8086 Architecture

1.Draw the pin diagram of 8086.

Ans. There would be two pin diagrams—one for MIN mode and the other for MAX mode of

8086. The pins that differ with each other in the two modes are from pin-24 to pin-31 (total 8 pins).

2. What is the technology used in 8086 pP?

Ans. It is manufactured using high performance metal-oxide semiconductor (HMOS)

technology. It has approximately 29,000 transistors and housed in a 40-pin DIP package.

3. Mention and explain the modes in which 8086 can operate.

Ans. 8086 pP can operate in two modes—MIN mode and MAX mode.

When MN/MX pin is high, it operates in MIN mode and when low, 8086 operates

in MAX mode. For a small system in which only one 8086 microprocessor is employed as a CPU,

the system operates in MIN mode (Uniprocessor). While if more than one 8086 operate

in a system then it is said to operate in MAX mode (Multiprocessor).
The bus controller IC (8288) generates the control signals in case of MAX mode, while

in MIN mode CPU issues the control signals required by memory and I/O devices.

4. Distinguish between the lower sixteen address lines from the upper four.

Ans. Both the lower sixteen address lines (AD0-AD15) and the upper four address lines(A16/ S3- A19 /S6) are multiplexed.

During T1 the lower sixteen lines carry address (Ao- A15), while during T2, T3 and T4 they carry data.

Similarly during Tx, the upper four lines carry address (A16- A19), while during T2, T3 and T4, they carry status signals.

5. In how many modes the minimum-mode signal can be divided?

Ans. In the MIN mode, the signals can be divided into the following basic groups: address/data

bus, status, control, interrupt and DMA.

7. Mention (a) the address capability of 8086 and (b) how many I/O lines can be accessed by 8086.

Ans. 8086 addresses via its A()-A19address lines. Hence it can address 220 = 1MB memory. Address lines Aq to A15 are used for accessing I/O's. Thus, 8086 can access 216 = 64 KB of I/O's.

8. What is meant by microarchitecture of 8086?

Ans. The individual building blocks of 8086 that, as a whole, implement the software and hardware architecture of 8086. Because of incorporation of additional features being necessitated by higher performance, the microarchitecture of 8086 or for that matter any microprocessor family, evolves over time.

9. Mention the conditions for which EU enters into WAIT mode.

Ans. There are three conditions that cause the EU to enter into WAIT state. These are:

- When an instruction requires the access to a memory location not in the queue.
- When a JUMP instruction is executed. In this case the current queue contents are aborted and the EU waits until the instructions at the jump address is fetched from memory.
- During execution of instruction which are very slow to execute. The instruction AAM (ASCII adjust for multiplication) requires 83 clock cycles for execution. For such a case, the BIU is made to wait till EU pulls one or two bytes from the queue before resuming the fetch cycle.

10. Mention the kind of operations possible with 8086.

Ans. It can perform bit, byte, word and block operations. Also multiplication and division operations can be performed by 8086.

11. Describe in brief the four segment registers.

Ans. The four segment registers are CS, DS, ES and SS—standing for code segment register, data segment register, extra segment register and stack segment register respectively.

When a particular memory is being read or written into, the corresponding memory address is determined by the content of one of these four segment registers in conjunction with their offset addresses.

The contents of these registers can be changed so that the program may jump from one active code segment to another one.

The use of these segment registers will be more apparent in memory segmentation schemes.

12. Discuss A16/S3—A19/S6 Signals of 8086.

Ans. These are time multiplexed signals. During T1? they represent A19 — A16 address lines. During I/O operations, these lines remain low. During T2-T4, they carry status signals.S4 and S3(during T2 to T4) identify the segment register employed for 20-bit physical address generation.Status signal S5 (during T2 to T4) represents interrupt enable status. This is updated at the beginning of each clock cycle.Status signal S6 remains low during T2 to T4.

13. Discuss the Reset pin of 8086.

Ans. Reset is an active high input signal and must be active for at least 4 CLK cycles to be accepted by 8086. This signal is internally synchronised and execution starts only after Reset returns to low value.

For proper initialisation, Reset pulse must not be applied before 50pS of "power on' of the circuit. During Reset state, all three buses are tristated and ALE and HLDA are driven low.

During resetting, all internal register contents are set to 0000 H, but CS is set to F000 H and IP to FFF0 H. Thus execution starts from physical address FFFF0 H. Thus EPROM in 8086 is interfaced so as to have the physical memory location forms FFFF0 H

to FFFFF H, i.e., at the end of the map.

14. Discuss the two pins (a) DT/ R and (b)

Ans. (a) DT/R is an output pin which decides the directions of data flow through the transreceivers (bidirectional buffers).

When the processor sends out data, this signal is 1 while when it receives data, the signal status is 0.

(b) DEN stands for data enable. It is an active low signal and indicates the availability of data over the address/data lines. This signal enables the transreceivers to separate data from the multiplexed address/data signal. It is active from the middle of T2 until the middle of T4.Both DT/ R and DEN are tristated during "hold acknowledge'

15. Explain the LOCK signal.

Ans. It is an active low output signal and is activated by LOCK prefix instruction and remains active until the completion of the next instruction. It floats to tri-state during hold acknowledge when LOCK signal is low, all interrupts get masked and HOLD request is not granted. LOCK signal is used by the processor to prevent other devices from accessing the system control bus. This symbol is used when CPU is executing some critical instructions and through this signal other devices are informed that they should not issue HOLD signal to 8086.

16. Explain the TEST signal.

Ans. It is an active low input signal. Normally the BUSY pin (output) of 8087 NDP is connected to the TEST input pin of 8086. When maths co-processor 8087 is busy executing some instructions, it pulls its BUSY signal high. Thus the TEST signal of 8086 is consequently high, and it (8086) is made to WAIT until the BUSY signal goes low. When 8087 completes its instruction executions, BUSY signal becomes low. Thus the TEST input

of 8086 becomes low also and then only 8086 goes in for execution of its program.

17. Discuss the Instruction Pointer (IP) of 8086.

Ans. Functionally, IP plays the part of Program Counter (PC) in 8085. But the difference is that IP holds the offset of the next word of the instruction code instead of the actual address (as in PC).

IP along with CS (code segment) register content provide the 20-bit physical (or real) address needed to access the memory. Thus CS:IP denotes the value of the memory address of the next code (to be fetched from memory).

Content of IP gets incremented by 2 because each time a word of code is fetched from memory.

18. Indicate the data types that can be handled by 8086 pP.

Ans. The types of data formats that can be handled by 8086 fall under the following categories:

- Unsigned or signed integer numbers—both byte-wide and word-wide.
- BCD numbers—both in packed or unpacked form.
- ASCII coded data. ASCII numbers are stored one number per byte.

19. Is direct memory to memory data transfer possible in 8086?

Ans. No, 8086 does not have provision for direct memory to memory data transfer.

For this to be implemented, AX is used as an intermediate stage of data. The source byte (from the memory) is moved into AX register with one instruction. The second instruction moves the content of AX into destination location (into another memory location). As example,

```
MOV AH, [SI]
MOV [DI], AH
```

Here, the first instruction moves the content of memory location, whose offset address remains in SI, into AH. The second instruction ensures that the content of AH

is moved into another memory location whose offset address is in DI.

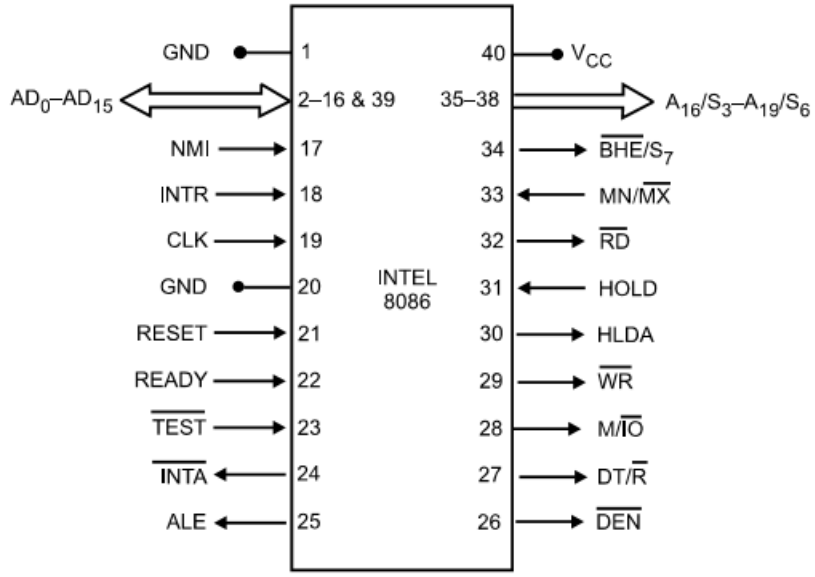
20. Can the data segment (DS) register be loaded directly by its address?

Ans. No, it cannot be done directly. Instead, AX is loaded with the initial address of the DS register and then it is transferred to DS register, as shown below:

MOV AX, DS ADDR: AX is loaded with initial address of DS register

MOV DS, AX: DS is loaded with AX, i.e., ultimately with DS ADDR

21.Draw Pin Diagram of 8086?



Signals of intel 8086 for minimum mode of operation

22.Architecture of 8086?

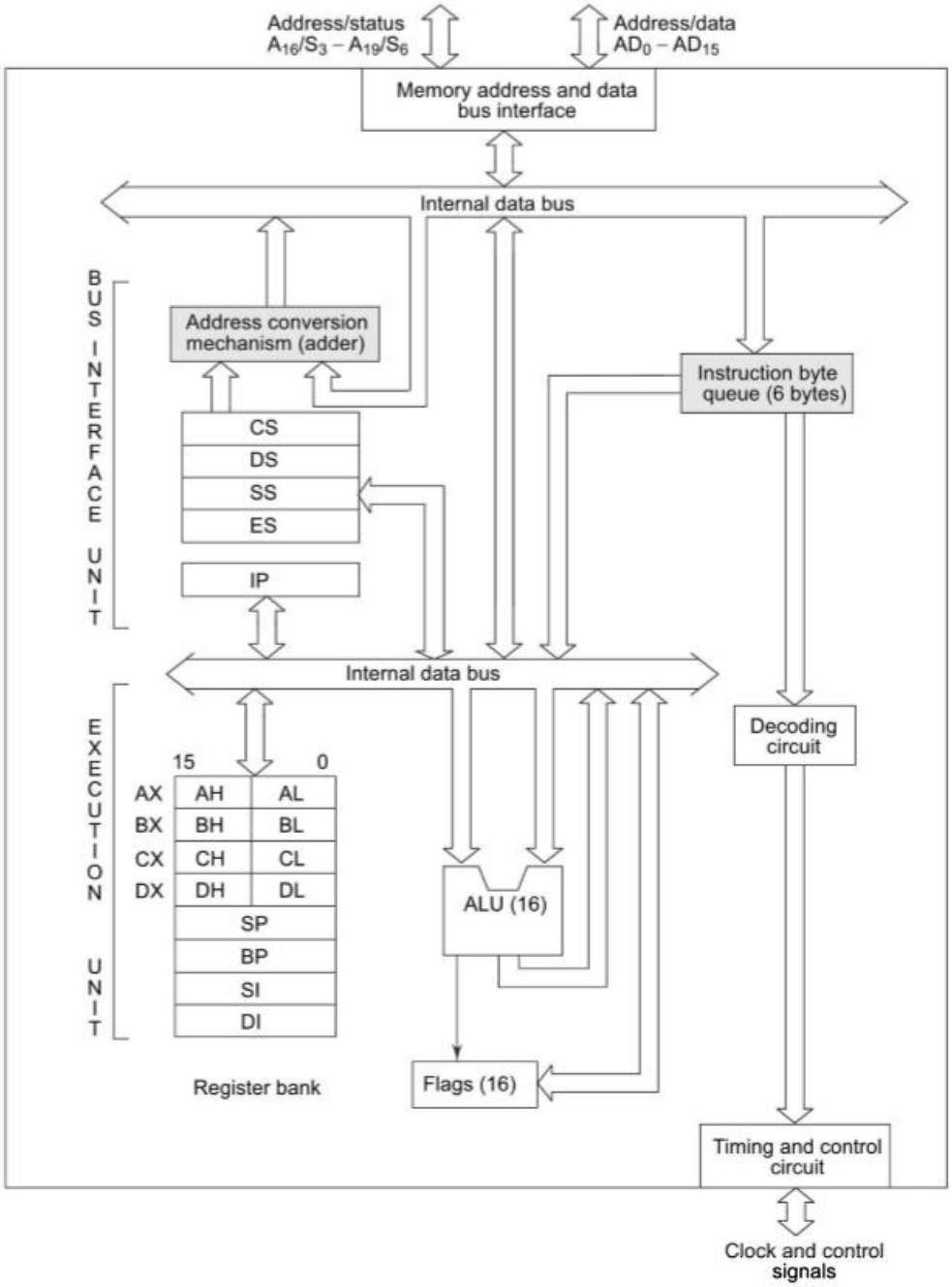
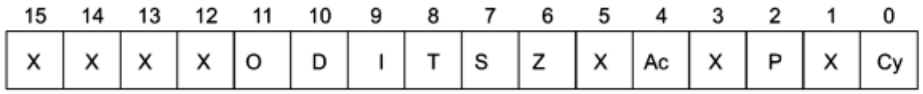


Fig. 1.2 *8086 Architecture*

23.Flags of 8086?



15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
X	X	X	X	O	D	I	T	S	Z	X	Ac	X	P	X	Cy

O — Overflow flag
D — Direction flag
I — Interrupt flag
T — Trap flag
S — Sign flag
Z — Zero flag
Ac — Auxiliary carry flag
P — Parity flag
Cy — Carry flag
X — Not used

Fig. 1.4 *Flag Register of 8086*

Memory Organization

1. Describe memory segmentation scheme of 8086. What is meant by currently active segments?

Ans. 1 MB memory of 8086 is partitioned into 16 segments—each segment is of 64 KB length. Out of these 16 segments, only 4 segments can be active at any given instant of time—these are code segment, stack segment, data segment and extra segment. The four memory segments that the CPU works with at any time are called currently active segments. Corresponding to these four segments, the registers used are Code Segment Register (CS), Data Segment Register (DS), Stack Segment Register (SS) and Extra Segment Register (ES) respectively.

Each of these four registers is 16-bits wide and user accessible—i.e., their contents can be changed by software.

The code segment contains the instruction codes of a program, while data, variables and constants are held in data segment. The stack segment is used to store interrupt and subroutine return addresses.

The extra segment contains the destination of data for certain string instructions.

Thus 64 KB are available for program storage (in CS) as well as for stack (in SS) while 128 KB of space can be utilized for data storage (in DS and ES).

One restriction on the base address (starting address) of a segment is that it must reside on a 16-byte address memory—examples being 00000 H, 00010 H or 00020 H, etc.

2. Mention the maximum size of memory that can be active for 8086.

Ans. The maximum size of active memory for 8086 is 256 KB. The break-up being

64 KB for program

64 KB for stack and

128 MB for data.

3. Why memory segmentation is done for 8086?

Ans. Memory segmentation, as implemented for 8086, gives rise to the following advantages:

• Although the address bus is 20-bits in width, memory segmentation allows one to work with registers having width 16-bits only.

• It allows instruction code, data, stack and portion of program to be more than 64 KB long by using more than one code, data, extra segment and stack segment.

• In a time-shared multitasking environment when the program moves over from one user's program to another, the CPU will simply have to reload the four segment registers with the segment starting addresses assigned to the current user's program.

• User's program (code) and data can be stored separately.

• Because the logical address range is from 0000 H to FFFF H, the same can be loaded at any place in the memory.

4. Discuss logical address, base segment address and physical address.

Ans. The logical address, also goes by the name of effective address or offset address (also known as offset), is contained in the 16-bit IP, BP, SP, BX, SI or DI.The 16-bit content of one of the four segment registers (CS, DS, ES, SS) is known as the base segment address.Offset and base segment addresses are combined to form a 20-bit physical address (also called real address) that is used to access the memory. This 20-bit physical address is put on the address bus (AD19 - AD0) by the BIU.

5. Although 8086 is a 16-bit pP, it deals with 8-bit memory. Why?

Ans. This is so for the following two reasons:

• It enables the microprocessor to work with both on bytes and words. This is very important because many I/O devices such as printers, terminals, modems etc, transfer ASCII coded data (7 or 8 bits).

• Quite a few of the operation codes of 8086 are single bytes while so many other

instructions are there which vary from 2 to 7 bytes. By working with byte-width memory, these varied opcodes can easily be handled.

6. Is the flat scheme of memory applied for 8086 pP?

Ans. No, the flat (or unsegmented) scheme of memory is not applied for 8086 pP, because the memory of the same is a segmented one. In flat scheme, the entire memory space is thought of as a single addressable memory unit.

The flat scheme can be applied for 8086 by initialising all the segment registers with identical (or same) base address. Then all memory operations will refer to the same memory space.

7. What is the maximum size of the memory that can be accessed by 8086?

Ans. The two status codes S4 and S3 together point to the segment register used for 20-bit physical address generation and can be examined by external circuitry to enable separate 1 MB address space for each of CS, ES, DS, and SS. This would enable memory address to be expanded to a maximum of 4MB for 8086 pP.

8.Physical memory organization

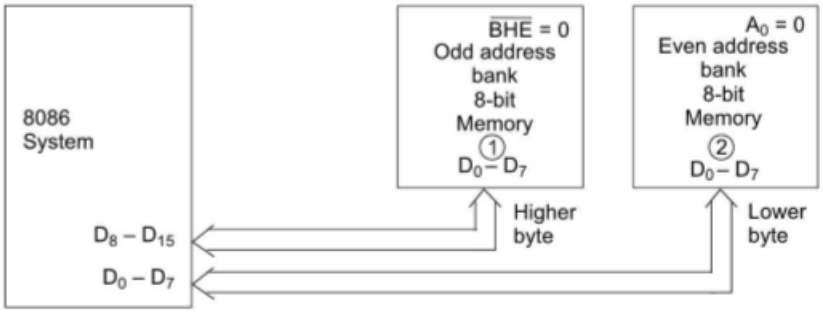


Fig. 1.7 *Physical Memory Organisation*

9.IO Addressing

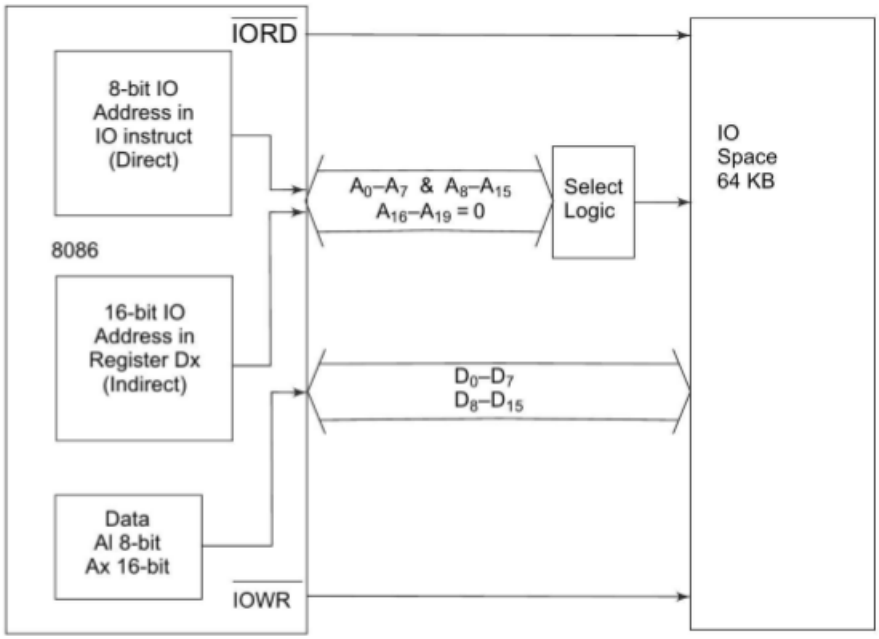


Fig. 1.9 *8086 IO Addressing*

Addressing Modes of 8086

1. What is meant by addressing mode?

Ans. An instruction consists of an opcode and an operand. The operand may reside in the accumulator, or in a general purpose register or in a memory location.

The manner in which an operand is specified (or referred to) in an instruction is called addressing mode.

2. Name the different addressing modes of 8086.

Ans. The following are the different addressing modes of 8086:

- Register operand addressing.
- Immediate operand addressing.
- Memory operand addressing.

3. Mention the different memory addressing modes.

Ans. The different memory addressing modes are:

- Direct Addressing
- Register Indirect Addressing
- Based Addressing
- Indexed Addressing
- Based Indexed Addressing and
- Based Indexed with displacement.

ADDITION OF 16-BIT NUMBERS

```
mov dx,2000h
mov ds,dx
mov cl,00h
mov bx,1500h
mov ax,[bx]
add ax,[bx+2]
mov [bx+4],ax
jnc skip
inc cl
skip: mov [bx+6] ,cl
hlt
```

SUBTRACTION OF 16-BIT NUMBERS

```
mov bx,2000h
mov ds,bx
mov cl,00h
mov ax,[1500h]
mov bx,[1502h]
sub ax,bx
mov [1504h],ax
jnb skip
inc cl
skip: mov [1506h],cl
hlt
```

MULTIPLICATION OF 8-BIT NUMBERS

```
assume cs: code,ds:data
 data segment
    mul1 db 47h
    mul2 db 76h
    res dw ?
data ends
code segment
    start:
    mov dx,data
    mov ds,dx
    mov al,mul1
    mov bl,mul2
    mul bl
    mov res,ax
code ends
end start
```

MULTIPLICATION OF 16-BIT

(BASE INDEX ADDRESS)

```
mov dx,2000h
mov ds,dx
mov bx,1000h
mov si,00h
mov ax,[bx][si]
mov cx,[bx][si+2]
mul cx
mov [bx][si+4],ax
mov [bx][si+6],dx
hlt
```

ADDITION OF 16-BIT ARRAY NUMBERS

```
org 1000h
mov bx,8000h
mov ds,bx
mov di,7200h
mov si,1000h
mov cx,10
mov dl,00h
xor ax,ax
li: add ax,[si]
jnc skip
inc dl
skip: inc si
    inc si
    loop li
    mov [si],ax
    mov [si+2],dl
    hlt
```

ADDITION OF 32-BIT NUMBERS

```
org 1000h
mov bx,8000h
mov ds,bx
mov si,1000h
mov ax,[si]
mov cx,[si+4]
mov bx,[si+2]
mov dx,[si+6]
add ax,cx
adc bx,dx
mov cl,00h
jnc skip
inc cl
skip: mov [si+8],ax
    mov [si+10],bx
    mov [si+12],cl
hlt
```

DIVISION OF 16-BIT BY 8-BIT NUMBERS

```
mov dx,2000h
mov ds,dx
mov bx,1000h
mov ax,[bx]
mov cl,[bx+2]
div cl
mov [bx+3],al
mov [bx+4],ah
hlt
```

SUBTRACTION OF 32-BIT NUMBERS

```
org 1000h
mov bx,5000h
mov ds,bx
mov si,1000h
mov ax,[si]
mov cx,[si+4]
mov bx,[si+2]
mov dx,[si+6]
sub ax,cx
sbb bx,dx
mov cl,00h
jnc skip
inc cl
skip: mov [si+8],ax
      mov [si+10],bx
      mov [si+12],cl
      hlt
```

DIVISION OF 32-BIT BY 16-BIT NUMBER

```
mov dx,2000h
mov ds,dx
mov si,1000h
mov ax,[si]
mov dx,[si+2]
mov cx,[si+4]
div cx
mov [si+6],ax
mov [si+8],dx
hlt
```

BINARY TO GRAY CODE CONVERTER

```
org 1000h
mov dx,2000h
mov ds,dx
mov bx,1000h
mov ax,[bx]
mov cx,ax
shr ax,01
xor ax,cx
mov [bx+2],ax
hlt
```

GRAY CODE TO BINARY CODE

```
org 1000h
mov dx,2000h
mov ds,dx
mov bx,1000h
mov ax,[bx]
mov cx,ax
shr ax,01
xor ax,cx
mov cx,ax
shr ax,02
xor ax,cx
mov cx,ax
shr ax,04
xor ax,cx
mov [bx+2],ax
hlt
```

UNPACKED BCD TO HEX CONVERSION

```
org ,1000h
mov dx,2000h
mov ds,dx
mov si,1000h
xor bx,bx
mov bl,[si]
mov al,[si+1]
mov cx,10
mul cx
add bx,ax
mov al,[si+2]
mov cx,100
mul cx
Add bx,ax
xor ax,ax
mov al,[si+3]
mov cx,1000
mul cx
add bx,ax
xor ax,ax
mov al,[si+4]
mov cx,1000
```

```
mul cx
add bx,ax
mov [si+5],bx
hlt
```

MULTIPLICATION OF 32-BIT BY 16-BIT NUMBER

```
mov dx,2000h
mov ds,dx
mov si,1000h
mov ax,[si]
mov bx,[si+4]
mul bx
mov [si+6],ax
mov [si+20],dx
mov ax,[si+2]
mul bx
mov [si+22],ax
mov [si+24],dx
xor cx,cx
mov ax,[si+20]
mov dx,[si+22]
add ax,dx
jnc skip
inc cx
skip : mov [si+8],ax
mov ax,[si+24]
add ax,cx
mov [si+10],ax
hlT
```

PACK TO UNPACK BCD

```
mov dx,2000h
mov ds,dx
mov si,1000h
mov ax,[si]
mov cl,10h
div cl
mov [si+2],ah
mov [si+3],al
hlt
```

UNPACK TO PACKED BCD

```
mov dx,2000h
mov ds,dx
mov si,1000h
mov bl,[si]
mov al,[si+1]
mov cl,10h
mul cl
add al,bl
mov [si+2],al
hlt
```

SORTING OF AN ARRAY OF 8-BIT NUMBERS

```
org 1000h
mov dx,2000h
mov ds,dx
mov cl,05h
dec cl
l1 : mov si,1000h
mov ch,05h
dec ch
l2 : mov al,[si]
inc si
cmp al,[si]
jc l3
xchg al,[si]
dec si
xchg al,[si]
inc si
l3 : dec ch
jnz l2
dec cl
jnz l1
hlt
```

ADDITION OF TWO MATRICES(2*2)

```
org 1000h
mov dx,2000h
mov ds,dx
mov si,1000h
mov al,[si]
mov bl,[si+16]
add ax,bx
mov [si+32],ax
xor ax,ax
inc si
mov al,[si]
mov bl,[si+16]
add ax,bx
mov [si+33],ax
xor ax,ax
inc si
mov al,[si]
mov bl,[si+16]
add ax,bx
mov [si+34],ax
xor ax,ax
```

```
inc si
mov al,[si]
mov bl,[si+16]
add ax,bx
mov [si+35],ax
hlt
```

UNPACKED BCD ADDITION

```
org 1000h
mov dx,2000h
mov ds,dx
mov si,1000h
mov al,[si]
add al,[si+16]
aaa
mov [si+32],al
mov [si+5],ah
xor ax,ax
mov al,[si+1]
add al,[si+5]
add al,[si+17]
aaa
mov [si+33],al
mov [si+34],ah
hlt
```

UNPACKED BCD MULTIPLICATION

```
mov dx,2000h
mov ds,dx
mov si,1000h
mov al,[si]
mov bl,[si+6]
mul bl
aam
mov [si+36],al
mov [si+5],ah
xor ax,ax
mov al,[si+1]
mul bl
aam
mov [si+6],al
mov [si+7],ah
xor ax,ax
mov al,[si+6]
add al,[si+5]
aaa
mov [si+33],al
shr ax,08h
```

add al,[si+7]

aaa

mov [si+34],al

mov [si+35],ah

hlt

MULTIPLICATION OF PACKED BCD NUMBER

```
mov dx,2000h
mov ds,dx
mov si,1000h
mov cx,12
mov ax,00
mov dl,00
l1 : add ax,[si]
daa
jnc skip
inc dl
skip : loop l1
mov [si+2],ax
mov [si+3],dl
hlt
```

MULTIPLICATION OF 2 DIGIT BY 1 DIGIT UNPACKED BCD USING AAM INSTRUCTION

```
mov dx,2000h
mov ds,dx
mov si,100h
mov al,[si]
mov bl,[si+2]
mul bl
aam
mov [si+10],ax
xor ax,ax
mov al,[si+1]
mul bl
aam
mov [si+12],ax
xor ax,ax
xor bx,bx
mov al,[si+11]
mov bl,[si+12]
mov cl,[si+13]
add al,bl
aaa
add cl,ah
```

```
mov [si+17],al
xor ax,ax
mov al,[si+10]
mov [si+16],al
mov [si+18],cl
hlt
```