GENERAL INSTRUMENT CP 1600 => 16- bit Processor

Third generation microprocessors

- After 1978
- 16-bit processors designed using HMOS(High density MOS)
- High packing density and better speed power product
- Provided with 40/48/64 pins package
- Strong processing capability
- Easier to program
- Allowed dynamic relocation with an internal register size of 8/16/32 bits
- Segmented address and virtual memory
- 1-16 MB physical memory space
- Powerful interrupt handling capability
- Examples → INTEL 8086 & 8088

- 4004 → In 1971 first microprocessor was developed by INTEL. It contained approximately 2300 PMOS transistors. It was four bit device used in calculators.
- 8008 → In 1972 INTEL developed a eight but microprocessor. It required 20 or more additional devices to form a functional CPU.
- 8080 → In 1974 INTEL developed 8080 which had much larger instruction set than 8008 and require only two additional devices to form a functional CPU. Also 8080 used NMOS transistors. So it can operate much faster than 8008. It requires +5V, -5V, and +12V supplies.
- MC6800 → To avoid difficulties in 8080 which require different power supplies. Motorola came out with MC6800 which requires only +5V supply. For several years 8080 and 6800 were top selling 8-bit processors.
- 8085 → It is a 8-bit processor. INTEL produced 8085, an upgrade of 8080, requires only +5V supply.
- MC6809 → Motorola then produced MC6809 which has few 16-bit instructions but still basically 8-bit processor.
- 8086 → In 1978 INTEL developed 8086 which is full 16-bit processor, it contains approximately 29,000 transistors and is fabricated using HMOS technology.

UNIT-1

8086 MICROPROCESSOR

-Typical Microprocessor based system

A typical microprocessor based system consists of

- CPU (central Processing Unit)

(ALU + Register organization + Control unit)

- Timing unit
- Bus control logic
- Memory
- I/O subsystem

Features of 8086

- Introduced in 1978 .

- Comes in Dual-In-Line Package(DIP) IC.
- 8086 is a 16-bit N-channel HMOS microprocessor .
- Works on 5 volts power supply and draws a current of 360 ma, with an internal circuitry made up of 29K transistors.
- It consists of an electronic circuitry built using 29000 transistors.
- It is built on single semiconductor chip and packaged in an 40-pin IC.
- It has 20-bit address bus and 16-bit data bus.
- It can directly address upto 2^{20} I.e., 1M bytes of memory.
- The 16-bit data word is divided into lower-order byte and

  higher order byte.
- The 20-bit address bus is time multiplexed:

➢ The lower order 16-bit address bus is time multiplexed with data bus.

➢ The higher order 4-bit address bus is time multiplexed with status signals.

- The maximum internal clock for 8086 is 5MHz.
- 8086 chip does not have the facility of internal clock generation.

(the INTEL 8284 clock generator driver is used to generate the clock signal for 8086 microprocessor

- The clock signal is divided by 3 in case of 8086 for internal clock requirements.
- 8086 uses I/O mapped I/O techniques hence I/O devices are accessed by using separate 16-bit address
- 8086 operates in two different modes

❖ Minimum mode

(It works as a simple single processor system when configured in minimum mode)

❖ Maximum mode

(It works as a multiprocessor system i.e., along with math coprocessor and I/O coprocessor when configured in maximum mode)

8086 Registers

8086 – Default 16 bit segment and offset address combinations

Segment	offset	special purpose
CS	IP	Instruction Address
SS	SP (or) BP	Stack address
DS	BX,DI,SI an 8-bit number 16 – bit number	Data address
ES	DI for string Instructions	String destination address

SS and SP Reg.

A stack is a section of memory set aside to store addresses and data while a subprogram executes.

The stack segment register is used to hold the upper 16 bits of the starting address for the program stack.

- 2 Logical Addresses for each Segments.
  - Base Address (16 bits)
  - Offset Address (16 bits)
- Segment registers are used to store the Base address of the segment.

Convention Example: EA = CS:[IP]

Segment number in Segment register

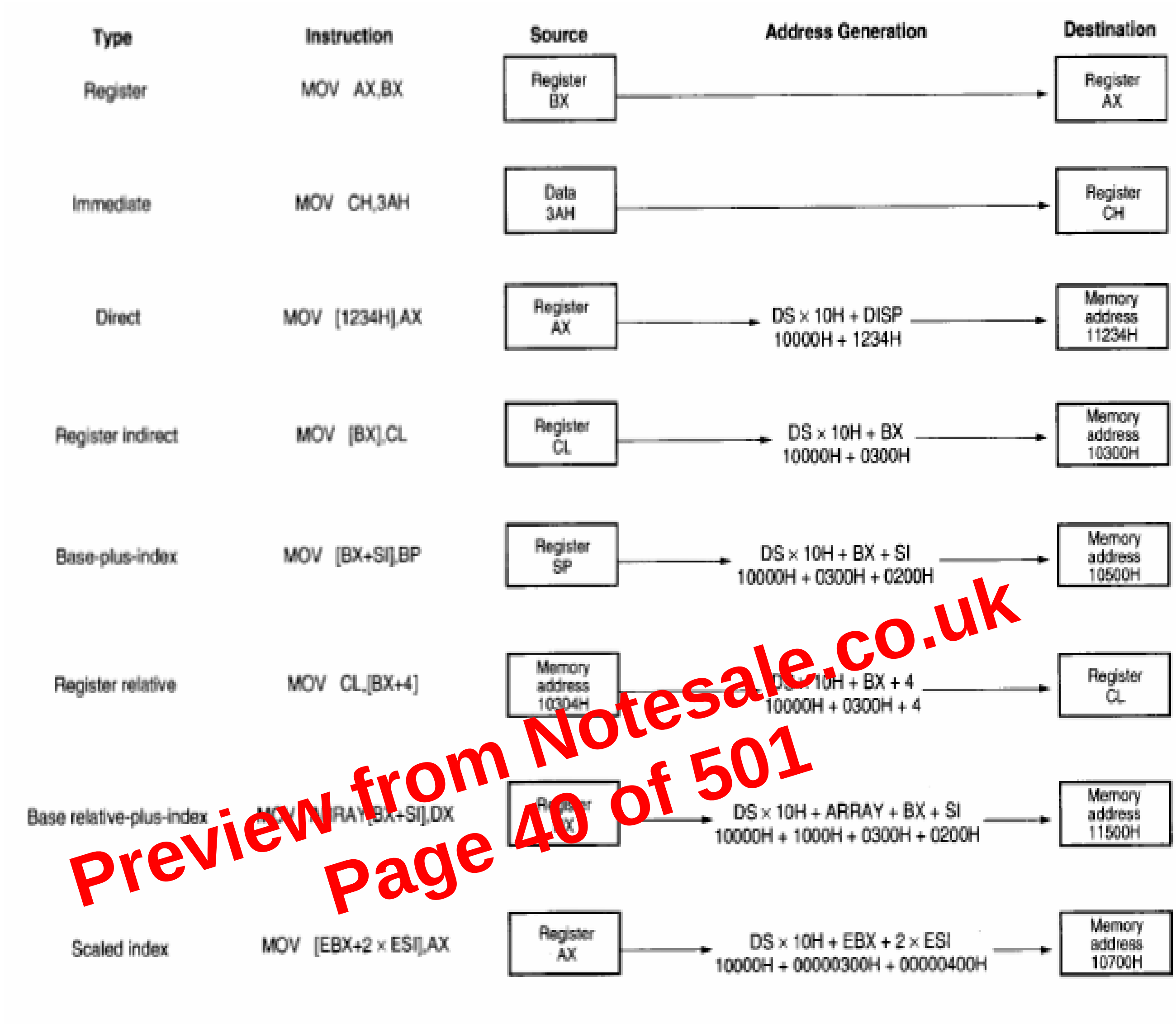
Offset: Literal or in a CPU register

- Default segment numbers in:
  - CS for program (code)
  - SS for stack
  - DS for data
  - ES for string (destination) data
- Default offset addresses that go with them:

Segment	Offset (16-bit) 8080, 8086, 80286	Offset (32-bit) 80386 and above	Purpose
CS	IP	EIP	Program
SS	SP, BP	ESP, EBP	Stack
DS	BX, DI, SI, 8-bit or 16-bit #	EBX, EDI, ESI, EAX ECX, EDX, 8-bit or 32-bit #	Data
ES	DI, with string instructions	EDI, with string instructions	String destination

Stack Pointer Register

A Stack, is a section of memory set aside to store addresses and data while asubprogram is being executed. An entire 64 K bytes segment is set aside as Stack in 8086MPU. The upper 16 bits of the starting address for this segment is kept in the stack segment register. The Stack Pointer (SP) register contain the 16-bit offset from the start of the segment to the memory location where a word was most recently stored on the Stack.The memory location where a word was most recently stored is called the top of Stack .Fig.6 shows the details.The physical address for a stack read or for a stack write is produced by adding thecontents of the stack pointer register to the segment base address in SS. To do this thecontents of the Stack segment register are shifted four bit positions left and the contentsof SP are added to the shifted result. In the figure 5000 H in SS is shifted



Notes: EBX = 00000300H, ESI = 00000200H, ARRAY = 1000H, and DS = 1000H

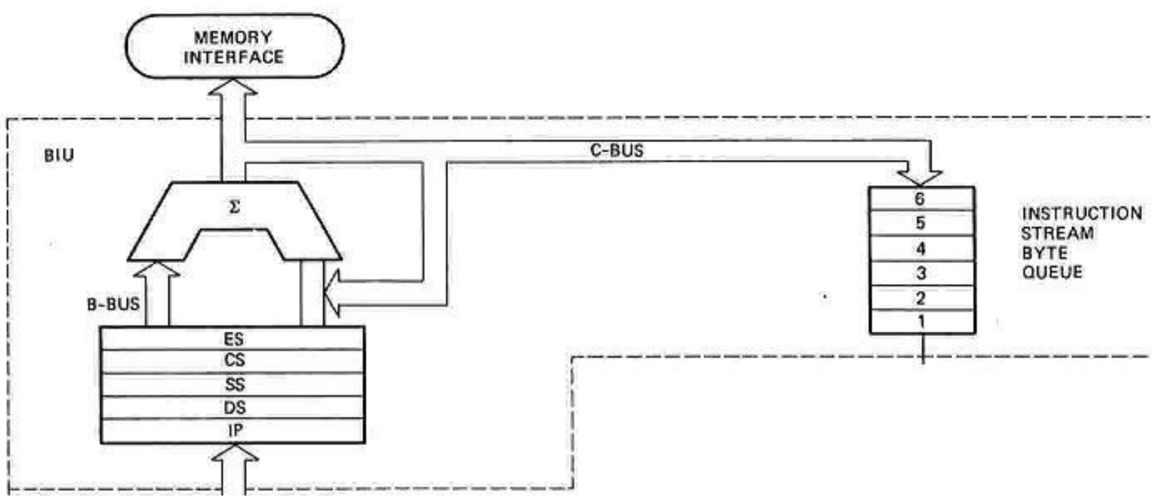
FIGURE 3–2 8086–Pentium 4 data-addressing modes.

- The BIU performs all bus operations such as instruction fetching, reading and writing operands for memory and calculating the addresses of the memory operands. The instruction bytes are transferred to the instruction queue.
- EU executes instructions from the instruction system byte queue.
- Both units operate asynchronously to give the 8086 an overlapping instruction fetch and execution mechanism which is called as Pipelining. This results in efficient use of the system bus and system performance.
- BIU contains Instruction queue, Segment registers, Instruction pointer, Address adder.
- EU contains Control circuitry, Instruction decoder, ALU, Pointer and Index register, Flag register.

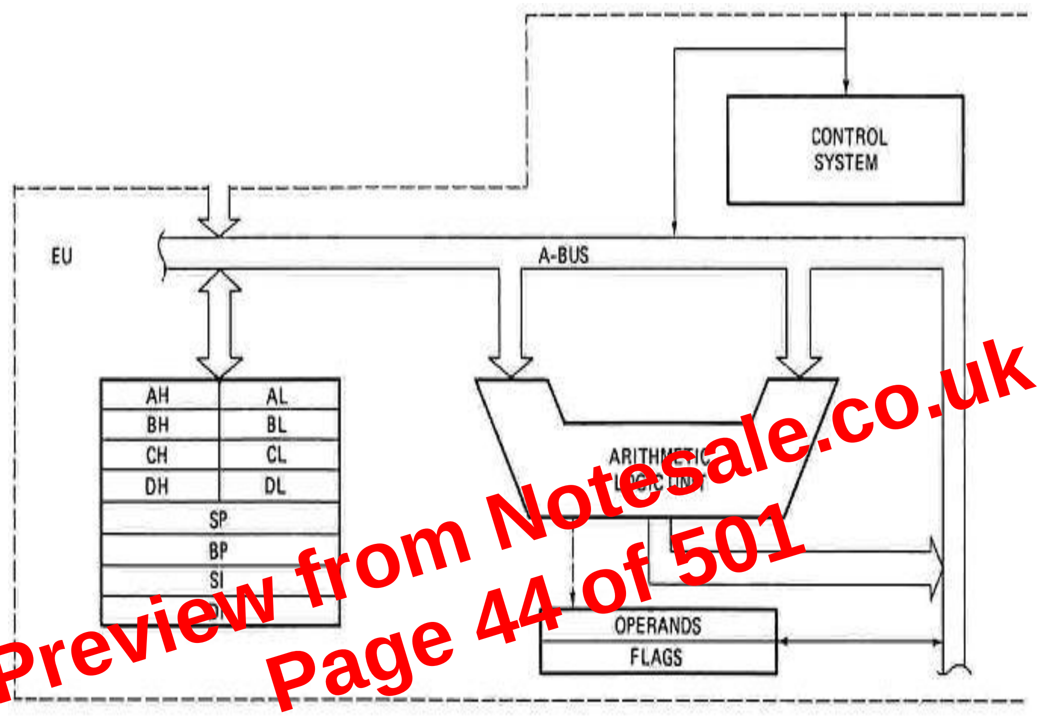
BUS INTERFACE UNIT (BIU)

Contains

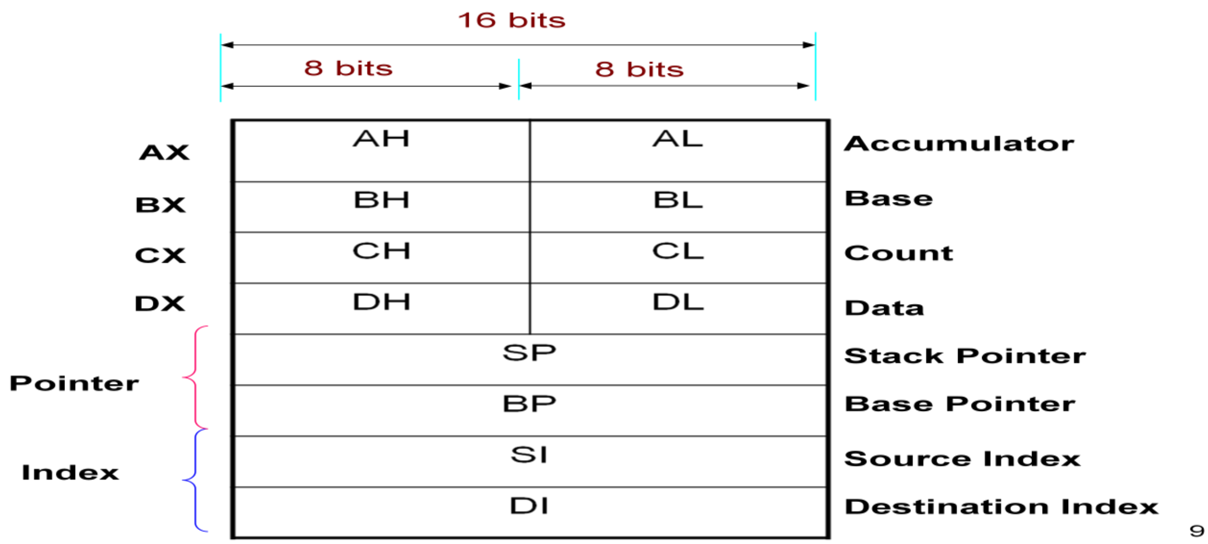
- 6-byte Instruction Queue (Q)
- The Segment Registers (CS, DS, ES, SS).
- The Instruction Pointer (IP).
- The Address Summing Block (Σ)



- **Instruction decoder**
- **ALU**

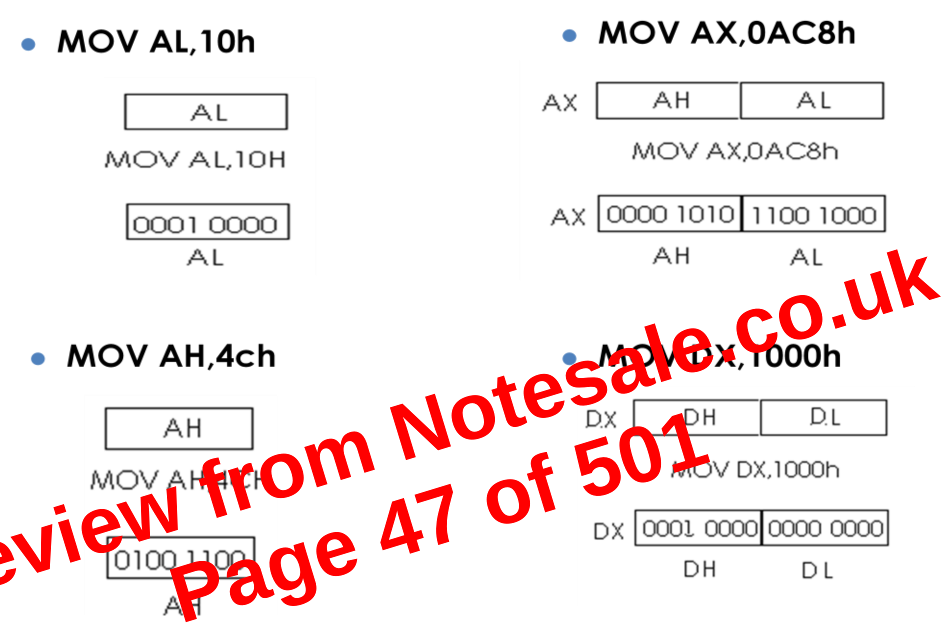


EXECUTION UNIT – General Purpose Registers

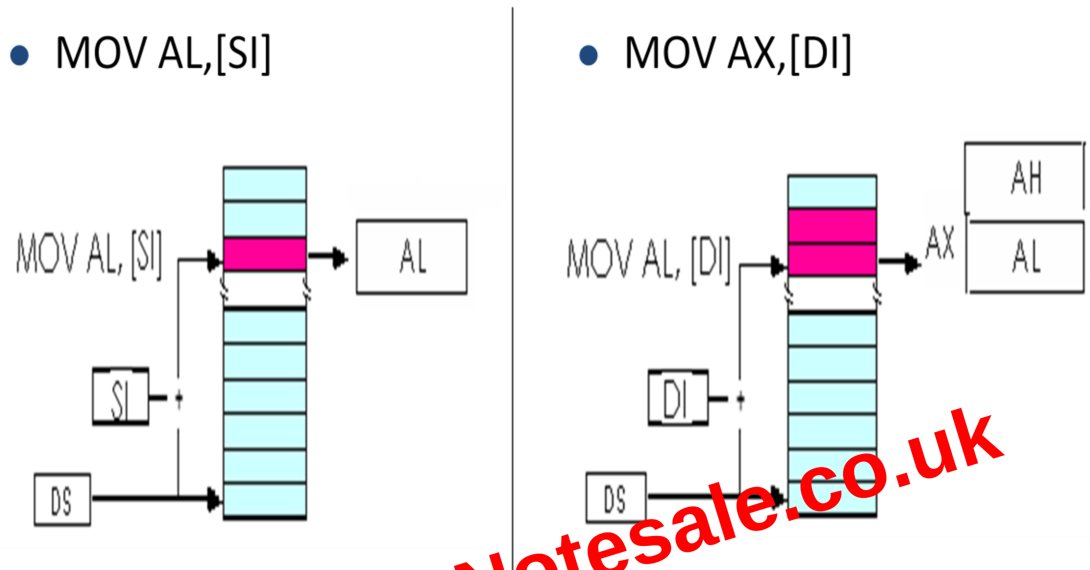


- **<u>Register Addressing Mode.</u>**
- **<u>Direct addressing Mode.</u>**
- **<u>Register Indirect Addressing Mode.</u>**

Immediate Addressing Mode



Indexed Addressing Mode



Based Indexed Addressing Modes

The based indexed addressing modes are simply combinations of the register indirect addressing modes. These addressing modes form the offset by adding together a base register (bx or bp) and an index register (si or di). The allowable forms for these addressing modes are

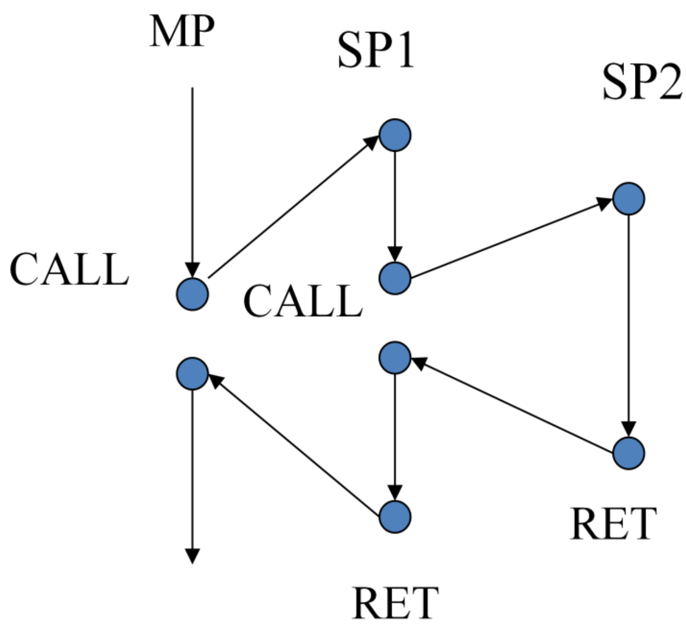
```
mov     al, [bx][si]
mov     al, [bx][di]
mov     al, [bp][si]
mov     al, [bp][di]
```

Suppose that bx contains 1000h and si contains 880h. Then the instruction

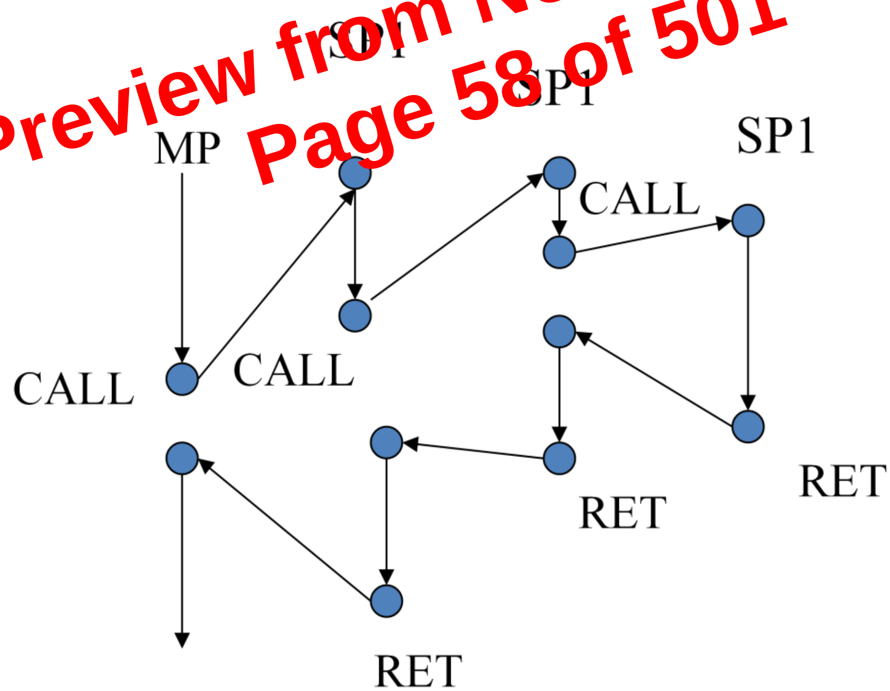
```
mov       al,[bx][si]
```

would load al from location DS:1880h. Likewise, if bp contains 1598h and di contains 1004, mov ax,[bp+di] will load the 16 bits in ax from locations SS:259C and SS:259D.

The addressing modes that do not involve bp use the data segment by default. Those that have bp as an operand use the stack segment by default.



- **Re-Cursive Procedure:**



- **The recursive procedures are implemented using procedure CALL itself, but care must be taken to assure that each successive call does not destroy the parameters and results**

```
Code SEGMENT
ASSUME CS : Code, DS : Data
START : Mov AX, Data
        Mov DS, AX
          :
        Mov AX,Num
        CALL X1
        INT 3H
    X1 PROC Near
          MOV BX,Num
              :
          Add AX, BX
              :
          RET
    X1 ENDP
          Code ENDS
          END START
```

➢ **Using (General purpose CPU Registers) Registers:**

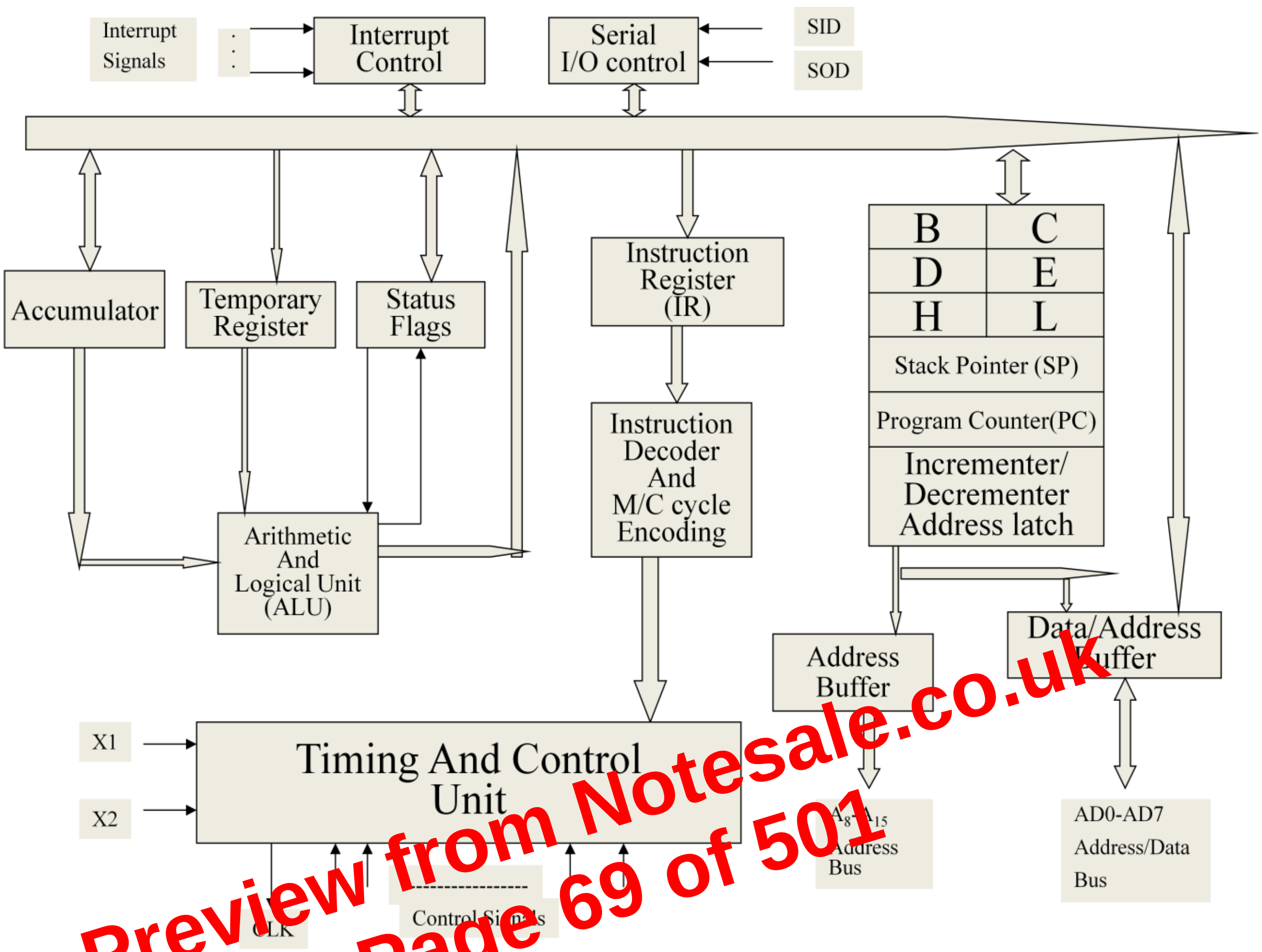
```
Code SEGMENT
ASSUME CS : Code
START : Mov AX, 2234H
        Mov BX, 3342H
          :
        CALL X1
          :
         INT 3H
```

```
X1 ENDP
    Code ENDS
    END START
```

- **Passing parameters using pointers passed in registers:**

```
        :
    Mov SI, OFFSET Str1
    Mov DI, OFFSET Str2
        :
    CALL X1
        :
    INT 3H
X1 PROC Near
    Mov AL, (SI)
    Mov BL, AL
    Mov CL, (DI)
        :
    RET
X1 ENDP
    Code ENDS
    END START
```


- **Features of Passing parameters using global variables:**
  - **Extra memory required.**
  - **All the globally declared variables need to be remembered.**
  - **Modification of parameters cannot be done directly.**
  - **Can be implemented only by using ASSEMBLER and not by any other means.**
- **Features of Passing parameters using CPU Registers:**

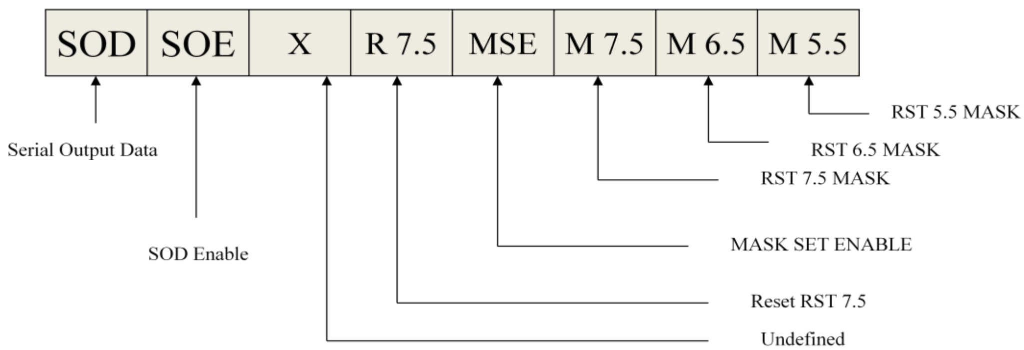


- **ALU**
- **The arithmetic and logical unit(ALU), performs the following arithmetic and logical operations:**

- **Addition**
- **Subtraction**
- **Logical AND**
- **Logical OR**
- **Logical Exclusive-OR**
- **Complement (Logical NOT)**
- **Increment (add 1)**
- **Decrement(Subtract 1)**

Functional unit of 8085

- Accumulator contents for a SIM instruction:



- **The other functional blocks other than the ALU and other registers is as follows:**

  1. Internal Data Bus

  2. Serial I/O Control

  3. Interrupt Control

  4.Timing and Control

  5. Address Buffer and Address / Data Buffer

➢ **Internal Data Bus:**

The internal data bus is 8-bits inside and carries instructions and data between the CPU registers.

- **Functional unit of 8085**
- **Serial I/O Control :**

Generally the Data flowing between microprocessor will be either parallel or serial, but for some devices it is necessary to accept data serially and output data serially and if there is a provision built in in the microprocessor for this purpose it is very efficient.

In 8085 there is such provision through SID and SOD pins.

The SID pin is used for accepting serial data input.

The contents of stack pointer or program counter can be loaded into these buffers. These buffers drive the external address bus and address-data bus. The internal data bus is also connected to the address / data buffer to send or receive the data.

- **Data and Address Bus**
- **The INTEL 8085 is an 8-bit microprocessor. Its data bus is 8-bit wide and hence, 8 bits of data can be transmitted in parallel from or to the microprocessor.**
- **The INTEL 8085 requires a 16-bit wide address bus as the memory address are of 16-bits.**
- **The 8 most significant bits of the address are transmitted by the address bus, A-bus (pins A_8 – A_{15}).**
- **The 8 least significant bits of the address are transmitted by Address / Data bus, AD- bus (pins AD_0 – AD_7).**
- **The Address / Data bus transmits data and address at different moments. At a particular moment it transmits either data or address. Thus the AD – bus operates in time shared mode. This technique is known as Time Multiplexing .**
- **Pin Configuration Of 8085**
- **The logical pin out of an 8085 microprocessor consists of an 40-pin DIP package.**
- **The pins of the 8085 microprocessor can be categorized int the following groups:**
    - ➢ **Address bus**
    - ➢ **Data bus**
    - ➢ **Control and Status signals**
    - ➢ **Power Supply and Frequency signals**
    - ➢ **Externally initiated signals**
    - ➢ **Serial I/O ports**

- **Control and Status signals:**

This group of signals includes two control signals ($\overline{RD}$ and $\overline{WR}$), three status signals ($IO/\overline{M}$, S_1 and S_0) to identify the nature of the operation, and one special signal (ALE) to indicate the beginning of the operation. These signals are as follows:

- **ALE – Address Latch Enable:**

Output and Tri-stated line

It is a Address Latch Enable signal. It goes high during the first clock cycle of a machine cycle and enables the lower 8 bits of the address to be latched either into the memory or external latch.

- **$\overline{RD}$: Read**

Output and a tri-stated line

It is a signal to control read operation

When it goes low the microprocessor reads the data from the selected memory location or an I/O device.

- **$\overline{WR}$: Write**

Output and Tri-stated line

It is a signal to control Write operation

When it goes low the microprocessor writes the data into the selected memory or I/O device.

- **$IO/\overline{M}$: I/O or Memory indicator**

Output and Tri-stated line.

It is a status signal which distinguishes whether the Address is for Memory or I/O.

When it goes high the address is for an I/O device and when it goes low the address on the address bus is for a memory location.

- **S_1 and S_0 : Bus state/ status indicator**

Output lines.

The status output signals from microprocessor and these signals gives the information about the various types of operations that take place.

.

S_1	S_0	Operations
0	0	HALT
0	1	WRITE
1	0	READ
1	1	FETCH

- **Power Supply and Clock Frequency Signals**

The power supply and frequency signals are as follows:

- **VCC : + 5 V Power Supply**
- **VSS : Ground Reference**
- **X_1 and X_2 : Crystal or RC Connections**

Input lines

These are terminals to be connected to an external crystal oscillator which drives an internal circuitry of the microprocessor to produce a suitable clock for the operation of microprocessor.

- **CLK(Out) : Clock signal**

Output line

It is a Clock Output for user,which can be used for other digital IC's. Its frequency is same at which processor operates.

- **Externally Initiated Signals including Interrupts:**

There are five Interrupt signals associated with 8085 microprocessor along with an acknowledge signal for these interrupts.

apart from the interrupt signals there are the following externally initiated signals associated with 8085 microprocessor pin configuration they are

- **HLDA : Output**

It is a signal for HOLD acknowledgement. It indicates that the HOLD request has been received.

The HLDA goes low after the HOLD signal goes low.

The CPU takes over the buses after Half clock cycle of removal of HLDA signal.

- **READY : Input**

It is used by the microprocessor to sense whether a peripheral is ready to transfer data or not.

A slow peripheral may be connected to the microprocessor through the Ready line.

If READY is high the peripheral is ready.If it is low the microprocessor waits till it goes high.

- **RESET OUT : Output**

Indicates that CPU is being reseted, and is used by the microprocessor to reset the other sub-systems in the microprocessor based system.

- **(RESET IN)’ : Input**

It resets the program counter to zero. It also resets interrupt enable and HLDA flip-flops. It does not affect any other flag or register except the instruction register.

The CPU is held in the reset condition as long as RESET is applied.

- **Serial I/O ports**

The 8085 microprocessor has two signals for serial communication i.e., SID and SOD

  - **SID: Serial Input Data (Input)**

It is data line for serial input. The data on this line is loaded into the 7th bit of the accumulator when RIM instruction is executed.

  - **SOD : Serial Output Data (Output)**

It is a data line for serial output. The 7th bit of the accumulator is output on SOD line when SIM instruction is executed.

Simple programs on 8086 :

```
        mov cx,0
        mov dx,0
        mov ax,num1 ;x1 * y1
        mul num2
        mov res,ax
        mov bx,dx
        mov ax,num1 ;x1 * y2
        mul num2+2
        add bx,ax
        mov cx,dx
        mov ax,num1+2 ;x2 * y1
        mul num2
        add bx,ax
        mov res+2,bx
        add cx,dx
        mov ax,num1+2 ;x2 * y2
        mul num2+2
        add cx,ax
        mov res+4,cx
        mov res+6,dx
        mov ax,4c00h
        int 21h
code ends
```

Or

If READY pin is high, the peripheral is ready otherwise the 8086 enters wait state.

This state is used by slow peripheral devices. The peripheral devices can transfer the data to or from the microprocessor by using READY input line. The microprocessor remains in wait state as long as READY line is low. During the wait state, the contents of the address, address/data and control buses are held constant.

10. 8086 has ____20____ address pins and _16____ data pins
11. Name different segments in 8086 microprocessor
    CS,SS,DS,ES
12. Name the index registers in 8086 microprocessor
    Source index, destination index
13. Each register in 8086 is of ___16_____ bit wide
14. what is the purpose of Bus Interface unit
15. what is the purpose of execution unit in 8086 microprocessor

ANS: Write short note on the Execution Unit (EU) and the Bus Interface Unit (BIU).

8086 microprocessor has two units; Execution Unit (EU) and Bus Interface Unit (BIU). They are dependent and get worked by each other. Below is a short description of these two units.

Execution Unit (EU) : Execution unit receives program instruction codes and data from the BIU, executes them and stores

the results in the general registers. It can also store the data in a memory location or send them to an I/O device by passing the data back to the BIU. This unit, EU, has no connection with the system Buses. It receives and outputs all its data through BIU.

ALU (Arithmetic and Logic Unit) : The EU unit contains a circuit board called the Arithmetic and Logic Unit. The ALU can perform arithmetic, such as, +,-,×,/ and logic such as OR, AND, NOT operations.

Registers : A register is like a memory location where the exception is that these are denoted by name rather than numbers. It has 4 data registers, AX, BX, CX, DX and 2 pointer registers SP, BP and 2 index registers SI, DI and 1 temporary register and 1 status register FLAGS . AX, BX, CX and DX registers has 2 8-bit registers to access the high and low byte data registers. The high byte of AX is called AH and the low byte is AL. Similarly, the high and low bytes of BX, CX, DX are BH and BL, CH and Cl, DH and DL respectively. All the data, pointer, index and status registers are of 16 bits. Else these, the temporary register holds the operands for the ALU and the individual bits of the FLAGS register reflect the result of a computation.

Bus Interface Unit : As the EU has no connection with the system Busses, this job is done by BIU. BIU and EU are connected with an internal bus. BIU connects EU with the memory or I/O circuits. It is responsible for transmitting data, addresses and control signal on the busses.

Registers : BIU has 4 segment busses, CS, DS, SS, ES. These all 4 segment registers holds the addresses of instructions and data in memory. These values are used by the processor to access memory locations. It also contain 1 pointer register IP. IP contains the address of the next instruction to executed by the EU.

Instruction Queue : BIU also contain an instruction queue. When the EU executes instructions, the BIU gets up to 6 bytes of the next instruction and stores them in the instruction queue and this process is called instruction prefetch. This is a process to speed up the processor. Also when the EU needs to be connected with memory or peripherals, BIU suspends instruction prefetch and performs the needed operations.

*****

UNIT-2

Assembly language programming involves all the instructions :

Write ALP and execute the program to

1. Add two 8-bit numbers
2. Add two 16-bit numbers

3. Add two 32-bit numbers
4. Subtract two 8-bit numbers
5. Subtract two 16-bit numbers
6. Subtract two 32-bit numbers
7. Multiply two 8-bit numbers
8. Multiply two 16-bit numbers
9. Perform 8-bit division
10. Perform 16-bit division
11. Find square of a number
12. Find cube of a number
13. Exchange two numbers

Experiment III, IV,V &VI: ARRAY PROGRAMMING **4**

Write ALP and execute the program to

14. Add a given series of numbers
15. Find average of a given series of numbers
16. Display squares of a given series of numbers in memory
17. Display cubes of a given series of numbers in memory
18. Find factorial of a given number
19. Sort a series of given numbers in ascending order
20. Sort a series of given numbers in descending order
21. Find GCD of two given numbers
22. Find LCM of two given numbers
23. Display Fibonacci series

Experiment VII : BCD, DECIMAL,ASCII OPERATIONS **5**

Write ALP and execute the program to

24. Perform one byte BCD addition
25. Perform one byte BCD subtraction
26. Produce packed BCD from two ASCII characters
27. Convert decimal number to binary
28. Convert a binary number to a decimal number

Experiment VIII & IX : STRING MANIPULATION PROGRAMS 6

29. Move a string of data bytes from one location to another
30. Concatenate two strings
31. Reverse a given string
32. Compare two strings
33. Find length of a given string
34. Find whether the given byte is in the string or not
35. Insert an element in a given string

Timers 0 of an 8253 provide the Transmit and receive baud clocks for the USART. (Refer to chapter 5 for a detailed discussion of the Hardware).This timer is initialized by the system firmware to provide proper baud clock based on the settings of the DIP Switch as shown below.

DIP SWITCH

1. **SW3 SW2 SW1 Baud rate**
   OFF OFF ON 9,600*
2. **Memory selection**:
   ESA 86/88-2 has four sockets, labeled U9, U8, U7, U6 for RAM. These sockets are configured for 62256(32X 4) devices. Two of these sockets are populated (providing 64K Bytes of RAM) and two are for user expansion.
   DEVICE DIP SWITCH JUMPER
   SW7 SW6
   27256 ON OFF JP10 – 1-2

3. Register Addressing Mode

   Data is stored in a register and it is referred using the particular register

   Ex: MOV BX,AX

4. Register Indirect Addressing Mode

   The offset address of data is in either BX or SI or DI register

   Default segment is either DS or ES

   EX: MOV AX,[BX]

5. Indexed Addressing Mode

   Offset of the operand is stored in one of Index register

   DS is default segment for index registers SI and DI

   For Strings DS and ES are default segments for SI and DI

   EX: MOV AX, [SI]

6. Register Relative Addressing Mode

   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 registers BX,BP,SI and DI in default segment

   EX: MOV AX,50H[BX]

7. Based Indexed Addressing Register

   Effective address of data is formed by adding content of base register to content of Index register

   Default segment register may be ES or DS

   EX: MOV AX,[BX][SI]

8. Relative Based Indexed

   Effective address is formed by adding an 8 or 16 bit displacement with the sum of contents of any one of base registers (BX or BP) and any one of Index registers in a default segment

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

9. Intrasegment Direct Mode

   Effective address to which the control is to be transferred is given by the sum of 8 or 16 bit displacement and current content of IP

1.4 8-BIT SUBTRACTION

```
.MODEL TINY
.STACK 32H
.CODE
START:
MOV AX,@DATA
MOV DS,AX
MOV AX,00
MOV AL,NUM1
MOV BL,NUM2
SUB AL,BL
MOV RESULT,AL
INT 3

.DATA
NUM1 DB 0FFH
NUM2 DB 0AAH
RESULT DB 00
END START
```


RESULT: 0FFH

0AAH

055H

1.7 8-BIT MULTIPLICATION

```
.MODEL TINY
.STACK 32H
.CODE
START:
MOV AX,@DATA
MOV DS,AX
MOV AX,00
MOV AL,NUM1
MOV BL,NUM2
MUL BL
MOV RESULT,AL
MOV RESULT1,AH
INT 3
.DATA
NUM1 DB 0FFH
NUM2 DB 0AAH
RESULT DB 00
RESULT1 DB 00
END START
```


RESULT: 0FFH

0AAH

A956H

1.9 8-BIT DIVISION (16 Bit by 8 Bit)

```
.MODEL TINY
.STACK 32H
.CODE
START: MOV AX,@DATA
MOV DS,AX
MOV AX,00
MOV DX,00
MOV AX,NUM1
MOV BL,NUM2
DIV BL
MOV QUOTIENT,AL
MOV REMAINDER,AH
INT 3
.DATA
NUM1 DW 0FFH
NUM2 DB 0AAH
QUOTIENT DB 00
REMAINDER DB 00
END START
```

RESULT: 0FFH

0AAH

5501H QUOTIENT: 01H

REMAINDER R: 55H

EX2: RAMP WAVE GENRATION USING DAC

```
;8086 MICROPROCESSOR
;INTERFACING TO 8086 WITH DAC INTERFACE
;PORT: PORTA USED ASOUTPUT

CODE SEGMENT
        ORG 2000H
        ASSUME CS:CODE
        MOV AL,80H        ;INTIALIZE 8255PPI
        MOV DX,0FFE6H     ;PORTA,PORTB,PORTC ARE OUTPUT
        OUT DX,AL         ;0FFE6H CONTAINS CWR OF 8255
        MOV DX,0FFE0H     ;0FFE0H INDICATES PORT
        MOV AL,00H
REPEAT: OUT DX,AL
        CALL DELAY
        INC AL
        JMP REPEAT
        INT 3H
DELAY PROC NEAR
        MOV CX,01FFH
L1:     NOP
        LOOP L1
        RET
DELAY ENDP
CODE ENDS
  END
```


EX3: TRIANGULAR WAVE GENRATION USING DAC

```
;8086 MICROPROCESSOR
;INTERFACING TO 8086 WITH DAC INTERFACE
;PORT: PORTA USED ASOUTPUT
```

```
        OR      AL,BL
       RET
KSCAN      ENDP
DELAY PROC NEAR
        PUSH    CX
        MOV     CX,0000H        ; DELAY ROUTINE
DLY: NOP
       LOOP    DLY
        POP     CX
        RET
 DELAY       ENDP
        END
```

D7							D0
SM0	SM1	SM2	REN	TB8	RB8	TI	RI

- SMO & SM1 (Modes of Operation)
- SM2 controls microprocessor to microprocessor communication
- REN (receive enable)
- TB8 (Transmit bit 8)
- RB8 (Received bit 8)
- TI (transmit flag)
- RI (Received Flag)

MODES OF OPERATION

SM1	SM0	MODE	OPERATION
0	0	0	Shift register, baud rate = f/12
0	1	1	8 BIT UART, baud rate programmable
1	0	2	9 BIT UART, baud rate = f/32, f/64
1	0	3	9 BIT UART, baud rate programmable

- BAUD frequency = 2^{SMOD} * Oscillator frequency/(256-TH1)(12*32)
  - SMOD is bit 7 of PCON register

INTERRUPT STRUCTURE OF 8051

There are 5 INTERRUPTS in 8051

i. Timer Interrupts (TF0 & TF1)
ii. External Interrupts (INT 0 & INT1)
iii. Serial port Interrupt

RESET considered as 6^{th} interrupt

RESPONSE TO INTERRUPT

- Complete current instruction
- Save PC on stack
- Save interrupt status
- Jump to fixed ISR address
- Execute ISR

- TF0
- INT1
- TF1
- Serial Port
-

I/0 INTERFACING

p1 is used as a general purpose port, P0 and P2 are used as data and address bus in case of external memory acess, otherwise can be used as input/output ports. P3 port contains multifunction pins. If we are using serial port, timer input pins ar external interrupt this port can not be used as port. Out of 8 bits of port p1 some can be programmed as inputs and some as outputs for example, higher nibble as output can be connected to leds & lower nibble as input can be connected to 4 switches.

Writing code for 8051

The code can be developed in either assembly language using 8051 instructions or by using a simple ‘c’ program by using cross compilers provided by microcontroller IDEs like KEIL µVision 3. The code will be compiles/assembled linked and then executable file will be converted to HEX file to dump it in Microcontroller. For that we use flash programming software provided my manufacturer i.e. Atmel. Atmel’s FLIP software is used for dumping the code into microcontroller.

Steps for writing the code

1. Open KEIL µVision IDE
2. Go to project menu select ‘new project’, navigate to desired project folder and give project name in the file name window and save
3. **Select device for target** window will open**,** click on Atmel to drop down the menu, select **AT89C51ED2** and press ok. Another window opens asking to add startup files, click no, to not to add startup.a51 file
4. Right click on Target1 in project Window and select **‘options for target Target1’**
   In **Target** select **Xtal(MHz): 11.0592**
   Check box use **on-chip ROM**
   In **output** window check the box **‘Create HEX file’**
5. Go to File menu open new file to open an editor. Create your souce file(s) and use the header file “at89c51xd2.h’ in the source file and save files.
6. Right click on Source group1 and select the option add files to group.
7. After adding source files go to projects-> “build Target” to build source files and create final outputs. It creates <hex file to be downloaded to target device. After successful build.

   Program downloading
   1. Set the slide **swich SW2** to **PROG** position and press **reset** with **SW1** on the kit.
   2. Open **atmel FLIP 2.4.2** tool

3. Go to **device option** select, select the specific device **AT89C51ED2** and press OK
4. Go to **file-> Load hex** file, Navigate to desired hex file of the project
5. Go to **settings option-> rs232,** a window will open make sure that no other application is using com port. Click on **COM select com1, set the baud rate to 115200** and click on **connect**
6. In **operations flow** region, check the options **ERASE, BLANK CHECK, PROGRAM, VERIFY.**
7. In the right most side of the window check the box BLJB abd set the address of BSB,EB,SBV as 00,FF and FC respectively and select option ‘level0’ in device SSB region.
8. After performing above steps click **run button** wait until the **status bar displays finished.**
9. After programming slide **SW2 to RUN** position and **reset SW1** to execute the program.

```
                display();
                P3 = 0xFF;
        }
}  //end of main()
// get_key() function will make columns high one by one
// and calls scan() function
// on sensing a key from scan() function it
// will compare the received scan code with
// scan code lookup table and returns ASCII code
// rows are read from Port P0 is scan() function
// this function is in an eternal loop
// wiil return to main() only after getting a key
void get_key(void)
{
        int i;
                display();
        flag = 0x00;
        while(flag == 0x00)
        {
                for(row=0;row<4;row++)
                {
                    if( row == 0)
                            temp3=0xFE;
                    else if(row == 1)
                            temp3=0xFD;
                    else if(row == 2)
                            temp3=0xFB;
                    else if(row == 3)
                            temp3=0xF7;
// make coulmn high one by one output to Port P1 and
// invoke scan() function

               P0 = temp3;
                            scan();
```

6.3 ADC interfacing

/* This program displays the ADC output of the ADC0809 IC.
Connections: CN2 port1 to CN15 connector and CN1 port0 connector to CN16 of adc block.Also Connect CN3 port2 to CN6 of LCD block. Vary pot R42 to gewt different input voltage values
*/

```
#include<at89c51xd2.h>
#include<stdio.h>

// LCD FUNCTION PROTOTYPE
void lcd_init(void);
void lcd_comm(void);
void lcd_data(void);

void delay(int);

unsigned char temp1;
unsigned char temp2,buf[8];

float adc_temp;
sbit EOC = P0^4;
sbit START_ALE = P0^5;
unsigned char xdata arr1[12]={"ADC O/P = "};
unsigned char xdata arr2[12]={"ADC I/P = "};
unsigned char i,a,temp_hi,temp_low;
unsigned int vtemp1,adc_val;
unsigned char temp_msg[]={"     "};

void main ()
{
        START_ALE = 0;

        lcd_init();

        temp1 = 0x80;                    // Display the data from first
position of first line
        lcd_comm();                         // Command Writing

        for(i=0;i<10;i++)
        {
          temp2 = arr1[i];
```

```
        {
         P0 = Val;                                  //* Write data for clock
wise direction
         Val = Val>>1;
         delay(575);
        }
       }
       else                                   // AntiClockwise Direction
       {

        Val = 0x01;
        for(i=0;i<4;i++)
        {
         P0 = Val;                                  // Write data for anticlock wise
direction
         Val = Val<<1;
         delay(575);
        }
       }
 }
}
```

Format: XCHG < destination > , < source >

Operation: (destination) ⇔ (Source)

Examples: XCHG Reg1, Reg2

XCHG Mem, Reg

XCHG AX, Reg16

General purpose byte or word transfer instructions

- XLAT: Translate a byte in AL, using a table in memory

Format: XLAT

Operation: $PA = DS \times 16_{10} + (BX) + (AL)$

$(AL) \leftarrow (PA)$

This instruction is used to translate a byte from one code to another code

The instruction replaces a byte in the AL register with a byte pointed to by BX register in a look up table in memory

Before executing XLAT instruction, the look up table is to be put into memory and the starting address of the look up table has to be loaded into BX register

Examples: ASCII value of 0-9 is 30-39 and EBCDIC is 0-9. Hence to convert EBCDIC code in ASCII, the ASCII values of the 0-9 has to be stored say from 2000H, then save 2000H in BX.

Simple input and output port transfer instructions

IN & OUT

- IN : Copy a byte or word from specified port to accumulator.

Format : IN <Accumulator>, <Source>

Example : IN AL/AX,[DX]

(AL/AX) ← (Port)

the contents of 8-bit port whose address is specified by DX register is transferred to 8-bit accumulator (AL/AX)

IN AL/AX, addr8

If the value in the lower nibble is greater than 9 then the AL is incremented by 06,AH is incremented by 1,the AF and CF flags are set to 1, and the higher nibble of AL is cleared to 0.

- Example: 1) AL = 67 (before AAA)

  AL = 07(after AAA)

  2) AL = 6A; AH = 00 (before AAA)

  A > 9 ,hence A + 6 = 1010 + 0110

  = 00010000 = 10H & AF = 1

Thus before AAA instruction AX = 006AH

- DAA : Decimal adjust Accumulator

This instruction is used to convert the result of the addition of two packed numbers to a valid BCD number,but the result has to be only in AL.

If the lower nibble is greater than 9, after addition or if AF is set, it will add 06H to the lower nibble in AL. After adding 06 in the lower nibble of AL is greater than 9 or if carry flag is set, DAA instruction adds 60H to AL.

The DAA instruction affects AF,CF,PF and ZF flags. The OF is undefined.

Example: AL = 53, CL = 29

ADD AL,CL ; AL <- (AL) + (CL)

AL = 53 + 29 = 7C H

After DAA AL <- 7C + 06 H

AL = 82

- Example:

  AL = 73 and CL = 29

ADD AL,CL

AL <- AL + CL

AL <- 73 + 29

Multiplication instructions:

MUL, IMUL, AAM

- o MUL : Unsigned Multiplication of byte or word.

MUL Reg. / Mem.

This instruction multiplies an unsigned byte or word by the contents of AL.

The Unsigned byte or word may be in any one of the general purpose registers or memory locations.

For Byte multiplication the most significant byte will be stored in AH register and least significant byte is stored in AL register.

For Word multiplication the most significant word of the result is stored in DX, while the least significant word of the result is stored in AX register

All the flags are modified depending upon the result of the operation.

Immediate operand is not allowed in this instruction.

If the most significant byte or word of result is '0' CF and OF both will be set.

Example: MUL BL

MUL BX

- ➢ IMUL : Signed Multiplication

This instruction multiplies a signed byte in source operand by a signed byte in AL register or A signed word in AX register.

The source can be a general purpose register, memory operand,index register or base register, but it cannot be an immediate data.

While using this instruction the content of accumulator and register should be sign extended binary in 2's complement form and the result is also in sign extended binary.

In case of 32-bit results, the higher order word(MSW) is stored in DX and lower order word is stored in AX

In case of 16-bit result it will be stored in AX register.

The AF, PF, SF, and ZF flags are undefined after IMUL instruction execution.

If AH and DX contains parts of 16-bit and 32-bit result respectively, CF and OF both will be set.

The AL and AX are the implicit operands in case of 8 –bits and 16-bits multiplication respectively.

The unsigned higher bits of the result are filled by sign bit and CF,AF are cleared.

Example: IMUL BL / IMUL BX

- AAM : ASCII Adjust after Multiplication

This instruction after execution, converts the product available in AL into unpacked BCD format.

The AAM instruction follows a multiplication instruction that multiplies two unpacked BCD operands, i.e., higher nibbles of the multiplication operands should be ‘0’.The multiplication of such operands is carried out using MUL instruction.

The result of the multiplication will be available in AX.

(AH) = (AL) ÷ $0A_H$

(AL) = (AL) MOD $0A_H$

The AAM instruction replaces the contents of AH by tens of decimal multiplication and AL by singles of the decimal multiplication.

MOV AL, 04 ; AL ← 04

MOV BL, 09 ; BL ← 09

MUL BL ; AH:AL ← 24H (9 X 4)

AAM ; AH ← 03

; AL ←06

- ARITHMETIC INSTRUCTIONS
  Division instructions

Division instructions: DIV, IDIV, AAD, CBW, CWD

- DIV : Unsigned division

DIV <reg./Mem>

This instruction performs unsigned division. It divides an unsigned word or double word by a 16-bit or 8-bit operand.

The dividend must be in AX for 16-bit operation and the divisor may be specified using any one of the addressing modes except immediate.

The dividend for 32-bit operation will be in DX:AX register pair (Most significant word in DX and least significant word in AX).

All the flags are undefined for DIV instruction.

The result of division is for 16-bit number divided by 8-bit number the Quotient will be in AL register and the remainder will be in AH register similarly for 32-bit number divided by 16-bit number the Quotient will be in AX register and the remainder will be in DX register.

If the result is too big to fit into AL or AX register then Type-0 (divide by zero) interrupt is generated and the ISR for the Type zero will be executed such that correction steps are taken to accommodate the result.

- <u>For 16-bit ÷ 8-bit</u>

  (AL) ← (AX) ÷ (reg.-8) ; Quotient

  reg.-8 : 8 – bit register

  (AH) ← (AX) Mod (reg.-8) ; Remainder

- <u>For 32-bit ÷ 16-bit</u>

  (AX) ← (DX)(AX) ÷ (reg.-16) ; Quotient

  (DX) ← (DX)(AX) Mod (reg.-16) ; Remainder

  Reg.-16 : 16 – bit register

  Example: DIV AX/ DIV [BX]

- IDIV : signed division

  IDIV <reg./Mem>

This instruction performs signed division. It divides an signed word or double word by a signed 16-bit or 8-bit operand.

While using IDIV instruction the contents of accumulator and register should be sign extended binary.

The signed dividend must be in AX for 16-bit operation and the signed divisor may be specified using any one of the addressing modes except immediate.

The signed dividend for 32-bit operation will be in DX:AX register pair (Most significant word in DX and least significant word in AX).

All the flags are undefined for IDIV instruction.

SHL / SAL, SHR, SAR

SHL / SAL : Shift Logical / Arithmetic Left

SHL <reg. / Mem>

CF ← R(MSB) ; R(n+1) ← R(n) ; R(LSD) ← 0

These instructions shift the operand word or byte bit by bit to the left and insert zeros in the newly introduced least significant bits.

The number of bits to be shifted if 1 will be specified in the instruction itself if the count is more than 1 then the count will be in CL register.

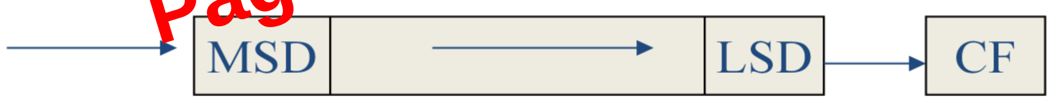
The operand to be shifted can be either register or memory location contents but cannot be immediate data.

All the flags are affected depending upon the result.The shift operation will considering using carry flag.

SHR : Shift Logical Right

SHR <reg. / Mem>

CF ← R(LSB) ; R(n) ← R(n+1) ; R(MSD) ← 0



These instructions shift the operand word or byte bit by bit to the right and insert zeros in the newly introduced Most significant bits.

The result of the shift operation will be stored in the register itself.

The number of bits to be shifted if 1 will be specified in the instruction itself if the count is more than 1 then the count will be in CL register.

The operand to be shifted can be either register or memory location contents but cannot be immediate data.

All the flags are affected depending upon the result.The shift operation will considering using carry flag.

SAR : Shift Logical Right

Compare one byte or word of a string data stored in data segment with that stored in extra segment.

The SI register points to the source string and DI register points to the destination string.

The CX register is decremented by one for each byte / word movement.

The SI and DI registers are automatically incremented or decremented depending on the status of DF.

MA = (DS) X 16_{10} + (SI)

MA_E = (ES) X 16_{10} + (DI)

Modify flags ← (MA) - (MA_E)

If (MA) > (MA_E) then CF = 0 ; ZF = 0 ; SF = 0

If (MA) < (MA_E) then CF = 1 ; ZF = 0 ; SF = 1

If (MA) = (MA_E) then CF = 0 ; ZF = 1 ; SF = 0

- For byte operation

If DF = 0, then (DI) ← (DI) + 1 ; (SI) ← (SI) + 1

If DF = 1, then (DI) ← (DI) - 1 ; (SI) ← (SI) - 1

- For word operation

If DF = 0, then (DI) ← (DI) + 2 ; (SI) ← (SI) + 2

If DF = 1, then (DI) ← (DI) - 2 ; (SI) ← (SI) - 2

- SCAS / SCASB / SCASW: Scan string byte or String word

One byte or word of a string data stored in extra segment is subtracted from the contents of AL / AX and the result modifies the flags.

The DI register points to the string byte or word.

The CX register is decremented by one for each byte / word movement.

The DI register is automatically incremented or decremented depending on the status of DF.

MA = (DS) X 16_{10} + (SI)

MA_E = (ES) X 16_{10} + (DI)

Modify flags ← (AL) - (MA_E) / (AX) - (MA_E : MA_E + 1)

The WAIT instruction is used to synchronize the 8086 processor with the external hardware such as the 8087 math processor.

- HLT : Halt Processing

The HLT instruction will cause the 8086 to stop the fetching and execution of the instructions. The 8086 will enter a halt state i.e., used to terminate a program.

The only ways to get processor out of Halt state are with an interrupt signal on INTR pin, an interrupt signal on NMI pin, or a valid reset signal on RESET input.

- NOP : No Operation

No operation is performed for three clock periods

This instruction simply uses up three clock cycles and increments the instruction pointer to point to the next instruction.

The NOP instruction does not affect any flag.

The NOP instruction can be used to increase the delay of a delay loop.

When hand coding, a NOP can also be used to hold a place in a program for instruction that will be added later.

- ESC : Escape

ESC opcode, Mem. / Reg.

This instruction is used to pass instructions to a coprocessor , such as the 8087 math coprocessor which shares the address and data bus with 8086

Instructions for coprocessor are represented by a 6-bit code embedded in the escape instruction.

As 8086 fetches the instructions bytes, the coprocessor also catches these bytes from the data bus and puts them in its queue , but treats all the normal 8086 instructions as NOPs and when ESC instruction is fetched by 8086, the coprocessor decodes the instruction and carries out the action specified by the 6-bit code in the instruction.

In most cases 8086 treats the ESC instruction as NOP but in some cases 8086 will access a data item in memory for the coprocessor.

For ESC opcode, Mem format the data is accessed by 8087 from memory

For ESC opcode, Mem format the data is accessed by 8087 from 8086 register specified in the instruction.

- LOCK : Assert Bus Lock signal

The LOCK is used as a prefix to a critical instruction which has to be executed without any disturbances to system bus from other bus masters.

When LOCK prefix is used in an instruction then during execution of this instruction the lock prefix ensures that the shared system resources are not taken over by other bus masters in the middle of the critical instruction execution.

When an instruction with LOCK prefix is executed the 8086 will assert its bus lock signal output.This signal is connected to an external bus controller device, which then prevents any other processor from taking over the system bus

LOCK affects no flags.

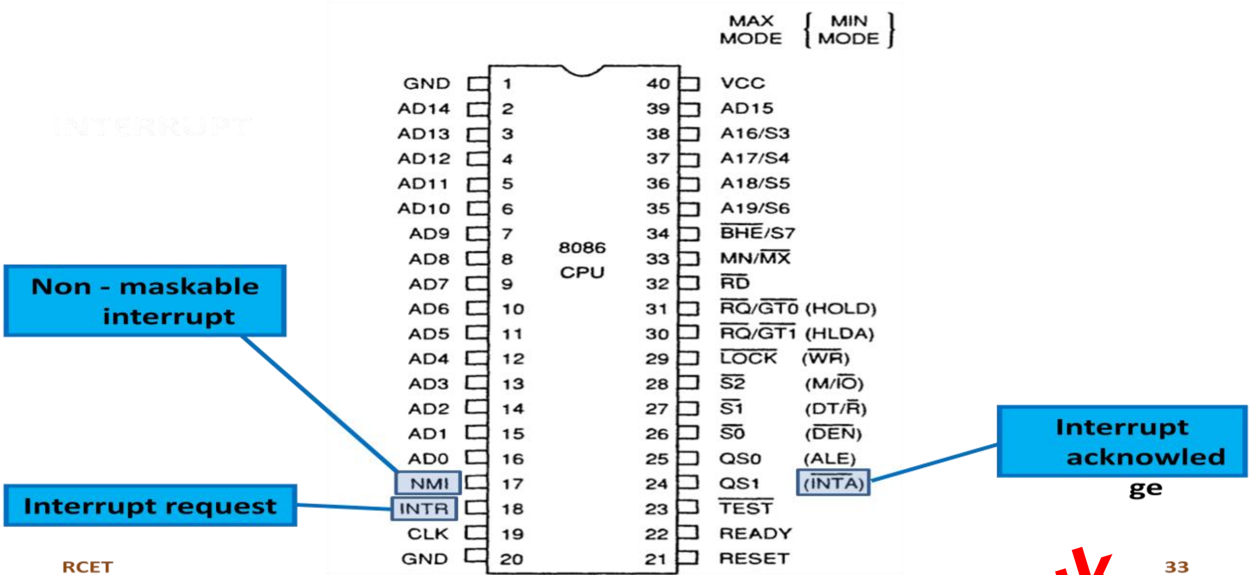
- Program execution transfer instructions

- The control transfer group consists of call, jump, loop and software interrupt instructions.
- Normally a program is executed sequentially(i.e., the program instructions are executed one after the other), when a branch instruction is encountered the program execution control is transferred to the specified destination or target instructions. The transfer of program execution control is done either by changing the content of IP or by changing the contents of IP and CS.
- When the content of IP alone is modified, the program control branches to new memory location in the same segment.
- When the contents of IP and CS are modified, the program control branches to new memory location in another memory segment.
- The control transfer instructions do not affect the flags of 8086.
- The jump and loop instructions can be classified into conditional and unconditional instructions.
- In conditional instructions, the status of one or more flags are checked and control transfer takes place only if the specified condition is satisfied.
- The program execution transfer instructions can be categorized as:
  - Unconditional transfer instructions
  - Conditional transfer instructions
  - Iteration control instructions
  - Software interrupt instructions
- Unconditional transfer instructions:

CALL

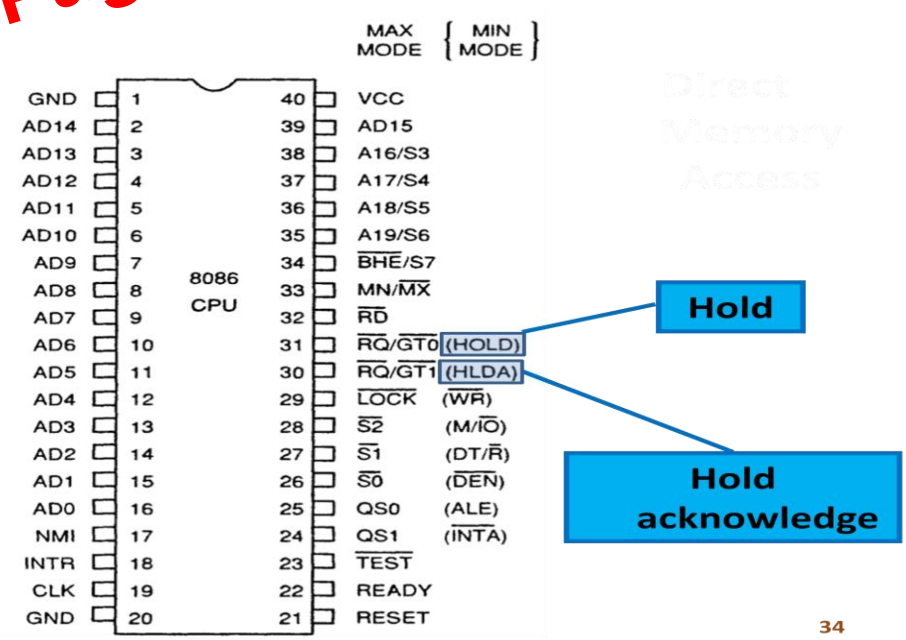
RET

INTEL 8086 - Pin Details



RCET 33

INTEL 8086 - Pin Details



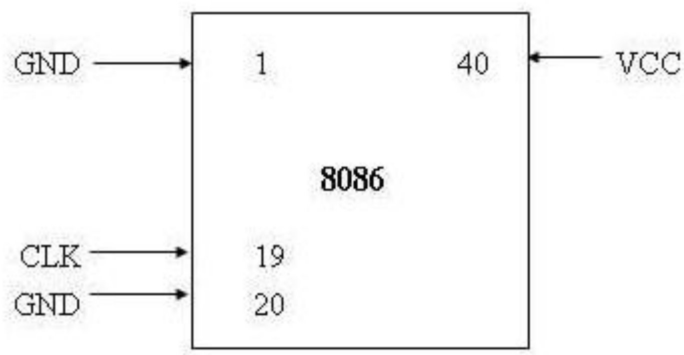
RCET 34

$AD_{15} - AD_0$	Address/ Data Bus	Bidirectional 3 - state
$A_{19}/S_6 - A_{16}/S_3$	Address / Status	Output 3 - State
BHE /S7	Bus High Enable / Status	Output 3- State
MN / MX	Minimum / Maximum Mode Control	Input
RD	Read Control	Output 3- State
TEST	Wait On Test Control	Input
READY	Wait State Controls	Input
RESET	System Reset	Input
NMI	Non - Maskable Interrupt Request	Input
INTR	Interrupt Request	Input
CLK	System Clock	Input
Vcc	+ 5V	Input
GND	Ground	

Minimum Mode Signals (MN/ MX = Vcc)		
Name	Function	Type

duty cycle to provide optimized internal timing. Minimum frequency of 2 MHz is required, since the design of 8086 processors incorporates dynamic cells. The maximum clock frequencies of the 8086-4, 8086 and 8086-2 are 4MHz, 5MHz and 8MHz respectively. Since the 8086 does not have on-chip clock generation

circuitry, and 8284 clock generator chip must be connected to the 8086 clock pin. The crystal connected to 8284 must have a frequency 3 times the 8086 internal frequency. The 8284 clock generation chip is used to generate READY, RESET and CLK.



MN/ MX **(I): Maximum / Minimum**

This pin indicates what mode the processor is to operate in. In minimum mode, the 8086 itself generates all bus control signals. In maximum mode the three status signals are to be decoded to generate all the bus control signals.

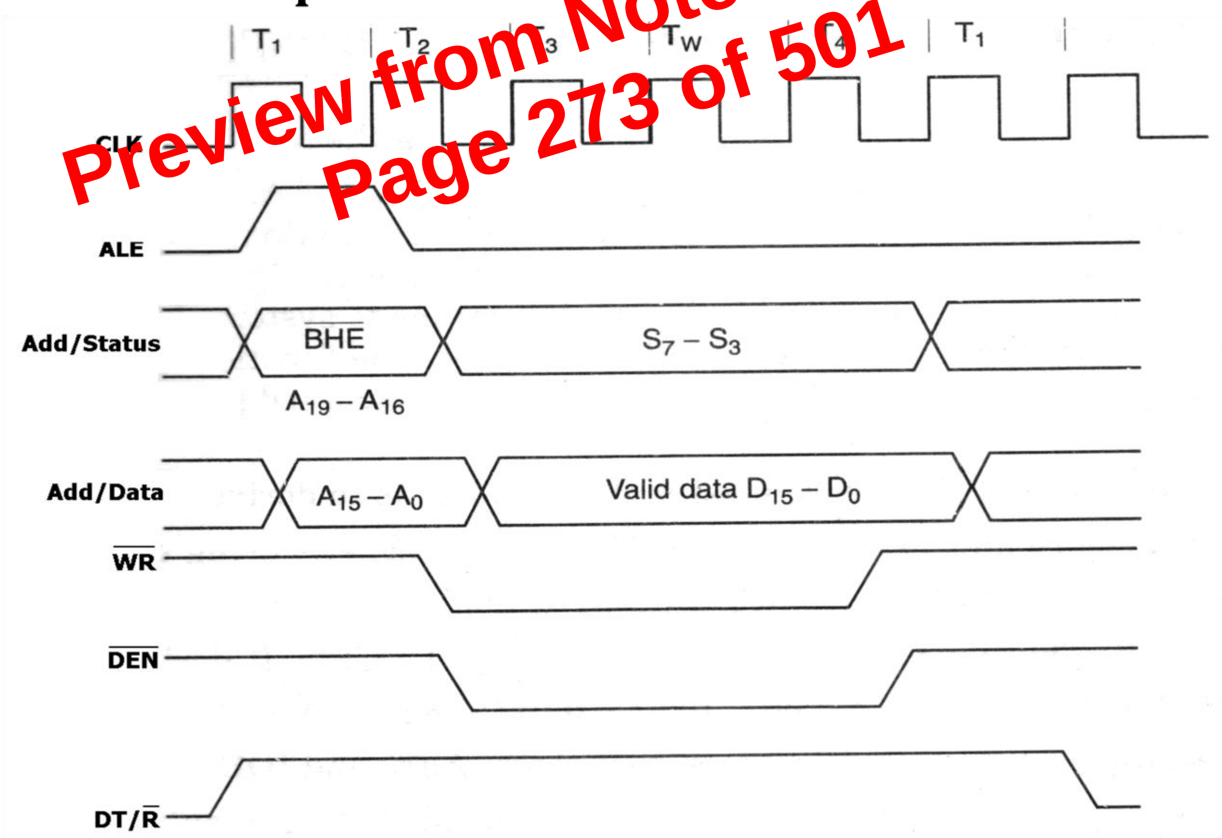
Minimum Mode Pins

The following 8 pins function descriptions are for the 8086 in minimum mode; MN/ MX = 1. The corresponding 8 pins function descriptions for maximum mode is explained later.

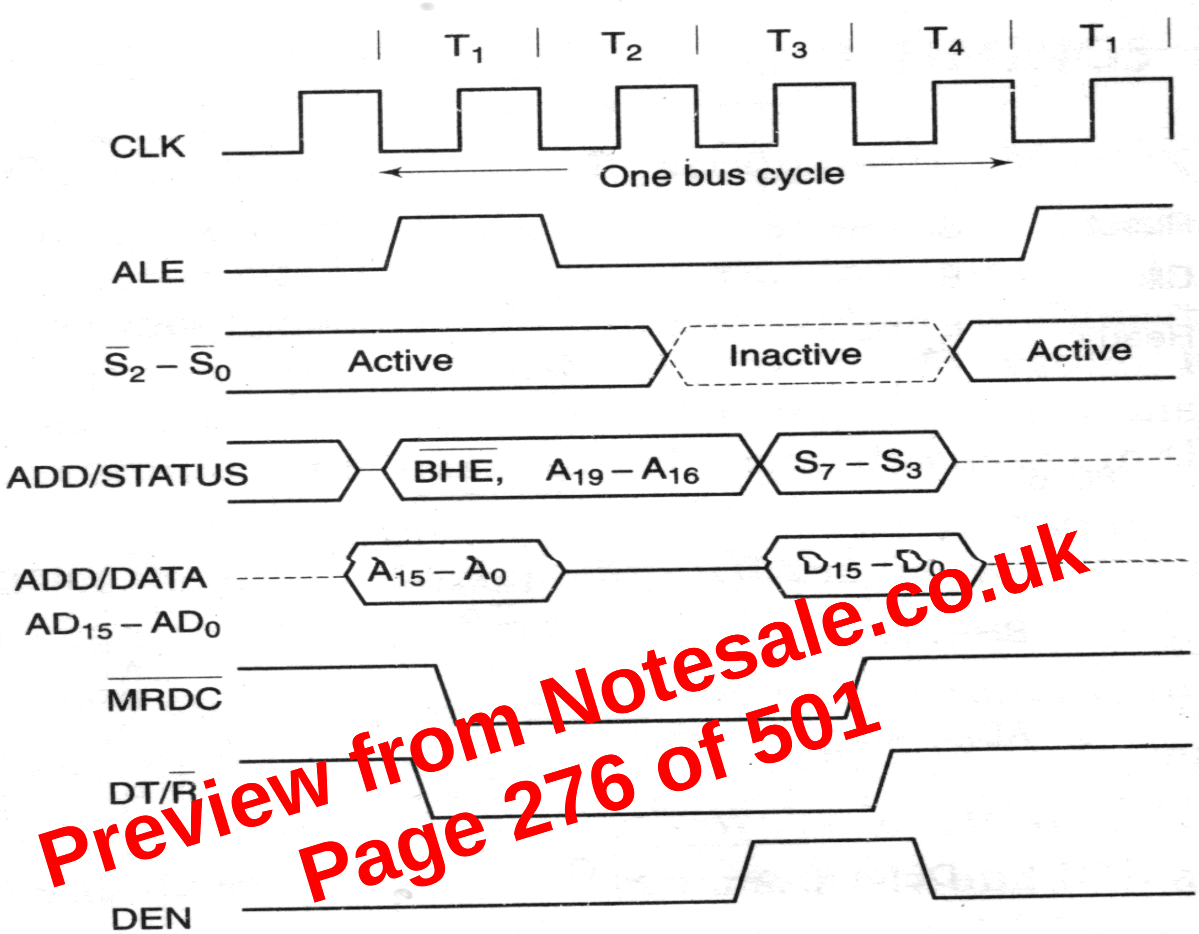
In the bus timing diagram, data transmit / receive signal goes low (RECEIVE) for Read operation. To validate the data, DEN* signal goes low. The Address/ Status bus carries A16 to A19 address lines during BHE* (low) and for the remaining time carries Status information. The Address/Data bus carries A0 to A15 address information during ALE going high and for the remaining time it carries data. The RD* line going low indicates that this is a Read operation. The curved arrows indicate the relationship between valid data and RD* signal.

The TW is Wait time needed to synchronize the fast processor with slow memory etc. The Ready pin is checked to see whether any peripheral needs more time for data transmission.

Write Cycle timing Diagram for Minimum Operation



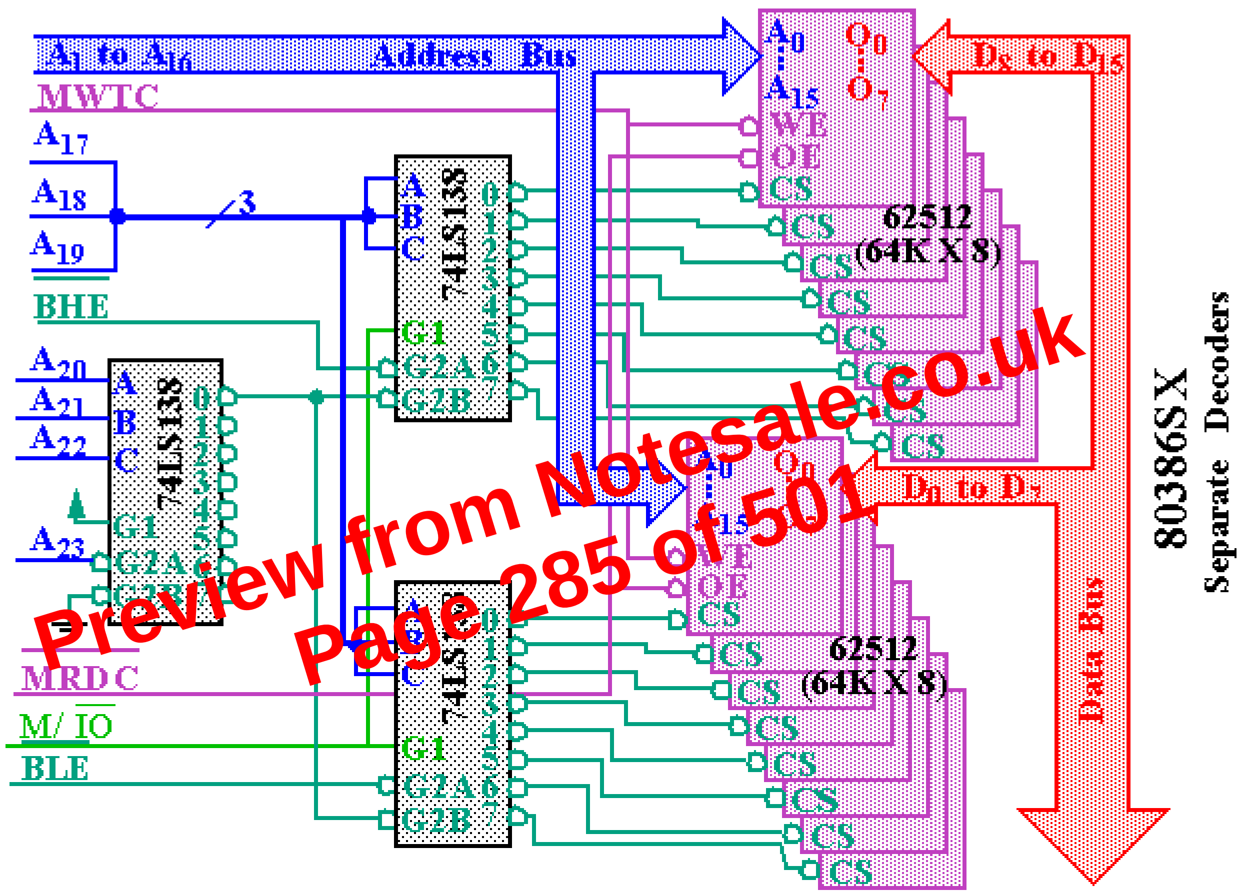
Memory Read timing in Maximum Mode



- *There does not seem to be a big difference between these methods although the book claims that there is.*

- *Note in either method that A_0 does not connect to memory and bus wire A_1 connects to memory pin A_0 , A_2 to A_1 , etc.*

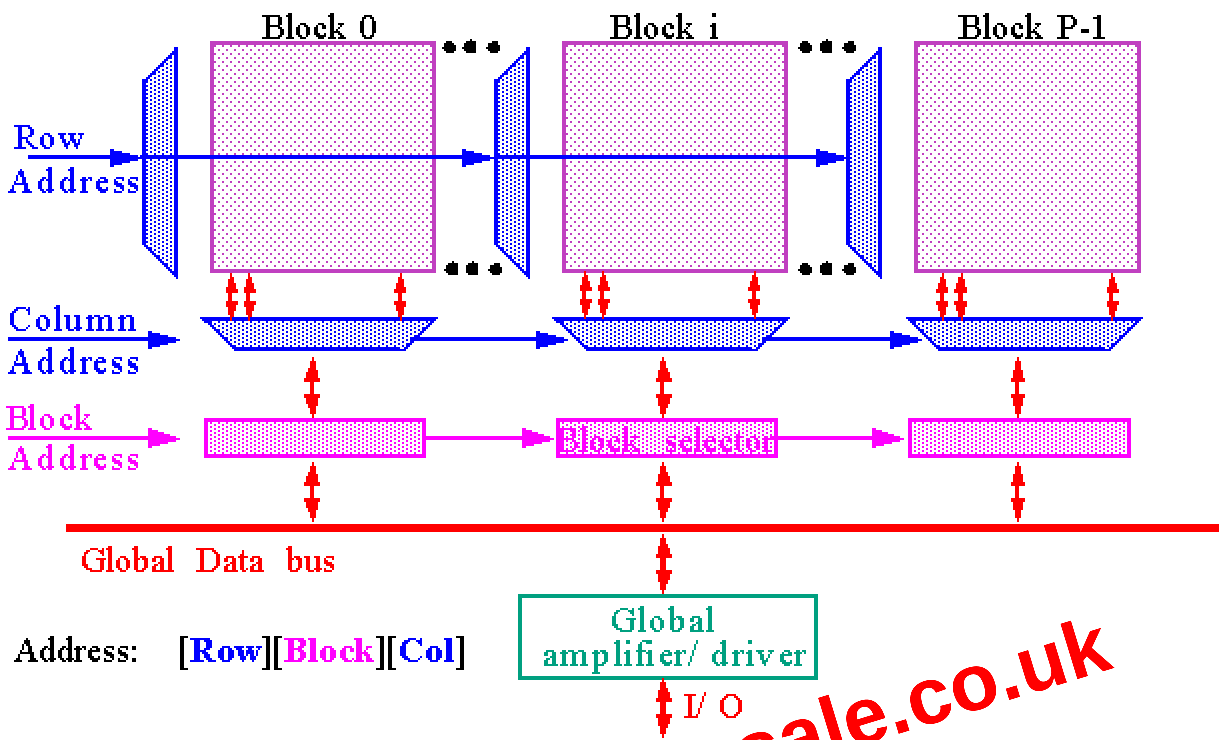
80386SX 16-bit Memory Interface (Separate Decoders)



Memory Interfaces

- *See text for Separate Write Strobe scheme plus some examples of the integration of EPROM and SRAM in a complete system.*
    - *It is just an application of what we've been covering.*

- *80386DX and 80486 have 32-bit data buses and therefore 4 banks of memory.*
    - *32-bit , 16-bit and 8-bit transfers are accomplished by different combinations of the bank selection signals* BE3 , BE2 , BE1 , BE0 .



Dynamic RAM

- *DRAM requires refreshing every 2 to 4 ms.*
  - *Refreshing occurs automatically during a read or write.*
  - *Internal circuitry takes care of refreshing cells that are not accessed over this interval.*

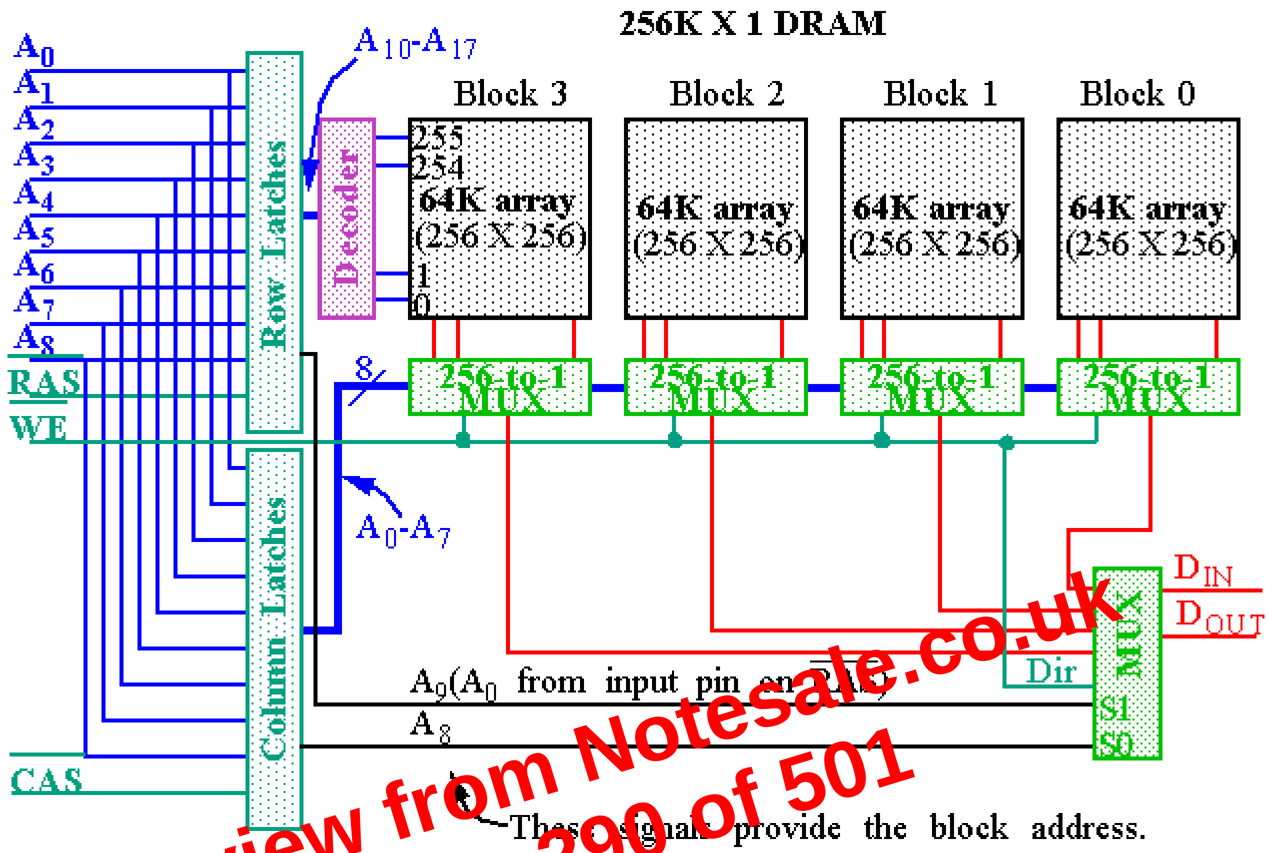
- *This special refresh occurs* **transparently** *while other memory components operate and is called transparent refresh or cycle stealing .*

- *A* RAS *-only cycle strobes a row address into the DRAM, obtained by 7- or 8-bit binary counter.*

- *The capacitors are recharged for the selected row by reading the bits out internally and then writing them back.*

- *For a 256K X 1 DRAM with 256 rows, a refresh must occur every 15.6us (4ms/256).*
  - *For the 8086, a read or write occurs every 800ns .*
  - *This allows* **19** *memory reads/writes per refresh or* **5%** *of the time.*

Dynamic RAM



E1. Interface two 4K X 8 EPROMs and two 4K X 8 RAM chips with 8086. Select suitable maps.

First we have to write the memory map fro the problem given. It will reveal the logic to be used for decoding circuit.

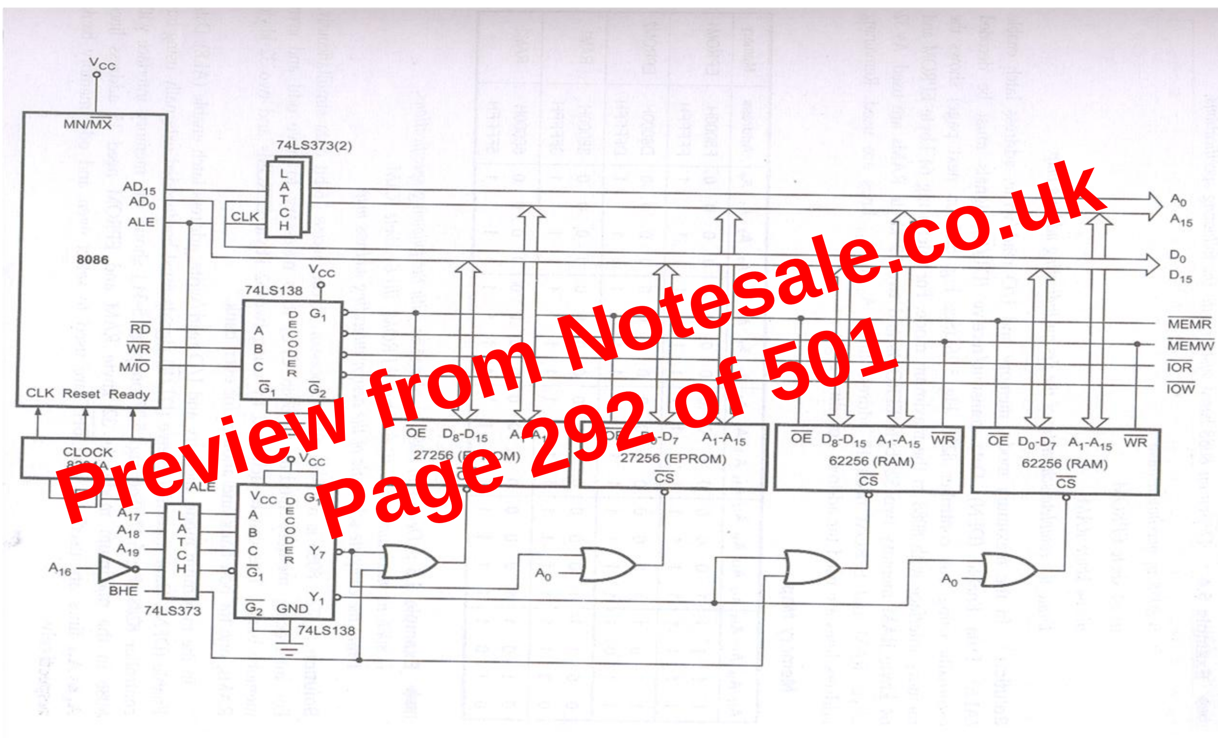
Since the first instruction is fetched from FFFF0h after the microprocessor is reset, we will make that address to be present in EPROM and write the memory map as follows. And, to avoid windowing let us keep the locations to be present in the RAM as immediate addresses.

Locations having addresses from FFFFFH to FE000H are allocated to EPROM1 and 2.

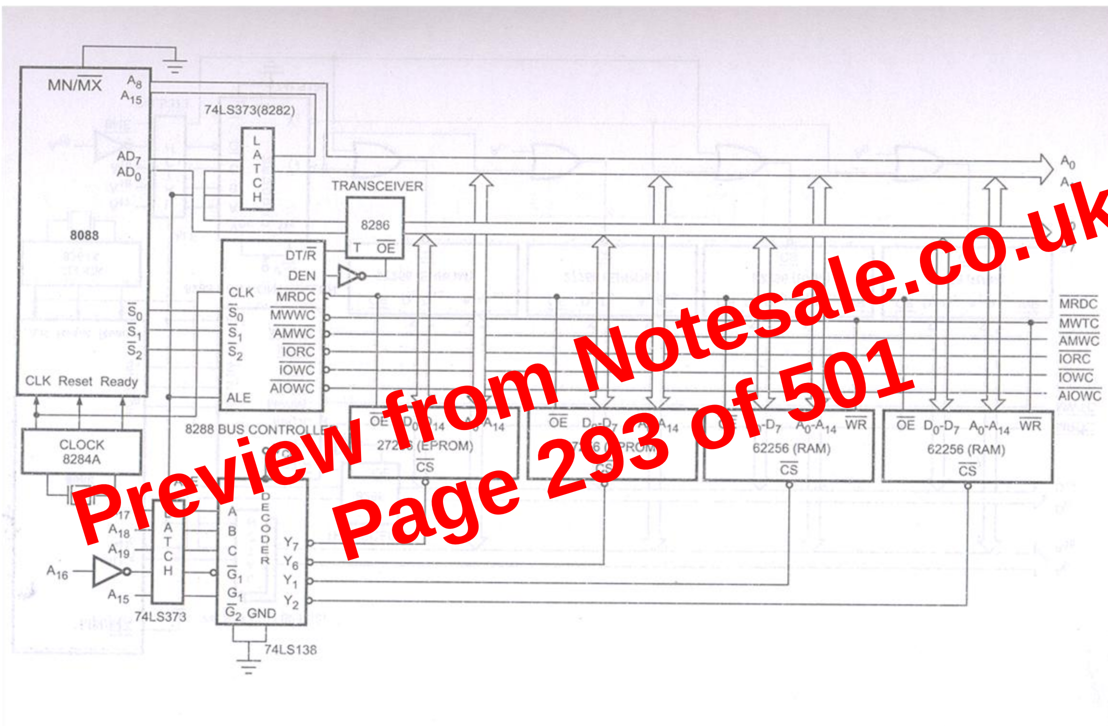
Immediate address map FDFFFH to FD000H is allocated to RAM1 and 2. The line which is differentiating EPROM from RAM if A_{13}. Let us use it along with A_0 and *BHE* to identify odd and even banks.

A_{19} A_{18} A_{17} A_{16}	A_{15} A_{14} A_{13} A_{12}	A_{11} A_{10} A_9 A_8	A_7 A_6 A_5 A_4	A_3 A_2 A_1 A_0	Memory Address in
1 1 1 1 1 1 1 1	1 1 1 1 1 1 0 1	1 1 1 1 0 0 0 0	1 1 1 1 0 0 0 0	1 1 1 1 0 0 0 0	FFFFFH To FE000H

Interfacing 64k RAM and 64k EPROM with 8086 min. mode



Interfacing 64 RAM and 64k Eprom with 8088 max mode



General organization of the DMA controller

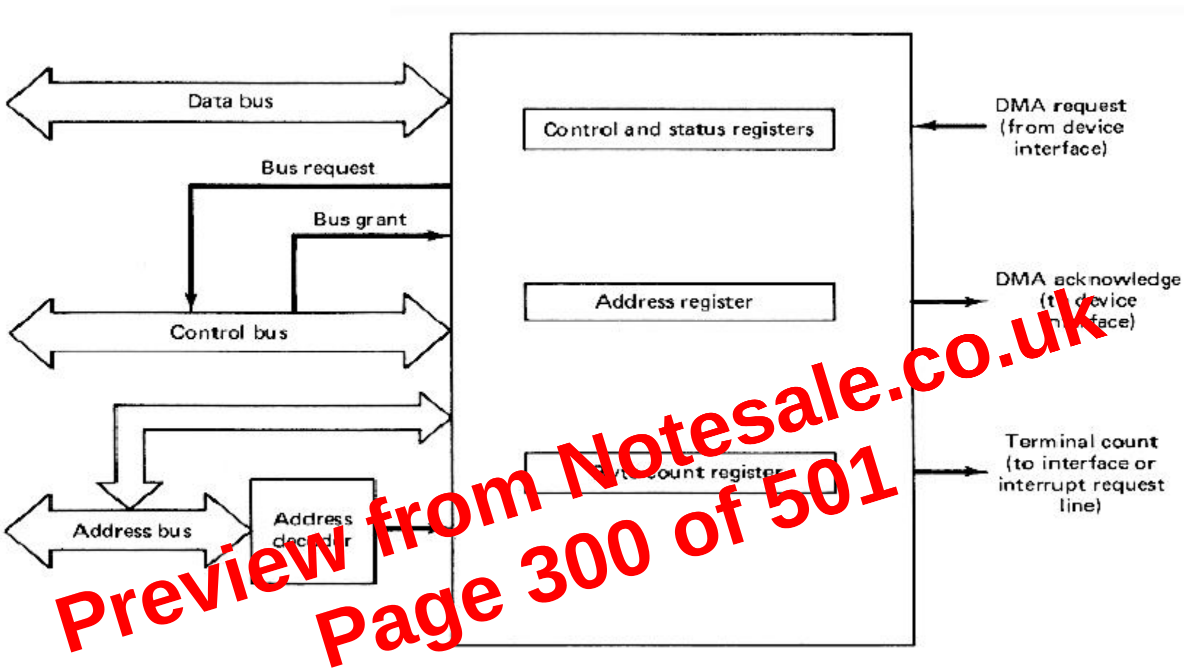


Figure 9-36 General organization of a DMA controller.

DMA operation:

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.

EXAMPLE

Assuming that a DMA initialization has an overhead of 10 cycles, while a CPU transfer to/from memory requires 4 cycles (no wait states required), compare a DMA and a CPU transfer from one memory location to another of

One byte of data

A block of 1Kbytes in burst mode

A block of 64Kbytes in burst mode

DMA controller

A DMA controller interfaces with several peripherals that may request DMA.

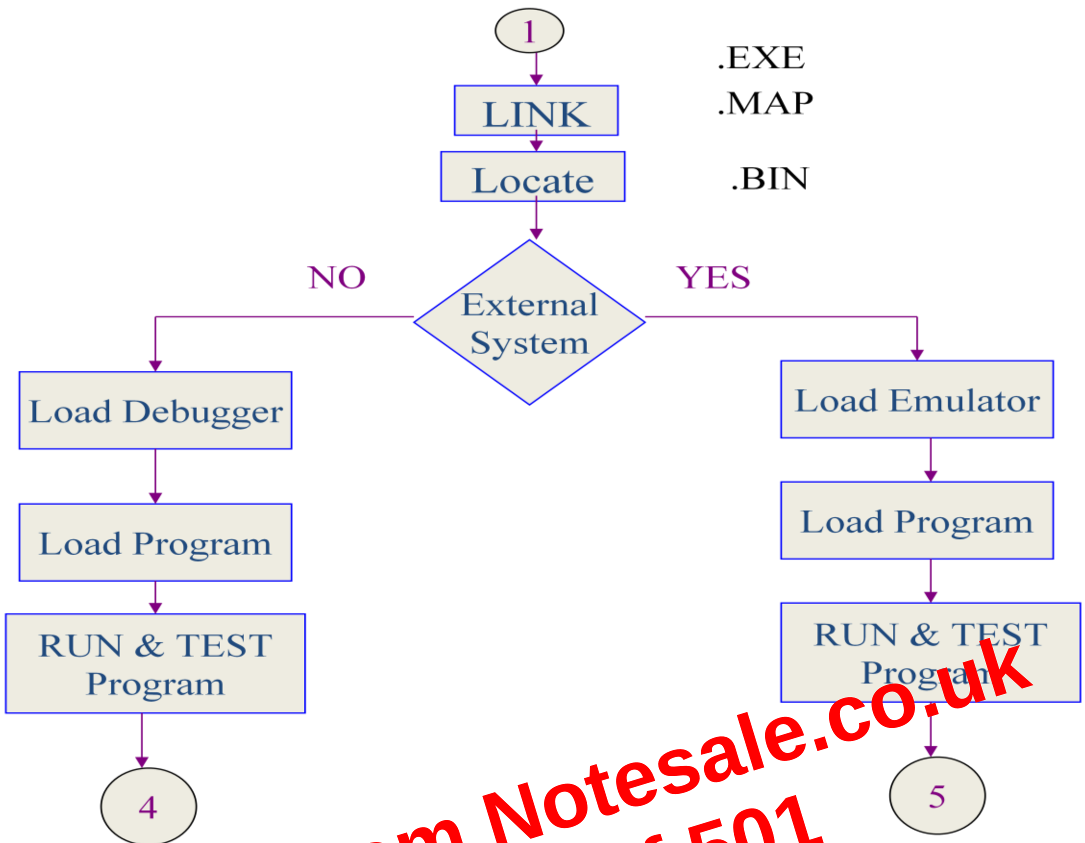
The controller decides the priority of simultaneous DMA requests communicates with the peripheral and the CPU, and provides memory addresses for data transfer.

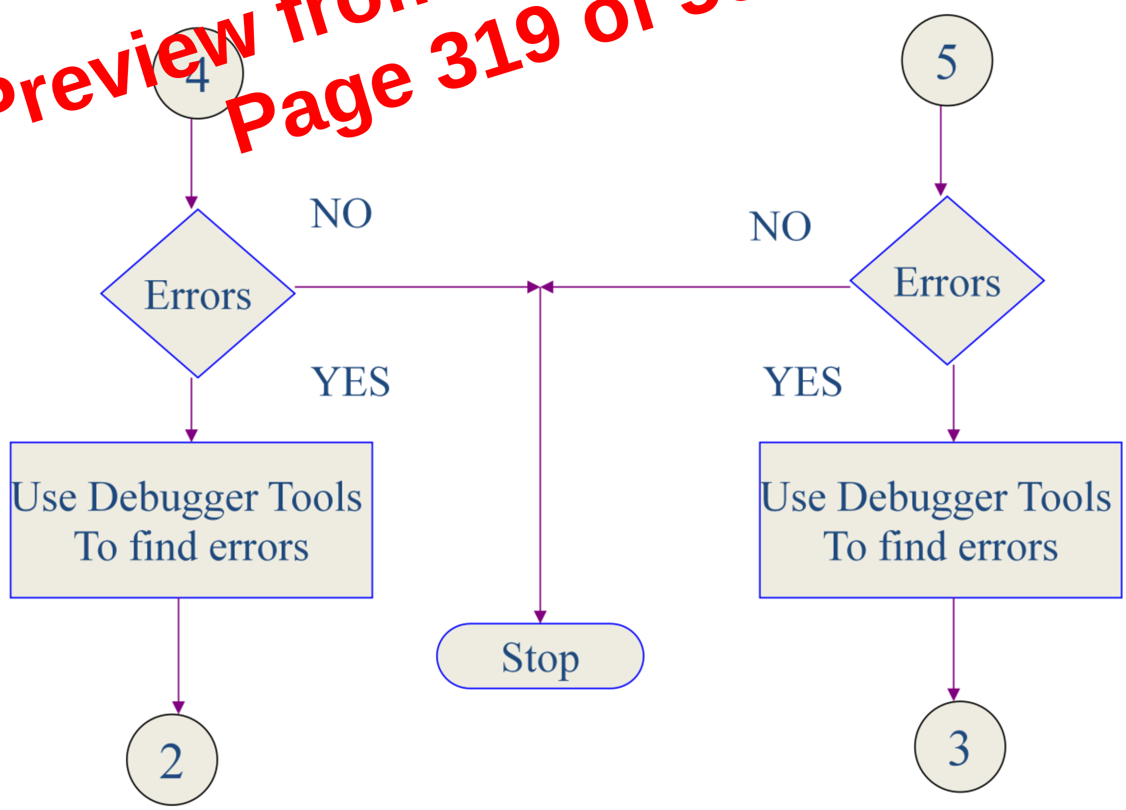
Advantages of DMA

- Fast memory transfer of data
- CPU and DMA run concurrently under cache mode
- DMA can trigger an interrupt, which frees the CPU from polling the channel

- Usually used to test and debug the hardware and software of an external system like prototype of microprocessor based instruments.

- Emulator have a multi-wire cable which connects the host system to prototype system.


1
LINK
Locate
.EXE
.MAP
.BIN
NO
YES
External System
Load Debugger
Load Program
RUN & TEST Program
4
Load Emulator
Load Program
RUN & TEST Program
5


4
5
NO
NO
Errors
Errors
YES
YES
Use Debugger Tools To find errors
Use Debugger Tools To find errors
Stop
2
3

- Machine Level Language Programming

➢ Generating the machine codes of program manually and execute it.

➢ Disadvantages of MLP :

  - The process is complicated and time consuming.
  - The chances of error being committed are more at the machine level(in hand-coding and entering the program byte-by-byte into the system).
  - Debugging a program at the machine-level is more difficult.
  - The programs are not understood by every one and the results are not stored in user friendly form.

➢ A program called 'assembler' is used to convert the mnemonics of instructions along with the data into their equivalent object code modules, which may further be converted in executable code using the linker and loader programs.

This type of programming is called Assembly level language programming(ALP).

In ALP, the mnemonics are directly used in the user programs. The assembler performs the task of coding.

➢ Advantages of ALP over MLP:

  - The programming in assembly language is not so complicated as in machine language because the function of coding is performed by an assembler.
  - The chances of error being committed are less because the mnemonics are used instead of numerical opcodes. It is easier to enter an ALP.
  - As the mnemonics are purpose suggestive, the debugging is easier.

- The constants and address locations can be labeled with suggestive labels hence imparting a more user friendly interface to user.Advanced assemblers provide facilities like macros, lists,……… etc making the task of programming much easier.
- The memory control is in the hands of users as in machine language.
- The results may be stored in a more user-friendly form.
- The flexibility of programming is more in assembly language programming as compared to machine language programming because of advanced facilities available with the modern assemblers.

➢ Assembly language programming (ALP) explains the way the computer hardware and operating system work together and also about how the application programs communicate with the operating system.

- Assembler Directives

- DD (Define Double Word) : Each operand datum is two words long with low-order word followed by high-order word.
- DQ(Define Quad Word) : Each operand data is 4 words long i.e. 8 bytes and is stored starting from lowest byte to higher bytes.
- DT(Define Ten Bytes) : Each operand datum is 10 bytes long and is stored starting from the lowest byte to higher bytes.
- The character string is stored in between single quotes and each character's ASCII codes are saved(stored) in successive locations(i.e. the first character goes into the first byte assigned to the variable, then second character of the string is stored at second byte and so on………..)
- The character string is defined or pre-assigned using byte type.
- The character string can be defined using DW and DD also, but they are rarely used as the bytes are reversed and also string operands in a DW or DD cannot exceed two characters in length.
- When an question mark (?) is used along with mnemonic as second mnemonic, it does not pre-assign any value but, the appropriate amount of space is reserved.
- DUP operator is used along with data size defining mnemonics to duplicate multiple locations with the specified value in the braces.

  ex: DUP(0) → fills 0's in all the variable locations.

- Types of numbers used in data statements:
  - Binary : ex → 11100101B
  - Negative number is represented in 2's complement sign-magnitude form.
  - Any binary number is represented using a suffix of character 'B'.
  - Decimal : ex → 20/ 20D / -20 / -20D
  - Any decimal number is represented using a suffix of character 'D'.
  - Decimal is default data type.
- Hexadecimal : ex: 30H
  - Always any hexadecimal number is represented using a suffix of character 'H'.
  - Always a '0'(zero) is to be appended in front of a hexadecimal number that starts with alphabet.

$\overline{RD}$	$\overline{WR}$	$\overline{CS}$	A0	A1	INPUT (READ) CYCLE
0	1	0	0	0	PORT A TO DATA BUS
0	1	0	0	1	PORT B TO DATA BUS
0	1	0	1	0	PORT C TO DATA BUS
0	1	0	1	1	CWR TO DATA BUS

$\overline{RD}$	$\overline{WR}$	$\overline{CS}$	A0	A1	OUTPUT (WRITE) CYCLE
1	0	0	0	0	DATA BUS TO PORT A
1	0	0	0	1	DATA BUS TO PORT B
1	0	0	1	0	DATA BUS TO PORT C
1	0	0	1	1	DATA BUS TO CWR

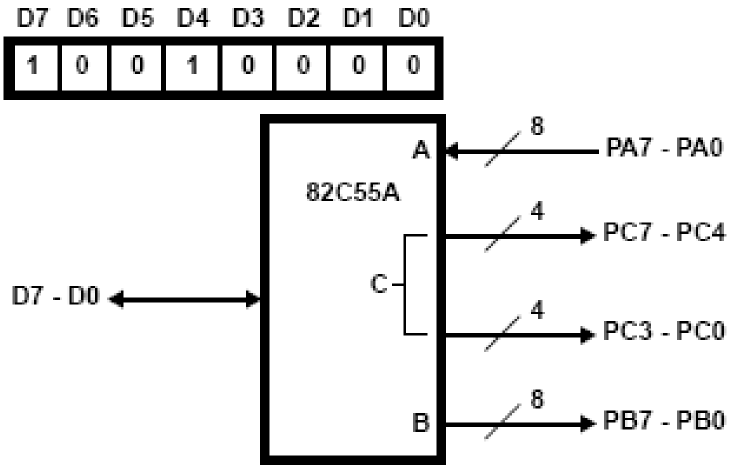
$\overline{RD}$	$\overline{WR}$	$\overline{CS}$	A0	A1	FUNCTION
X	X	1	X	X	DATA BUS TRISTATED
1	1	0	X	X	DATA BUS TRISTATED

MODE - 0 (BASIC I/O MODE)

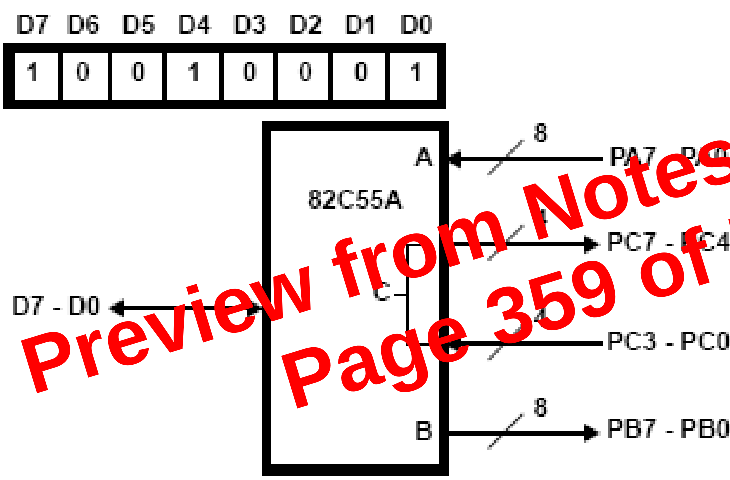
SALIENT FEATURES OF THIS MODE

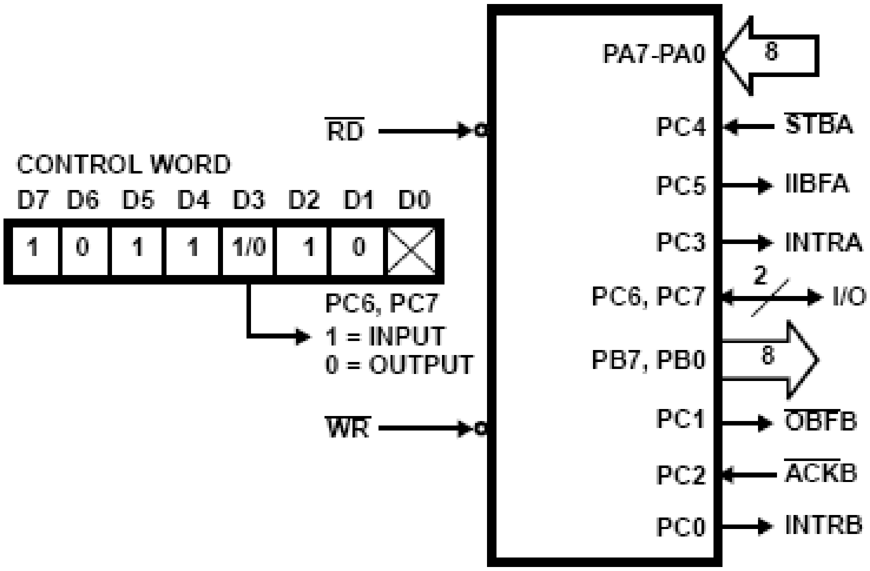
1. **Two 8-bit ports (port A and port B) and two 4- bit ports (port C upper and port C lower) are available.**
2. **The two 4-bit ports can be combined used as a third 8-bit port.**
3. **Any port can be used as an input or output port.**
4. **Output ports are latched. Input ports are not latched.**
5. **A maximum of four ports are available so that overall 16 I/O configurations are possible.**

CONTROL WORD #8

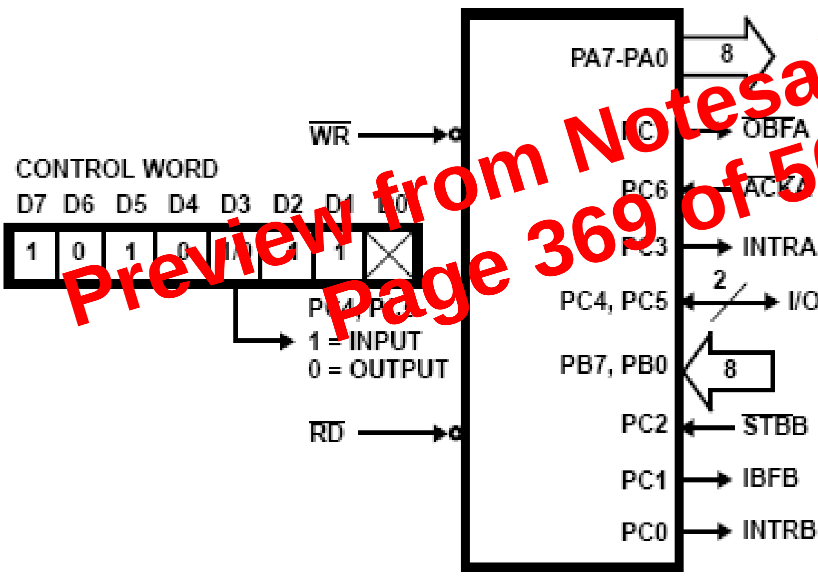


CONTROL WORD #9





PORT A - (STROBED INPUT)
PORT B - (STROBED OUTPUT)



PORT A - (STROBED OUTPUT)
PORT B - (STROBED INPUT)

Combinations of Mode 1:

Port A and Port B can be individually defined as input or output in Mode 1 to support a wide variety of Strobbed I/O applications.

Distinguish between Microprocessor and Microcontroller

S.No	Microprocessor	Microcontroller
1	A microprocessor is a general purpose device which is called a CPU	A microcontroller is a dedicated chip which is also called single chip computer.
2	A microprocessor do not contain onchip I/OPorts, Timers, Memories etc..	A microcontroller includes RAM, ROM, serial and parallel interface, timers, interrupt circuitry (in addition to CPU) in a single chip.
3	Microprocessors are most commonly used as the CPU in microcomputer systems	Microcontrollers are used in small, minimum component designs performing control-oriented applications.
4	Microprocessor instructions are mainly nibble or byte addressable	Microcontroller instructions are both bit addressable as well as byte addressable.
5	Microprocessor instruction sets are mainly intended for catering to large volumes of data.	Microcontrollers have instruction sets catering to the control of inputs and outputs.

Microcontrollers for Embedded Systems

- **Home**
  - Appliances, intercom, telephones, security systems, garage door openers, answering machines, fax machines, home computers, TVs, cable TV tuner, VCR, camcorder, remote controls, video games, cellular phones, musical instruments, sewing machines, lighting control, paging, camera, pinball machines, toys, exercise equipment etc.
- **Office**
  - Telephones, computers, security systems, fax machines, microwave, copier, laser printer, color printer, paging etc.
- **Auto**
  - Trip computer, engine control, air bag, ABS, instrumentation, security system, transmission control, entertainment, climate control, cellular phone, keyless entry

Criteria for Choosing a Microcontroller

- Meeting the computing needs of the task at hand efficiently and cost effectively
  - Speed
  - Packaging
  - Power consumption
  - The amount of RAM and ROM on chip
  - The number of I/O pins and the timer on chip
  - How easy to upgrade to higher performance or lower power-consumption versions
  - Cost per unit
- Criteria for Choosing a Microcontroller
- Availability of software development tools, such as compilers, assemblers, and debuggers

- Wide availability and reliable sources of the microcontroller
  - The 8051 family has the largest number of diversified (multiple source) suppliers
    - Intel (original)
    - Atmel
    - Philips/Signetics
    - AMD
    - Infineon (formerly Siemens)
    - Matra
    - Dallas Semiconductor/Maxim

TYPES OF MICROCONTROLLERS :

Microcontrollers can be classified on the basis of internal bus width, architecture, memory and instruction set as 4-bit,8-bit,16-bit and 32-bit micrcontrollers.

4-bit Microcontrollers: These 4-bit microcontrollers are small size, minimum pin count and low cost controllers which are widely used for low end applications like LED & LCD display drivers ,portable battery chargers etc.. Their power consumption is also low. The popular 4-bit controllers are Renasa M34501 which is a 20 pin DIP chip with 4kB of ROM,256 Bytes of RAM,2-Counters and 14 I/O Pins. Similarly ATAM862 series from ATMEL.

8-bit Microcontrollers : These are the most popular and widely used microcontrollers .About 55% of all CPUs sold in the world are 8-bit microcontrollers only.The 8-bit microcontroller has 8-bitinternal bus and the ALU performs all the arithmetic and logical operations on a byte instruction. The well known 8-bit microcontroller is 8051 which was designed by Intel in the year 1980 for the use in embedded systems. Other 8-bit microcontrollers are Intel 8031/8052 and Motorola MC68HC11 and AVR Microcontrollers, Microchip's PIC Microcontrollers 12C5XX ,16C5X and 16C505 etc...

16-bit Microcontrollers : When the microcontroller performs 16-bit arithmetic and logical operations at an instruction, the microcontroller is said to be a 16-bit microcontroller. The internal bus width of 16-bit microcontroller is of 16-bit. These microcontrollers are having increased memory size and speed of operation when compared to 8-bit microcontrollers.These

Manufacturer	I/O	Pins	RAM (bytes)	ROM (bytes)	Counters	Features
80c196(INTEL)	40	68	232 8K	2	—	PWM generator, watchdog timer
HPC Family (National)	52	68	512 16K	4	—	PWM generator, watchdog timer, 8-channel A/ D, serial port

16-Bit Microcontrollers

The following table gives the list of PIC microcontrollers from Micro chip Inc

Microcontroller	Pins	I/O Lines	On chip ADCs	EPROM X 12 words	On chip RAM (Bytes)
16C54	18	12	None	512	25
16C55	28	20	None	512	24
16C56	18	12	None	1k	25
16C57	28	20	None	2k	72
17C42A	40	33	None	2k	232
17C43	40	33	None	4k	454
17C44	40	33	None	8k	454
17C71	18	13	8bit ADCs	1kx14	36
17C752	40	33	10Bit ADC	8kx16	678

Development/Classification of microcontrollers (Invisible)

Microcontrollers have gone through a silent evolution (invisible). The evolution can be rightly termed as silent as the impact or application of a microcontroller is not well known to a common user, although microcontroller technology has undergone significant change since early 1970's. Development of some

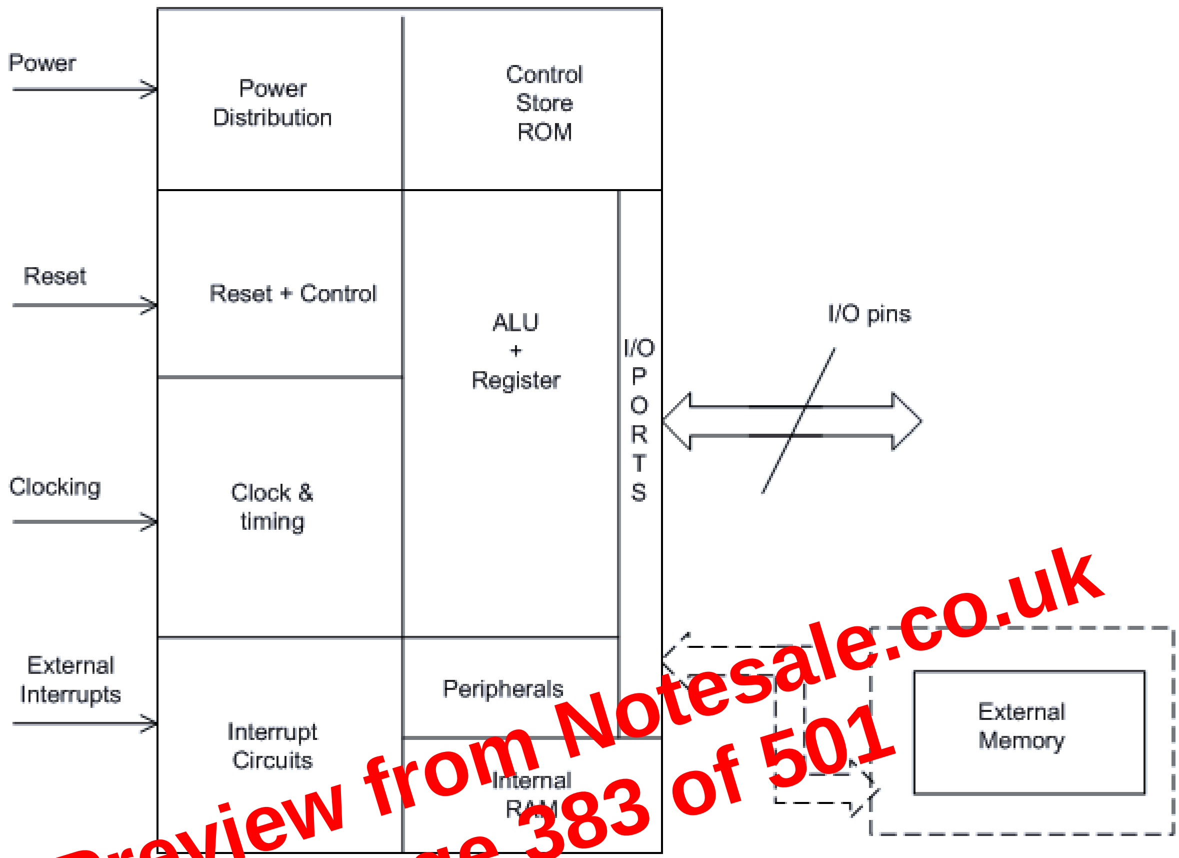


Fig. 2.1 Internal Structure of a Microcontroller

At times, a microcontroller can have external memory also (if there is no internal memory or extra memory interface is required). Early microcontrollers were manufactured using bipolar or NMOS technologies. Most modern microcontrollers are manufactured with CMOS technology, which leads to reduction in size and power loss. Current drawn by the IC is also reduced considerably from 10mA to a few micro Amperes in sleep mode(for a microcontroller running typically at a clock speed of 20MHz).

Harvard vs. Princeton Architecture

Many years ago, in the late 1940's, the US Government asked Harvard and Princeton universities to come up with a computer architecture to be used in computing distances of Naval artillery shell for defense applications. Princeton suggested computer architecture with a single memory interface. It is also known as Von Neumann architecture after the name of the chief scientist of the project in Princeton University John Von Neumann (1903 - 1957 Born in Budapest, Hungary).

Harvard suggested a computer with two different memory interfaces, one for the data / variables and the other for program / instructions. Although Princeton architecture was accepted for simplicity and ease of implementation, Harvard architecture became popular later, due to the parallelism of instruction execution.

Princeton Architecture (Single memory interface)

Fig. 2.3   Harvard Arcitecture

The same instruction (as shown under Princeton Architecture) would be executed as follows:
Cycle 1
- Complete previous instruction
- Read the "Move Data to Accumulator" instruction

Cycle 2
- Execute "Move Data to Accumulator" instruction
- Read next instruction

Hence each instruction is effectively executed in one instruction cycle, except for the ones that modify the content of the program counter. For example, the "jump" (or call) instructions takes 2 cycles. Thus, due to parallelism, Harvard architecture executes more instructions in a given time compared to Princeton Architecture.

MICROCONTROLLER DEVELOPMENT TOOLS:

To develop an assembly language program we need certain program development tools. An assembly language program consists of Mnemonics which are nothing but short abbreviated English instructions given to the controller. The various development tools required for Microcontroller programming are explained below.

1. Editor : An Editor is a program which allows us to create a file containing the assembly language statements for the program. Examples of some editors are PC write Wordstar. As we type the program the editor stores the ACSII codes for the letters and numbers in successive RAM locations. If any typing mistake is done editor will alert us to correct it. If we leave out a program statement an editor will let you move everything down and insert a line. After typing all the program we have to save the program . This we call it as source file. The next step is to process the source file with an assembler.

Ex: Sample. asm

2.Assembler : An Assembler is used to translate the assembly language mnemonics into machine language(i.e binary codes). When you run the assembler it reads the source file of your program from where you have saved it. The assembler generates a filee with the extension **.hex**. This file consists of hexadecimal values encoding a sequence of data and their starting offset or absolute address.

- Two 16-bit timer/counters
- Full duplex UART
- 6-source/5-vector interrupt structure with two priority levels
- On-chip clock oscillator

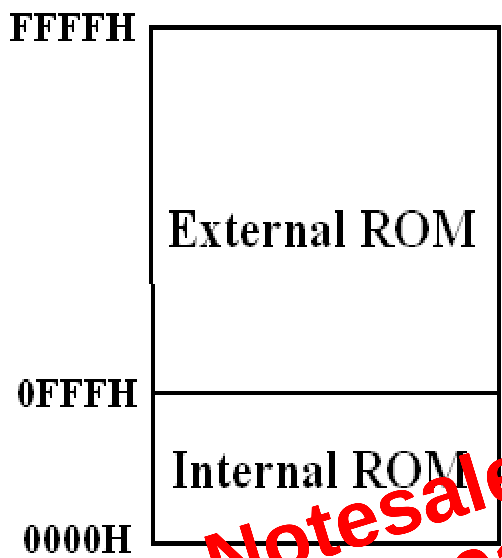
- It is a 40 pin dip (dual-in-line) package.
- Eight bit cpu with rigisters A(accumlater) and B.
- 8051 available in both NMOS & CMOS.
- Sixteen bit program counter (PC) and data pointer (DPTR).
- Eight bit program status word (PSW).
- FOUR register banks,each containing 8-registers.
- FOUR ports, each port having 8-bits.
- FULL DUPLEX serial data reciver/ transmitter.

ARCHITECTURE & BLOCK DIAGRAM OF 8051 MICROCONTROLLER:

The architecture of the 8051 microcontroller can be understood from the block diagram. It has Harward architecture with RISC (Reduced Instruction Set Computer) concept. The block diagram of 8051 microcontroller is shown in Fig 3. below1.It consists of an 8-bit ALU, one 8-bit PSW(Program Status Register), A and B registers , one 16-bit Program counter , one 16-bit Data pointer register(DPTR),128 bytes of RAM and 4kB of ROM and four parallel I/O ports each of 8-bit width.

ARCHITECTURE OF 8051

CODE segment is accessed using the program counter (PC) for opcode fetches and by DPTR for data. The external ROM is accessed when the EA(active low) pin is connected to ground or the contents of program counter exceeds 0FFFH.When the Internal ROM address is exceeded the 8051 automatically fetches the code bytes from the external program memory.

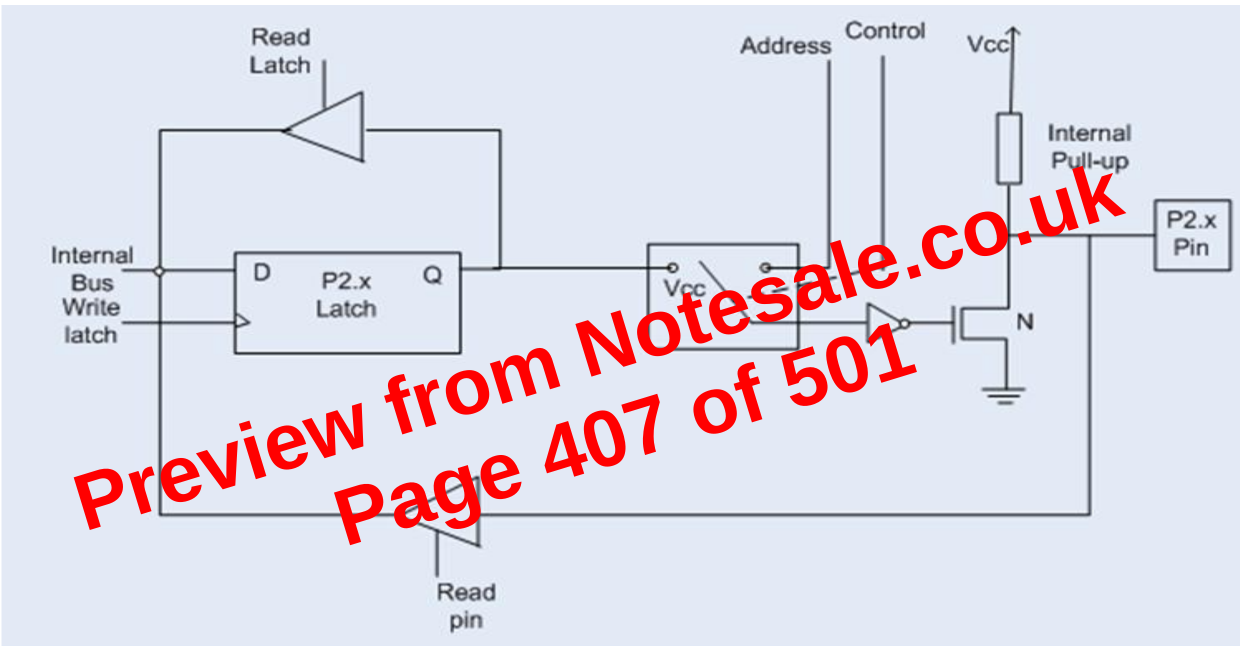


Interfacing Memories

Port 2 : Port 2 is also an eight bit parallel port. (pins 21- 28). It can be used as input or output port. As this port is provided with internal pull-up resistors it does not need any external pull-up resistors. Upon reset, Port 2 is configured as an output port. If the port is to be used as input port, all the port bits must be made high by sending FF to the port. For ex,

MOV A, #0FFH ; A=FF hex

MOV P_2, A ; make P_2 an input port by writing all 1's to it

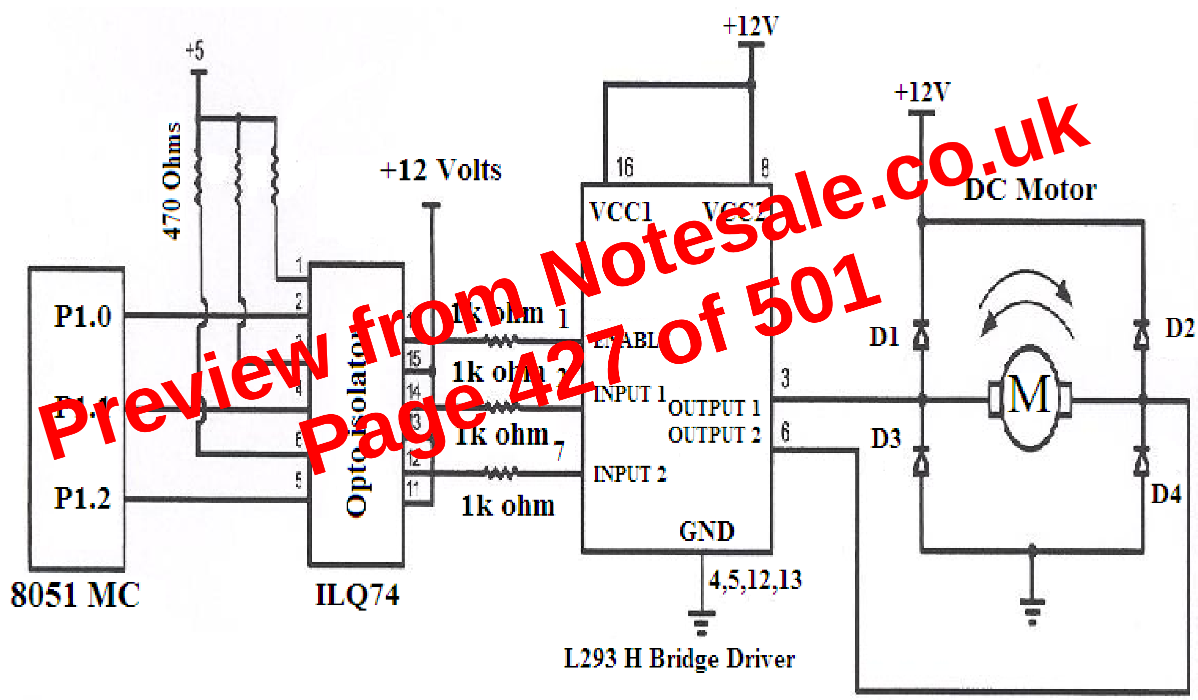


Dual role of port 2 : Port2 lines are also associated with the higher order address lines A8-A15. In systems based on the 8751, 8951, and DS5000, Port2 is used as simple I/O port.. But, in 8031-based systems, port 2 is used along with P0 to provide the 16-bit address for the external memory. Since an 8031 is capable of accessing 64K bytes of external memory, it needs a path for the 16 bits of the address. While P0 provides the lower 8 bits via A0-A7, it is the job of P2 to provide bits A8-A15 of the address. In other words, when 8031 is connected to external memory, Port 2 is used for the upper 8 bits of the 16 bit address, and it cannot be used for I/O operations.

PORT 3 : Port3 is also an 8-bit parallel port with dual function.(pins 10 to 17). The port pins can be used for I/O operations as well as for control operations. The details of these

INTERFACING DC MOTOR- 8051

A DC motor runs with the help of Direct Current. It produces torque by using both electricity and magnetic fields. The DC motor has rotor, stator, field magnet, brushes, shaft, commutator. The DC motor requires more current to produce initial torque than in running state.Interfacing the DC motor directly to 8051 microcontroller is not possible. Because the DC motor uses large current (200-300mA in small DC motors) to run. When this current flow into the 8051 microcontroller, the IC will get damaged. Therefore we use a driving circuit with an opto isolator and a L298 Dual H-Bridge driver. The opto-isolator provides additional protection to the microcontroller.



Continuous, sustained operation of the motor will cause the L293 Dual H-Bridge driver to overheat. So,a suitable heat sink must be used.

38. Why 8085 processor is called an 8 bit processor?
Because 8085 processor has 8 bit ALU (Arithmetic Logic Review).

39. Expand HCMOS?
High-density n- type Complimentary Metal Oxide Silicon field effect transistor.

40. What does microprocessor speed depend on?
The processing speed depends on DATA BUS WIDTH.

41. Give examples for Maskable interrupts?
RST 7.5, RST6.5, RST5.5 are Maskable interrupts

42. What is Tri-state logic?
Three Logic Levels are used and they are High, Low, High impedance state. The high and low are normal logic levels & high impedance state is electrical open circuit conditions. Tri-state logic has a third line called enable line.

43. Give an example of one address microprocessor?
8085 is one address microprocessor.

44. In what way interrupts are classified in 8085?
In 8085 the interrupts are classified as Hardware and Software interrupts.

45. What are Hardware interrupts?
TRAP, RST7.5, RST6.5, RST5.5, INTR.

46. What are Software interrupts?
RST0, RST1, RST2, RST3, RST4, RST5, RST6, RST7.

47. Which interrupt has the highest priority?
TRAP has the highest priority.

interrupt service.
If two or more interrupts go high at the same time, the 8085 will service them on priority basis. The TRAP has the highest priority followed b ye RST 7.5, RST 6.5, RST 5.5. The priority of interrupts in 8085 is shown in the table.

Interrupts	Priority
TRAP	1
RST 7.5	2
RST 6.5	3
RST 5.5	4
INTR	5

116. **What is a microcomputer?**
A computer that is designed using a microprocessor as its CPU is called microcomputer.

117. **What is the signal classification of 8085**
All the signals of 8085 can be classified into 6 groups
- Address bus
- Data bus
- Control and status signals
- Power supply and frequency signals
- Externally initiated signals
- Serial I/O ports

118. **What are operations performed on data in 8085**
The various operations performed are
- Store 8-bit data
- Perform arithmetic and logical operations
- Test for conditions
- Sequence the execution of instructions
- Store data temporarily during execution in the defined R/W memory locations called the stack

119. **Steps involved to fetch a byte in 8085**
i. The PC places the 16-bit memory address on the address bus
ii. The control unit sends the control signal RD to enable the memory chip
iii. The byte from the memory location is placed on the data bus
iv. The byte is placed in the instruction decoder of the microprocessor and the task is carried out according to the instruction

120. How many interrupts does 8085 have, mention them
The 8085 has 5 interrupt signals; they are INTR, RST7.5, RST6.5, RST5.5 and TRAP

to
override the declared type of a variable.

145.Explain about MODEL

This directive provides short cuts in defining segments. It initializes memory model before defining any segment. The memory model can be SMALL, MEDIUM, COMPACT or LARGE.

8086 :

1. **What is the purpose of segment registers in 8086?**

There are 4 segment registers present in 8086. They are

1. Code Segment (**CS)** register
2. Data Segment (**DS)** register
3. Stack Segment (**SS)** register
4. Extra Segment (**ES)** register

The **code segment** register gives the segment address of the current code segment.

The **data segment** register points out where the operands are stored in the memory.

The **stack segment** registers points out the address of the current stack.

The **Extra segment** registers points out where the large amount of data is stored in the memory.

2. **What do you mean by pipelining in an 8086 processor?[NOV/DEC 2006]**

49. Give example for Non-Maskable interrupts?
Trap is known as Non-Maskable interrupts, which is used in emergency condition.

50. What are the various segment registers in 8086?
Code, Data, Stack, Extra Segment registers in 8086.

51. Which Stack is used in 8086?
FIFO (First In First Out) stack is used in 8086.In this type of Stack the first stored information is retrieved first.

52. Where does CPU Enhanced mode originate from?
Intel's 80386 was the first 32-bit processor, and since the company had to backward-support the 8086. All the modern Intel-based processors run in the Enhanced mode, capable of switching between Real mode (just like the real 8086) and Protected mode, which is the current mode of operation.

53. How many bit combinations are there in a byte?
Byte contains 8 combinations of bits.

54. Have you studied buses? What types?
There are three types of buses.
Address bus: This is used to carry the Address to the memory to fetch either Instruction or Data.
Data bus : This is used to carry the Data from the memory.
Control bus : This is used to carry the Control signals like RD/WR, Select etc.

55. What is the Maximum clock frequency in 8086?
5 Mhz is the Maximum clock frequency in 8086.

56. What are the different functional units in 8086?
Bus Interface Unit and Execution unit, are the two different functional units in 8086.

57. What are the various segment registers in 8086?
Code, Data, Stack, Extra Segment registers in 8086.

d. Register indirect addressing mode
e. Indexed addressing mode
f. Register relative addressing mode
g. Based indexed addressing mode
h. Relative based indexed addressing mode
i. Intra segment direct mode
j. Intra segment indirect mode
k. Inter segment direct mode
l. Inter segment indirect mode

75. What are the types of instructions in instruction set of 8086?

- Data copy / Transfer instructions
- Arithmetic and Logical instructions
- Branch instructions
- Machine control instructions
- Flag manipulation instructions
- String instructions

76. List some functions of BIU?

- Sends address of the memory or I/O
- Fetches instructions from memory
- Reads data from port / memory
- Writes data into port / memory
- Supports instruction queuing
- Provides address relocation facility

77. Define assembler directives?

There are some instructions in the assembly language program which are not a part of processor instruction set. These are instructions to assembler and are referred as pseudo operations or assembler directives.

78. List some features of 8086?

- 16 bit microprocessor
- Has a 16 bit data bus, 20 bit address bus
- Can generate 16 bit I / O address
- Provides fourteen 16 bit registers
- Has multiplexed address and data bus
- Can operate in minimum and maximum mode

79. Define instruction pipelining?

101. List the commands that can be executed by 8237?

1. Clear First / Last Flip flop
2. Clear Mask Register
3. Master Clear Command

102. List the advantages of loosely coupled systems over the tightly couples systems?

a. More number of CPUs can be added in a loosely coupled system to improve the system performance.
b. System structure is modular and hence easy to maintain and trouble shoot.
c. Fault in a single module does not lead to a complete system break down.
d. It is more fault tolerant due to independent processing modules.
e. More suitable to parallel applications due to its modular organization.

103. **Explain PROC & ENDP**

PROC directive defines the procedures in the program. The procedure name must be unique. After PROC the term NEAR or FAR are used to specify the type of procedure.

Example FACT PROC FA R. ENDP is used along with PROC and defines the end of the procedure.

104. **Explain SEGMENT & ENDS**

An assembly program in .EXE format consists of one or more segments. The starts of these segments are defined by SEGMENT and the end of the segment is indicated by ENDS directive. Format Name SEGMENT

Name ENDS

105. **Explain TITLE & TYPE**

The TITLE directive helps to control the format of a listing of an assembled program. It causes a title for the program to print on line 2 of each page of the program listing.

Maximum 60 characters are allowed. Format TITLE text.

TYPE operator tells the assembler to determine the type of specified variable in bytes. For bytes the assembler gives a value 1, for word 2 & double word 4.

106. **Define SOP**

The segment override prefix allows the programmer to deviate from the default segment

Eg : MOV CS : [BX] , AL

107. **Define variable**

A variable is an identifier that is associated with the first byte of data item. In assembly language statement: COUNT DB 20H, COUNT is the variable.

Library files are collection of procedures that can be used in other programs. These procedures are assembled and compiled into a library file by the LIB program. The library file is invoked when a program is linked with linker program. when a library file is linked only the required procedures are copied into the program. Use of library files increase s/w reusability & reduce s/w development time.

113. **What are Macros**

Macro is a group of instruction. The macro assembler generates the code in the program each time where the macro is called. Macros are defined by MACRO & ENDM directives. Creating macro is similar to creating new opcodes that can be used in the program

```
INIT MACRO
MOV AX, data
MOV DS
MOV ES, AX
ENDM
```

114. **How do 8086 interrupts occur**

An 8086 interrupt can come from any of the following three sources

- External signals
- Special instructions in the program
- Condition produced by instruction

115. **What are the 8086 interrupt types**

Dedicated interrupts

- Type 0: Divide by zero interrupt
- Type 1: Single step interrupt
- Type 2:Non maskable interrupt
- Type 3: Breakpoint
- Type 4: Overflow interrupt

Software interrupts

- Type 0-255

116. **What is interrupt service routine**

Interrupt means to break the sequence of operation. While the CPU is executing a program an interrupt breaks the normal sequence of execution of instructions & diverts its execution to some other program. This program to which the control is transferred is called the interrupt service routine.

It is a word stored in a register (control register) used to control the operation of a program digital device.

168. What is the purpose of control word written to control register in 8255?

The control words written to control register specify an I/O function for each I.O port. The bit D_7 of the control word determines either the I/O function of the BSR function.

169. What is the size of ports in 8255?

Port-A : 8-bits
Port-B : 8-bits
Port-C_U : 4-bits
Port-C_L : 4-bits

170. What is interfacing?

An interface is a shared boundary between the devices which involves sharing information. Interfacing is the process of making two different systems communicate with each other.

171. What is memory mapping?

The assignment of memory addresses to various registers in a memory chip is called as memory mapping.

172. What is I/O mapping?

The assignment of addresses to various I/O devices in the memory chip is called as I/O mapping.

173. What is an USART?

USART stands for universal synchronous/Asynchronous Receiver/ Transmitter. It is a programmable communication interface that can communicate by using either synchronous or asynchronous serial data.

174. What is the use of 8251 chip?

8251 chip is mainly used as the asynchronous serial interface between the processor and the external equipment.

175. The 8279 is a programmable __________ interface.

Keyboard/Display

176. List the major components of the keyboard/Display interface.

a. Keyboard section
b. Scan section
c. Display section
d. CPU interface section

2.Display modes

- Left entry (Type writer mode)
- Right entry (Calculator mode)

39. What are the different functional units in 8279?

- CPU interface section
- Keyboard section
- Display section
- Scan section

40. What are the priority modes in 8259?

- **Fully nested mode**
- **Special fully nested mode c. Rotating Priority mode**
- **Special Masked mode e. Polled mode**

41. What is IMR(Interrupt mask register)?

IMR stores the masking bits of the interrupt lines to be masked. This register can be programmed by an operation command word (OCW).

42. What is priority resolver?

It determines the priorities of the bits set in the Interrupt request register (IRR), bit corresponding to the highest priority interrupt input is set in the ISR during INTA input.

43. What is the use of IRR?

The interrupt request register is used to store all the interrupt levels which are requesting the service. The eight interrupt inputs sets corresponding bits of the Interrupt Request Register upon the service request.

44. What is Interrupt service register(ISR)?

The interrupt service register stores all the levels that are currently being serviced.