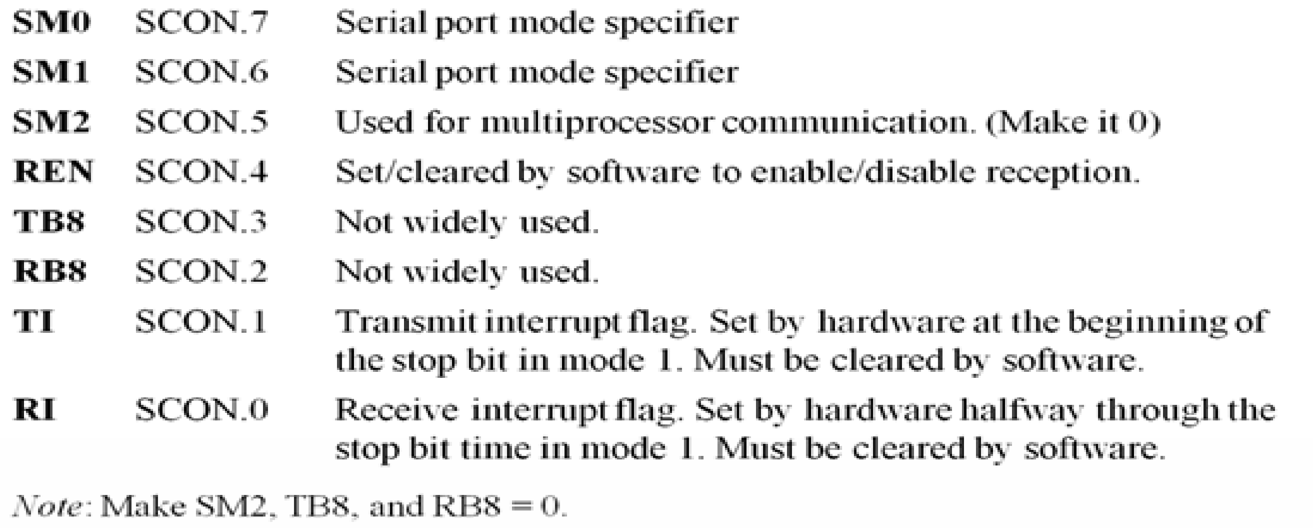
SM0	SM1	SM2	REN	TB8	RB8	TI	RI

* SCON is bit-addressable.

SCON Serial Port Control Register (Bit Addressable):



SM0	SM1	SM2	REN	TB8	RB8	TI	RI



SM0	SCON.7	Serial port mode specifier
SM1	SCON.6	Serial port mode specifier
SM2	SCON.5	Used for multiprocessor communication. (Make it 0)
REN	SCON.4	Set/cleared by software to enable/disable reception.
TB8	SCON.3	Not widely used.
RB8	SCON.2	Not widely used.
TI	SCON.1	Transmit interrupt flag. Set by hardware at the beginning of the stop bit in mode 1. Must be cleared by software.
RI	SCON.0	Receive interrupt flag. Set by hardware halfway through the stop bit time in mode 1. Must be cleared by software.

Note: Make SM2, TB8, and RB8 = 0.

Interrupt structure of 8051

Interrupts

- An *interrupt* is an external or internal event that interrupts the microcontroller to inform it that a device needs its service.

Interrupts vs. Polling

- A single microcontroller can serve several devices.
- There are two ways to do that:

– interrupts

  – polling.

Interrupts

- In Polling , the microcontroller ‘s program simply checks each of the I/O

  devices to see if any device needs servicing. If so, it performs the service.

- In the interrupt method, whenever any device needs microcontrollers

  service, it tells to microcontroller by sending an interruptsignal.

- The program which is associated with the interrupt is called the

  interrupt service routine (ISR) or ***interrupt handler***.

Steps in executing an interrupt

- Finish current instruction and saves the PC on stack.
- Jumps to a fixed location in memory depend on type of interrupt.
- Starts to execute the interrupt service routine until RETI (return from interrupt).
- Upon executing the RETI the microcontroller returns tothe place where it was interrupted. Get pop PC from stack.

Interrupt Sources

- Original 8051 has 6 sources of interrupts

1. Reset
2. Timer 0 overflow
3. Timer 1 overflow
4. External Interrupt 0
5. External Interrupt 1
6. Serial Port events buffer full, buffer empty, etc)

Interrupt Vectors

- Each interrupt has a specific place in code memory where program execution (interrupt service routine) begins.

External Interrupt 0	:	0003h
Timer 0 overflow	:	000Bh
External Interrupt 1	:	0013h
Timer 1 overflow	:	001Bh
Serial	:	0023h
Timer 2 overflow(8052+)	:	002bh

Note: that there are only 8 memory locations between vectors.

Interrupt Enable (IE) register

- All interrupt are disabled after reset
- We can enable and disable them by IE

D7 D0

EA	--	ET2	ES	ET1	EX1	ET0	EX0

EA	IE.7	Enables / disables all interrupts
--	IE.6	No implemented, reserved for future use
ET2	IE.5	Enables or disables timer 2 overflow interrupt
ES	IE.4	Enables or disables the serial port interrupt
ET1	IE.3	Enables or disables timer 2 overflow interrupt
EX1	IE.2	Enables or disables external interrupt 1
ET0	IE.1	Enables or disables timer 0 overflow interrupt
EX0	IE.0	Enables or disables external interrupt

Enabling an interrupt

- by bit operation
- Recommended in the middle of program

```
SETB    EA        SETB   IE.7    ;Enable All
SETB    ET0       SETB   IE.1    ;Enable Timer0 over flow
SETB    ET1       SETB   IE.3    ;Enable Timer1 over flow
SETB    EX0       SETB   IE.0    ;Enable INT0
SETB    EX1       SETB   IE.2    ;Enable INT1
SETB    ES        SETB   IE.4    ;Enable Serial port
```

- by mov instruction
- Recommended in the first of program
  - **MOV IE, #10010110B**

Disabling an interrupt

```
CLRB    EA                                  ;Disable All
CLRB    ET0                                 ; Disable Timer0 overflow
CLRB    ET1                                ; Disable Timer1 overflow
CLRB    EX0                                 ; Disable INT0
CLRB    EX1                                 ; Disable INT1
CLRB    ES                                  ; Disable Serial port
```

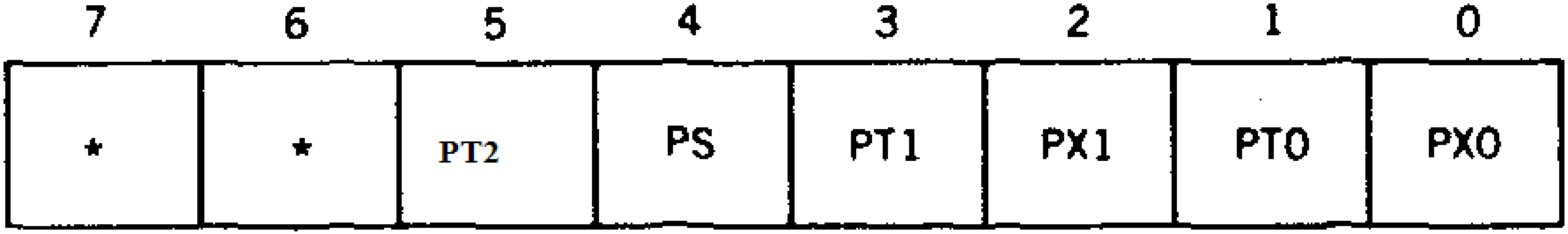
Interrupt Priorities

- What if two interrupt sources interrupt at the same time?
- The interrupt with the highest PRIORITY gets serviced first.
- All interrupts have a power on default priority order.

1. External interrupt 0 (INT0)
2. Timer interrupt0 (TF0)
3. External interrupt 1 (INT1)
4. Timer interrupt1 (TF1)
5. Serial communication (RI+TI)

- Priority can also be set to “high” or “low” by IP reg.

Interrupt Priorities (IP) Register



7	6	5	4	3	2	1	0
*	*	PT2	PS	PT1	PX1	PT0	PX0

IP.7: reserved

IP.6: reserved

IP.5: timer 2 interrupt priority bit(8052 only)

IP.4: serial port interrupt priority bit

IP.3: timer 1 interrupt priority bit

IP.2: external interrupt 1 priority bit

IP.1: timer 0 interrupt priority bit

IP.0: external interrupt 0 priority bit

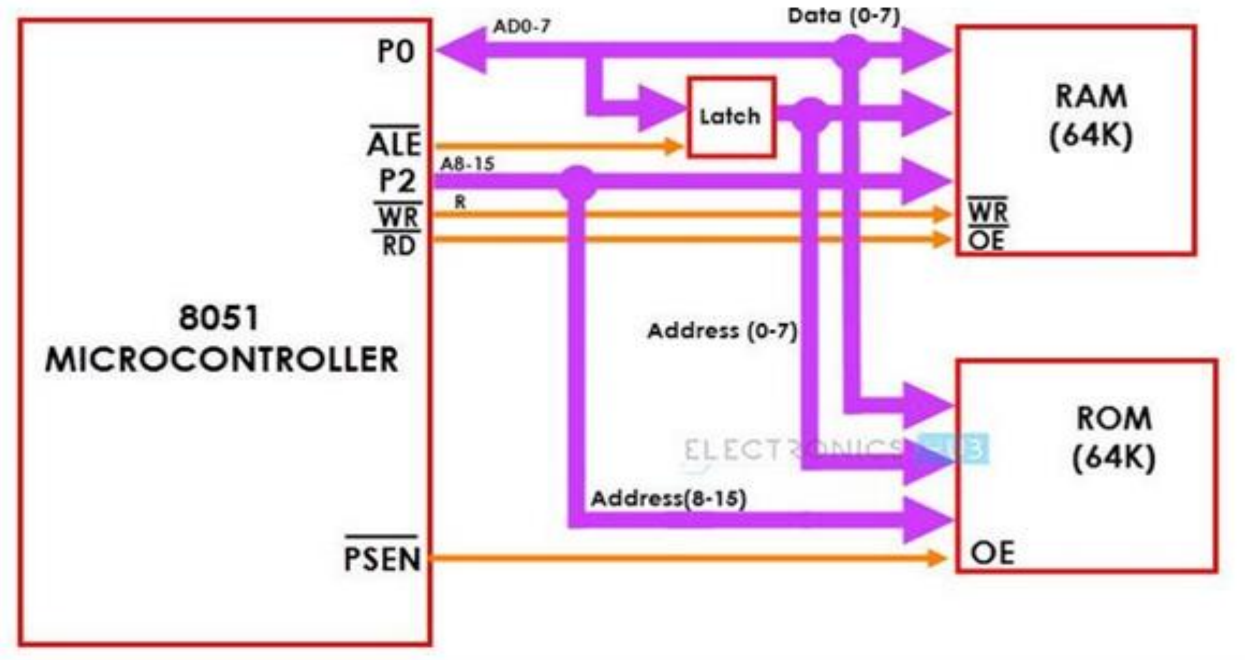
Interrupt Addresses

Interrupt	Address
INT0	0003H
INT1	000BH
T0	0013H
T1	001BH
TI/RI	0023H

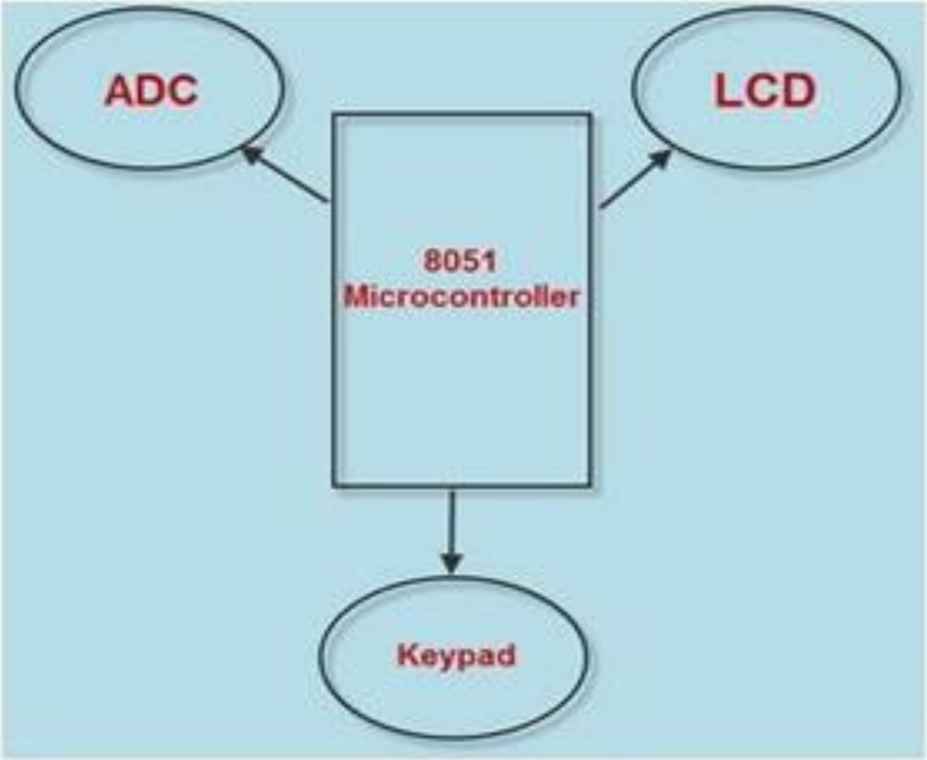
Memory and I/O interfacing with 8051

Block Diagram

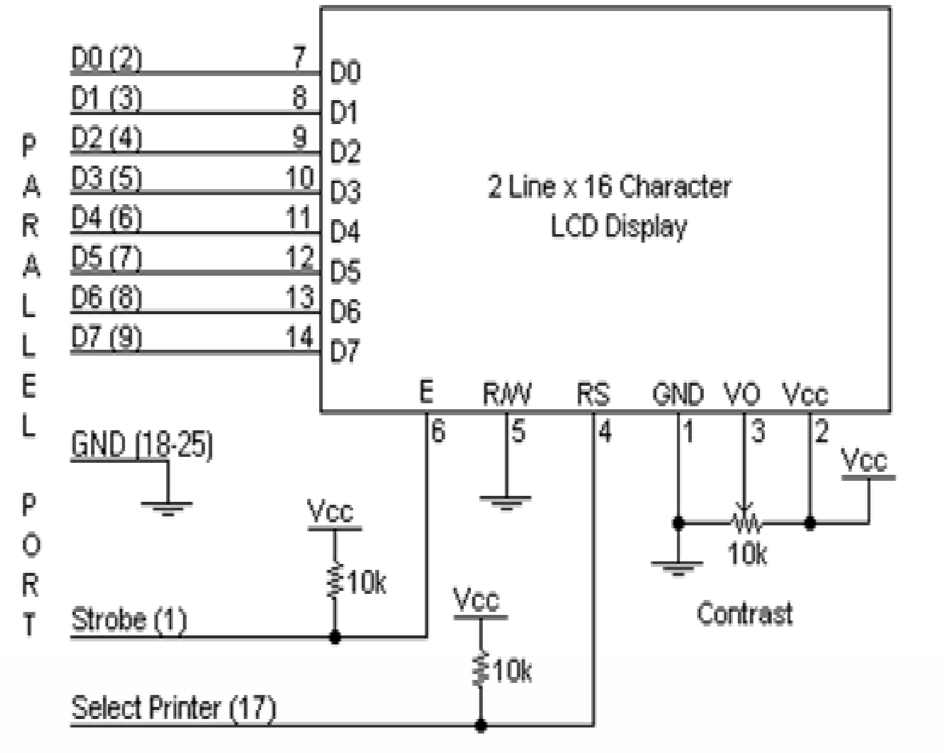
interfacing 64KB of External RAM and 64KB of External ROM with the 8051 Microcontroller.

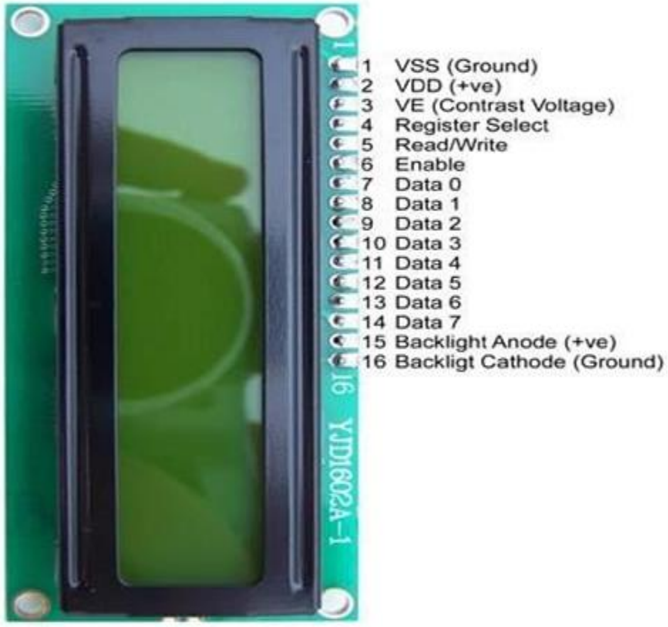


Interfacing I/O Devices

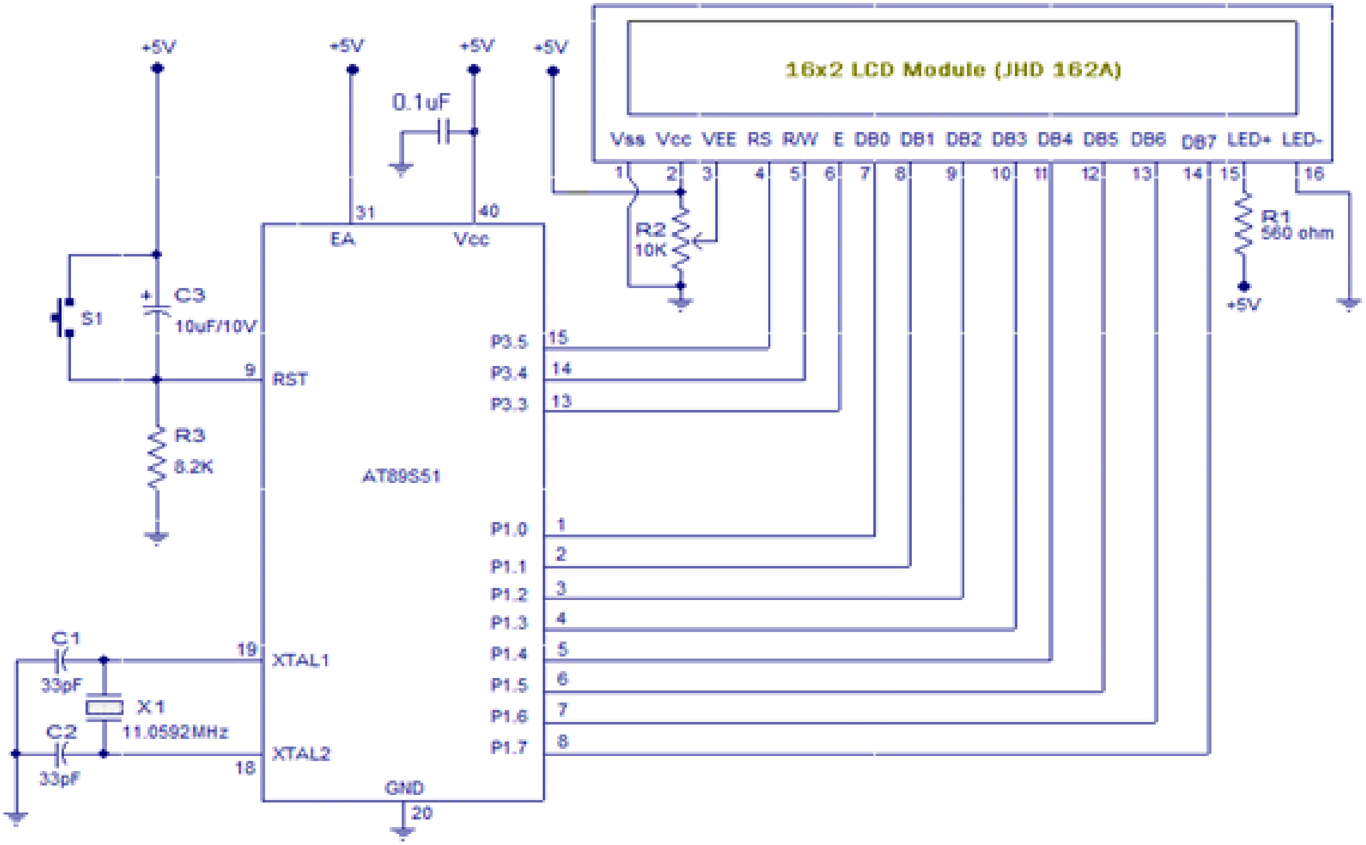


Intelligent LCD Display





Interface Intelligent LCD Circuit With 8051



16x2 LCD Module (JHD 162A)
Vss
Vcc
VEE
RS
R/W
E
DB0
DB1
DB2
DB3
DB4
DB5
DB6
DB7
LED+
LED-
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
+5V
+5V
+5V
+5V
0.1uF
31
40
EA
Vcc
R2
10K
R1
560 ohm
+5V
C3
10uF/10V
S1
9
RST
P3.5
15
P3.4
14
P3.3
13
R3
8.2K
AT89S51
P1.0
1
P1.1
2
P1.2
3
P1.3
4
P1.4
5
P1.5
6
P1.6
7
P1.7
8
C1
33pF
X1
11.0592MHz
C2
33pF
19
XTAL1
18
XTAL2
GND
20