NAME: SINGH SUDHAM DHARMENDRA

CLASS: CSE(AI & ML)

ROLL NO.: AIML51

SUBJECT: MICROPROCESSOR

TOPIC: EXPERIMENT NO. 1

DATE OF SUBMISSION: 11/02/2022

Exp: No.1

Aim: Study of addition and subtraction arithmetic operations through various addressing modes.

Apparatus: 8086 Emulator, PC.

Theory: Addressing modes refer to the different methods of addressing the operands.

Addressing modes of 8086 are as follows:-

Immediate addressing mode:- In this mode, the operand is specified in the instruction itself. Instructions are longer but the operands are easily identified.

Example: MOV CL, 12H This instruction moves 12 immediately into CL register. CL ← 12H

Register addressing mode:- In this mode, operands are specified using registers. This addressing mode is normally preferred because the instructions are compact and fastest executing of all instruction forms. Registers may be used as source operands, destination operands or both.

Example: MOV AX, BX This instruction copies the contents of BX register into AX register. AX ← BX

Direct memory addressing mode:- In this mode, address of the operand is directly specified in the instruction. Here only the offset address is specified, the segment being indicated by the instruction.

Example: MOV CL, [4321H] This instruction moves data from location 4321H in the data segment into CL. The physical address is calculated as DS * 10H + 4321 Assume DS = 5000H ∴PA = 50000 + 4321 = 54321H ∴CL ← [54321H]

Register based indirect addressing mode:- In this mode, the effective address of the memory may be taken directly from one of the base register or index register specified by instruction. If register is SI, DI and BX then DS is by default segment register. If BP is used, then SS is by default segment register.

Example: MOV CX, [BX] This instruction moves a word from the address pointed by BX and BX + 1 in data segment into CL and CH respectively. CL ← DS: [BX] and CH ← DS: [BX + 1] Physical address can be calculated as DS * 10H + BX.

Register relative addressing mode:- In this mode, the operand address is calculated using one of the base registers and an 8 bit or a 16 bit displacement.

Example: MOV CL, [BX + 04H] This instruction moves a byte from the address pointed by BX + 4 in data segment to CL. CL ← DS: [BX + 04H] Physical address can be calculated as DS * 10H + BX + 4H.

Base indexed addressing mode:- Here, operand address is calculated as base register plus an index register.

Example: MOV CL, [BX + SI] This instruction moves a byte from the address pointed by BX + SI in data segment to CL. CL ← DS: [BX + SI] Physical address can be calculated as DS * 10H + BX + SI.

Relative based indexed addressing mode:- In this mode, the address of the operand is calculated as the sum of base register, index register and 8 bit or 16 bit displacement.

Example: MOV CL, [BX + DI + 20] This instruction moves a byte from the address pointed by BX + DI + 20H in data segment to CL. CL ← DS: [BX + DI + 20H] Physical address can be calculated as DS * 10H + BX + DI + 20H.

Implied addressing mode:- In this mode, the operands are implied and are hence not specified in the instruction. Example: STC This sets the carry flag.

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

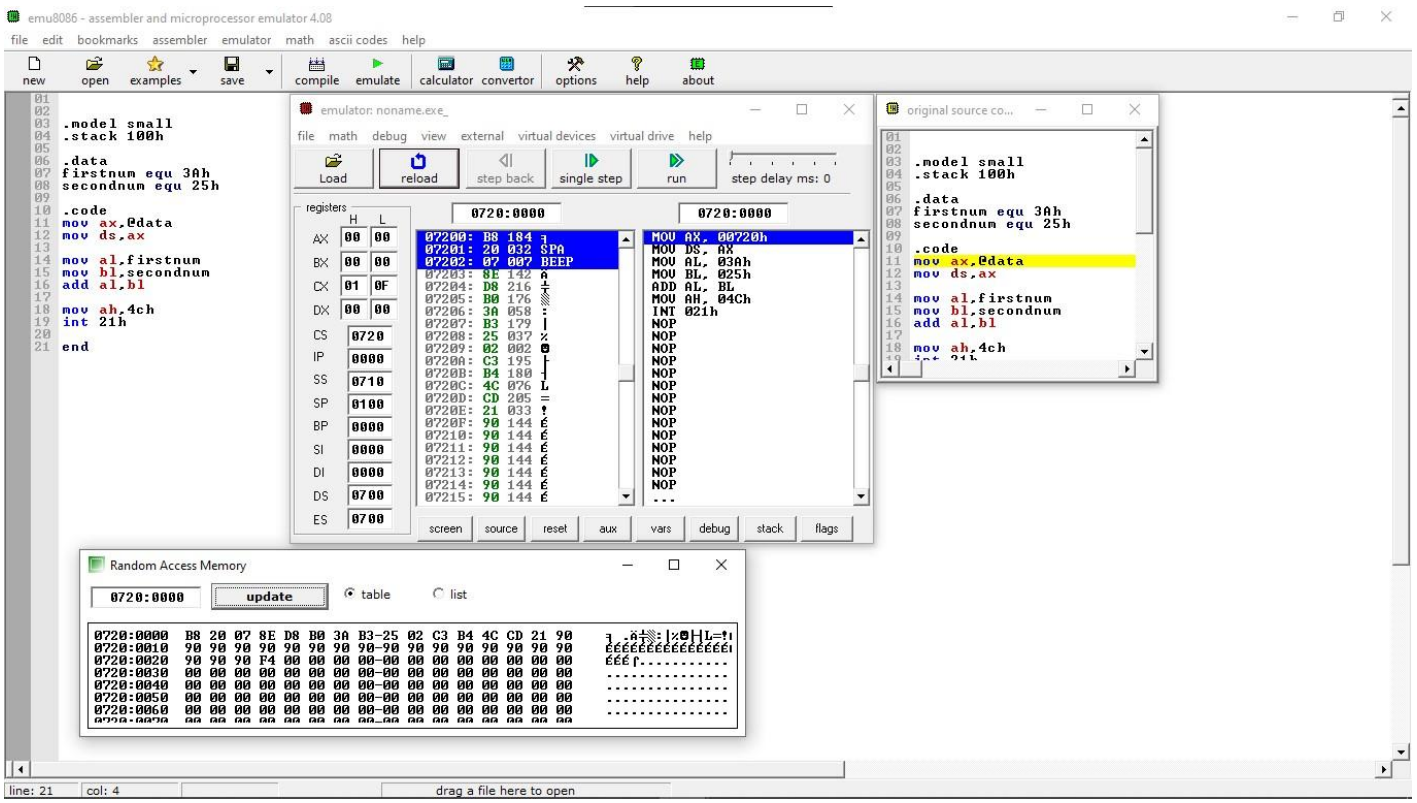
Example : JMP SHORT LABEL(8-bit)

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

1.1: Program to add two 8-bit using immediate addressing mode

```
;Addition of two 8-bits numbers and result 8-bit using immediate addressing mode
.model small
.stack 100H

.data
firstnum equ 3AH ; equ is an assembler directive which equate
secondnum equ 25H
result equ 00H

.code
MOV AX,@data
MOV DS,AX

MOV AL,firstnum
MOV BL,secondnum
ADD AL,BL
MOV result,AL

MOV AH,4CH
INT 21H

end ; assembler directive
```

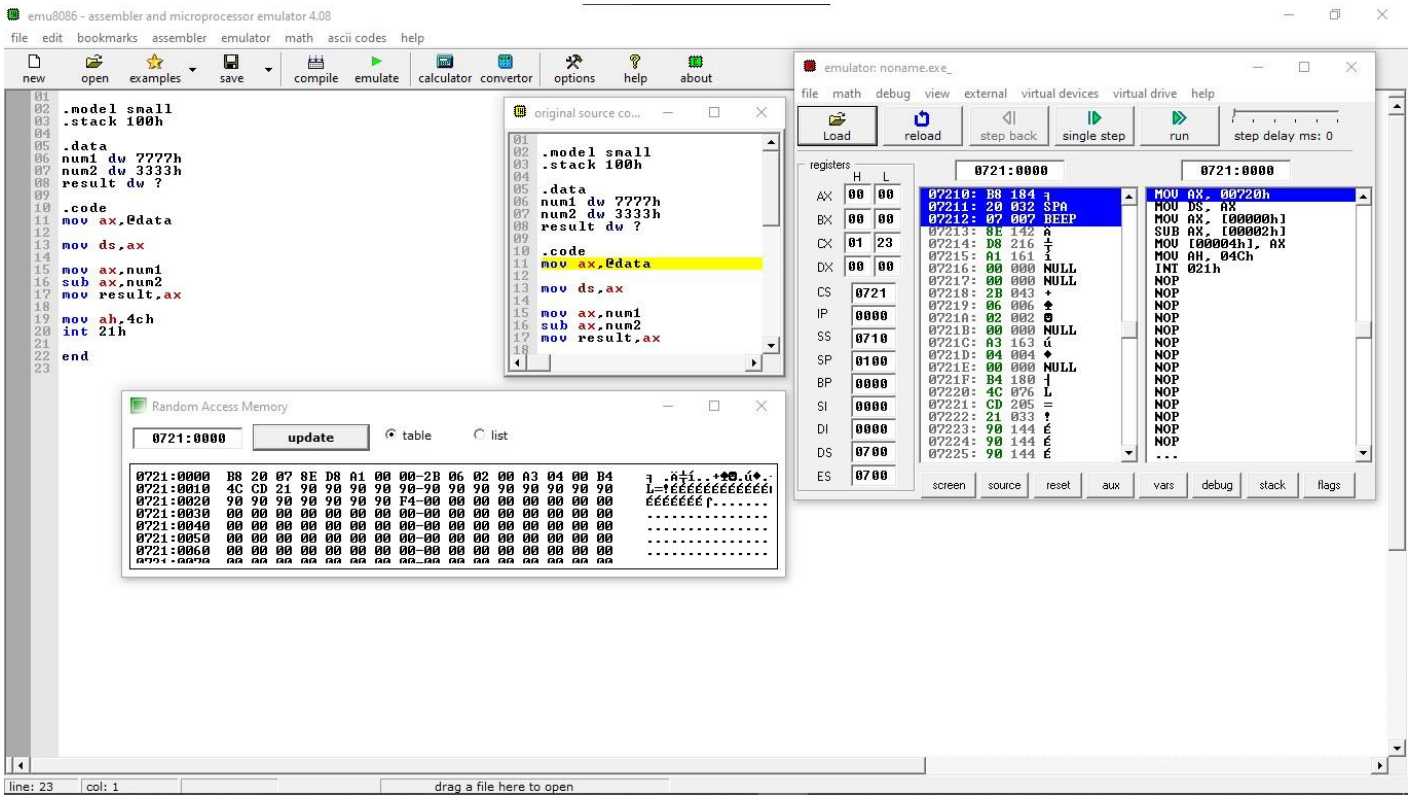
Result:


emu8086 - assembler and microprocessor emulator 4.08
file edit bookmarks assembler emulator math ascii codes help
new open examples save compile emulate calculator convertor options help about
.model small
.stack 100h
.data
firstnum equ 3Ah
secondnum equ 25h
.code
mov ax,@data
mov ds,ax
mov al,firstnum
mov bl,secondnum
add al,bl
mov ah,4ch
int 21h
end
emulator: noname.exe_
file math debug view external virtual devices virtual drive help
Load reload step back single step run step delay ms: 0
registers
AX BX CX DX CS IP SS SP BP SI DI DS ES
0720:0000
07200: B8 184 ¸
07201: 20 032 SPA
07202: 07 007 BEEP
07203: 8E 142 Ä
07204: D8 216 ╪
07205: B0 176 ░
07206: 3A 058 :
07207: B3 179 │
07208: 25 037 %
07209: 02 002 ☻
0720A: C3 195 ├
0720B: B4 180 ┤
0720C: 4C 076 L
0720D: CD 205 ═
0720E: 21 033 !
0720F: 90 144 É
07210: 90 144 É
07211: 90 144 É
07212: 90 144 É
07213: 90 144 É
07214: 90 144 É
07215: 90 144 É
MOV AX, 0720h
MOV DS, AX
MOV AL, 03Ah
MOV BL, 025h
ADD AL, BL
MOV AH, 04Ch
INT 021h
NOP
screen source reset aux vars debug stack flags
original source co...
Random Access Memory
0720:0000 update table list
0720:0000 B8 20 07 8E D8 B0 3A B3-25 02 C3 B4 4C CD 21 90
0720:0010 90 90 90 90 90 90 90 90-90 90 90 90 90 90 90 90
0720:0020 90 90 90 F4 00 00 00 00-00 00 00 00 00 00 00 00
0720:0030 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
0720:0040 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
0720:0050 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
0720:0060 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
line: 21 col: 4
drag a file here to open

1.2: Subtraction of two 16-bit numbers using direct addressing mode

```
;Program to subtract two 16-bit numbers using direct addressing mode

.model small
.stack 100H

.data
num1 dw 1234H ; dw is anassembler directive which is defining a word(16-bit)
num2 dw 5678H
result dw ?

.code
MOV AX,@data
MOV DS,AX

MOV AX,num1
SUB AX,num2
MOV result,AX

MOV AH,4CH
INT 21H

end
```

Result:


emu8086 - assembler and microprocessor emulator 4.08
file edit bookmarks assembler emulator math ascii codes help
new open examples save compile emulate calculator convertor options help about
.model small
.stack 100h
.data
num1 dw 7777h
num2 dw 3333h
result dw ?
.code
mov ax,@data
mov ds,ax
mov ax,num1
sub ax,num2
mov result,ax
mov ah,4ch
int 21h
end
original source co...
Random Access Memory
0721:0000
update
table
list
emulator: noname.exe_
file math debug view external virtual devices virtual drive help
Load
reload
step back
single step
run
step delay ms: 0
registers
AX 00 00
BX 00 00
CX 01 23
DX 00 00
CS 0721
IP 0000
SS 0710
SP 0100
BP 0000
SI 0000
DI 0000
DS 0700
ES 0700
screen source reset aux vars debug stack flags
line: 23
col: 1
drag a file here to open

NAME: SINGH SUDHAM DHARMENDRA

CLASS: CSE(AI & ML)

ROLL NO.: AIML51

SUBJECT: MICROPROCESSOR

TOPIC: EXPERIMENT NO. 2

DATE OF SUBMISSION: 20/02/2022

Exp: No.2

Aim: Study of multiplication and division arithmetic operations for various data lengths.

Apparatus: 8086 Emulator, PC.

Theory:

MUL (8 and 16 bits multipliers)-

>This is an unsigned multiplication instruction.

>It will multiply the operand with the accumulator.

>The result will be stored in the accumulator.

>8-bit accumulator inAL.

>16-bit accumulator in AX.

>32-bit accumulator is a combination of DX and Ax where DX is higher and AX lower.

Operand- Register, Memory Location

Eg.- 8 bit multiplication-

```
MUL BL      ;AX<-ALxBL
```

16 bit multiplication-

```
MUL BX      ;DX,AX<-AXxBX
```

IMUL-

>This is a signed multiplication instruction.

>It is used to multiply signed numbers.

>The rest is the same as MUL.

DIV (8 and 16 bits divisor)-

>This is an unsigned division instruction.

>It will divide the accumulator by the operand(divisor).

>The result will be stored in the accumulator.

Operand- Register, Memory location

Eg.- 16-bit / 8-bit division-

```
DIV BL     ;Will perform AXxBL
           ;AL gets the quotient, AH gets the remainder
```

Eg.- 32-bit / 16-bit division-

```
DIV BX     ;Will perform DX.AX / BX
           ;AX gets the quotient, DX gets the remainder
```

IDIV-

>This is a signed division instruction.

>It is used to divide signed numbers.

>The rest is the same as DIV.

2.1: Multiplication of two 8-bit numbers

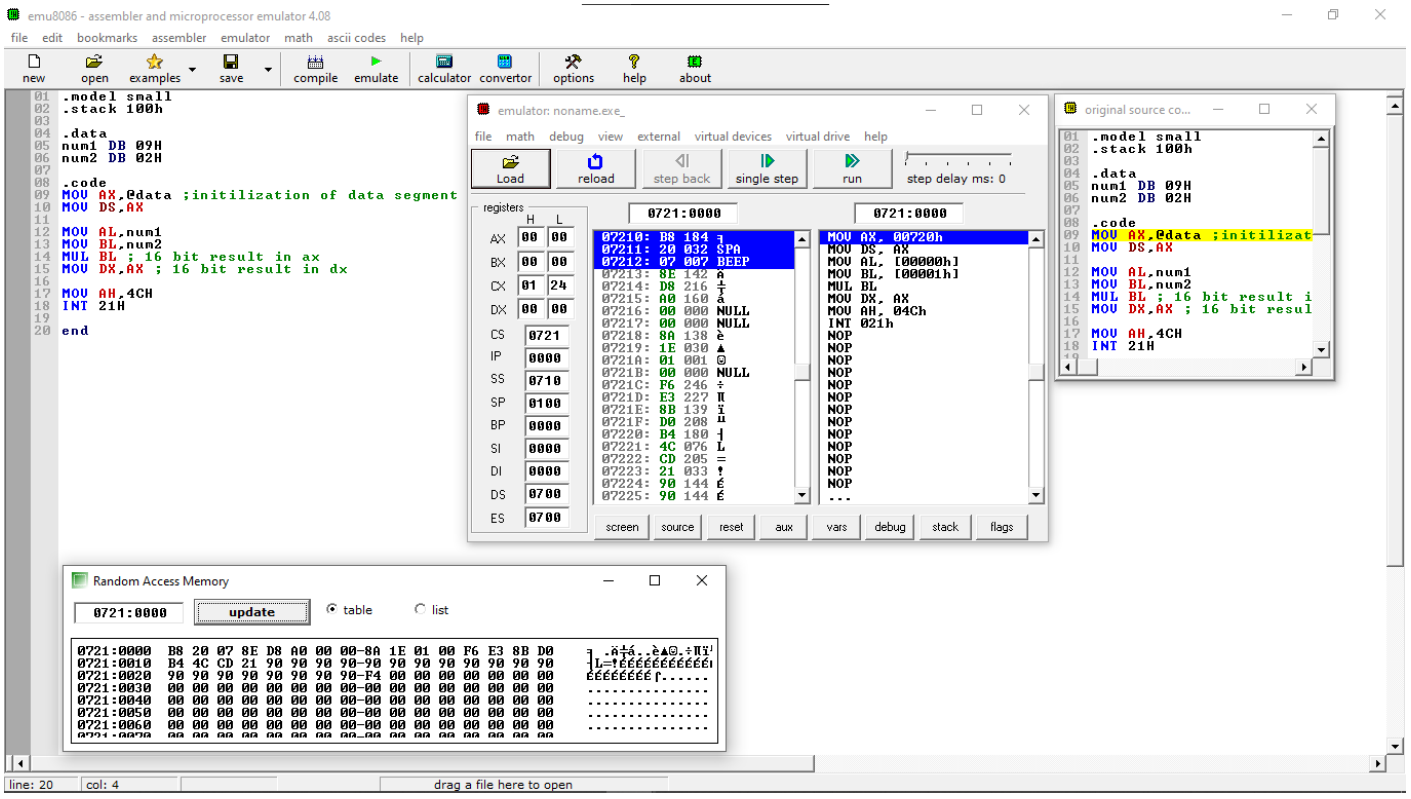
; Write assembly language program for 8086 microprocessor for multiplication of
; two 8 bits numbers, resulting in 16 bits with direct addressing modes.

```
.model small
.stack 100h

.data
num1 DB 09H
num2 DB 02H

.code
MOV AX,@data ;initialization of data segment
MOV DS,AX

MOV AL,num1
MOV BL,num2
MUL BL ; 16 bit result in ax
MOV DX,AX ; 16 bit result in dx

MOV AH,4CH
```

INT 21H

end

Result:

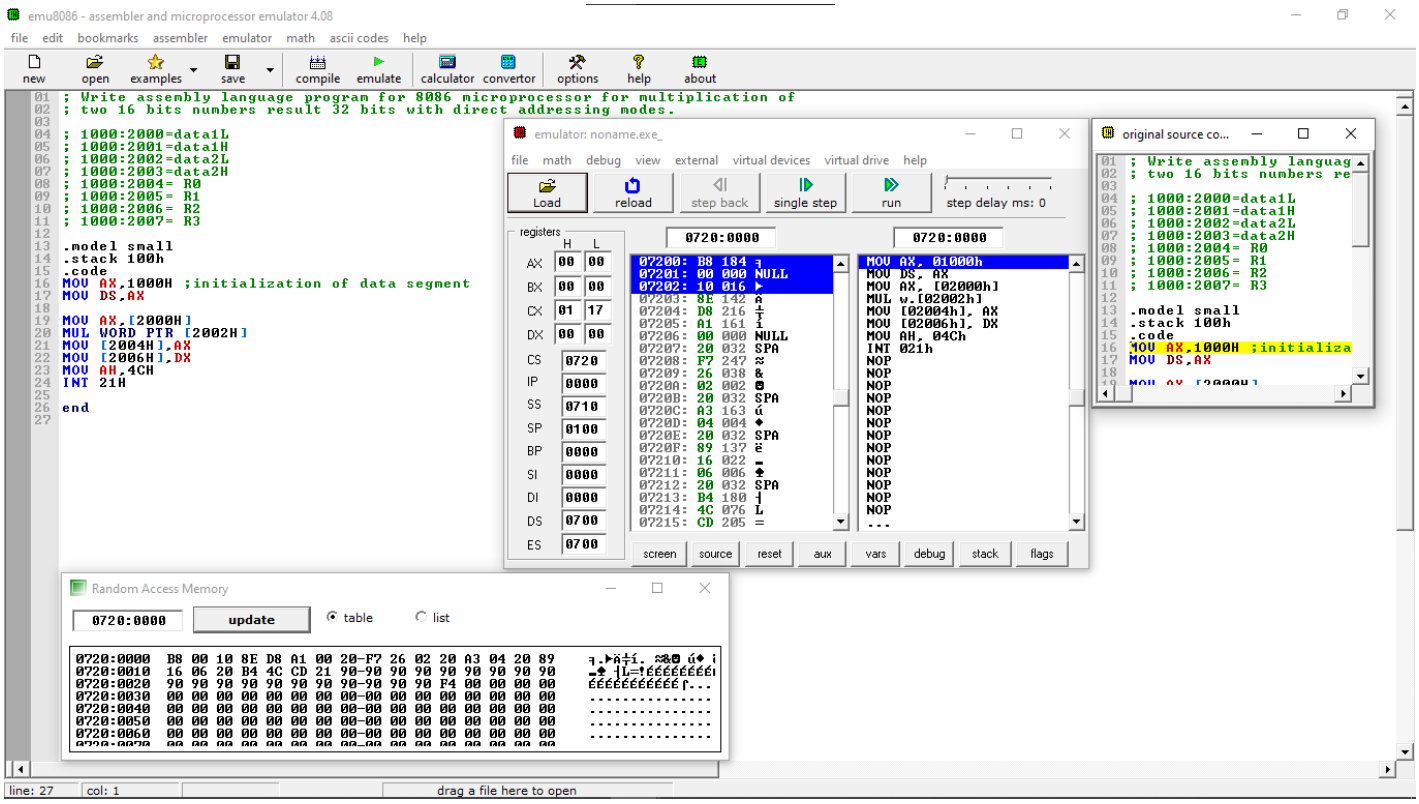


2.2: Multiplication of two 16-bit numbers located at specified memory addresses.

```
; Write assembly language program for 8086
microprocessor for multiplication of
; two 16 bits numbers result 32 bits with direct addressing
modes.

; 1000:2000=data1L
; 1000:2001=data1H
; 1000:2002=data2L
; 1000:2003=data2H
; 1000:2004= R0
; 1000:2005= R1
; 1000:2006= R2
; 1000:2007= R3

.model small
.stack 100h
.code
MOV AX,1000H ;initialization of data segment
MOV DS,AX

MOV AX,[2000H]
```

```
MUL WORD PTR [2002H]
MOV [2004H],AX
MOV [2006H],DX
MOV AH,4CH
INT 21H

end
```

Result:



2.3: Division a 16 bit number by 8 bit number

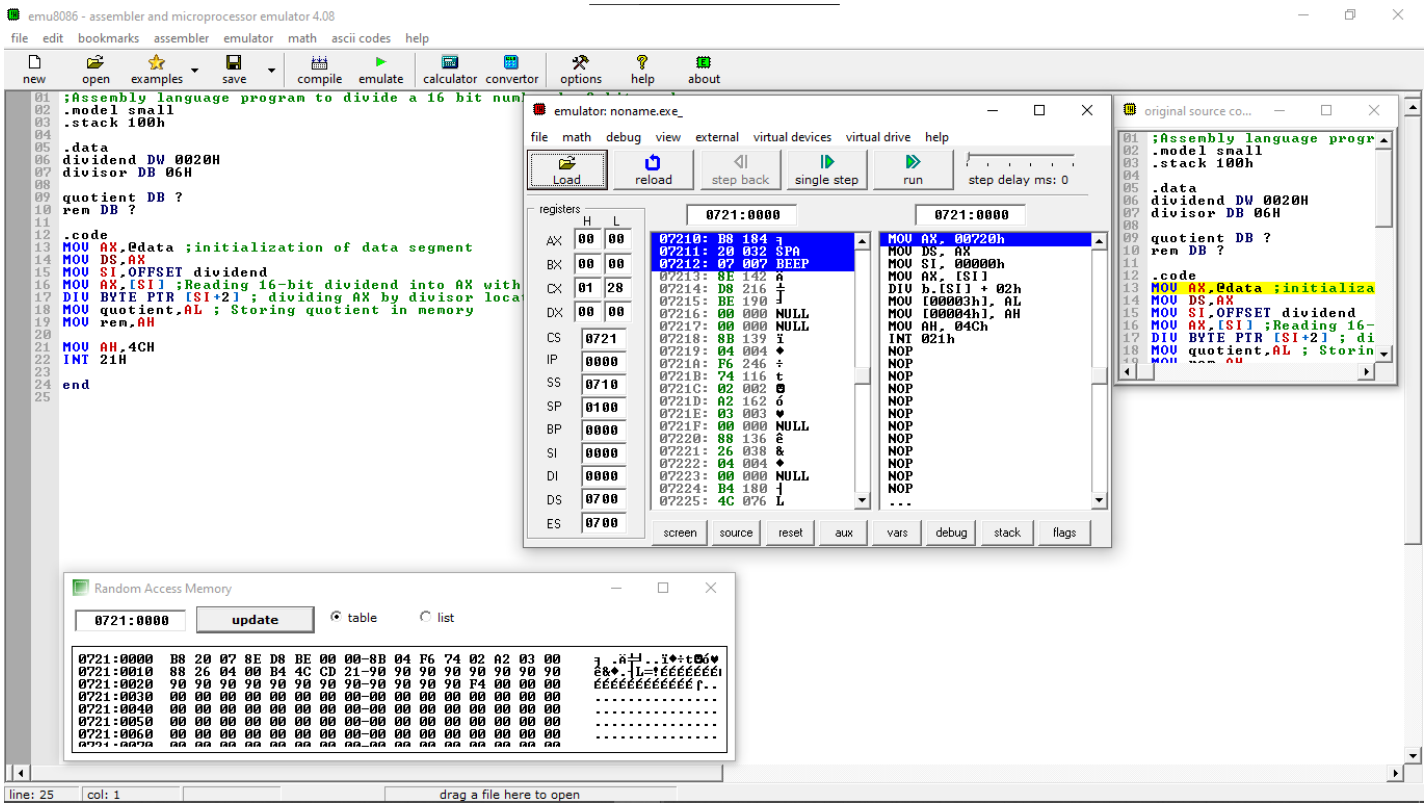
```
;Assembly language program to divide a 16 bit number by 8 bit number
.model small
.stack 100h

.data
dividend DW 0020H
divisor DB 06H

quotient DB ?
rem DB ?

.code
MOV AX,@data ;initialization of data segment
MOV DS,AX
MOV SI,OFFSET dividend
MOV AX,[SI] ;Reading 16-bit dividend into AX with base index addressing mode
DIV BYTE PTR [SI+2] ; dividing AX by divisor located at SI+2 in memory
MOV quotient,AL ; Storing quotient in memory
```

MOV rem,AH

MOV AH,4CH
INT 21H

end

Result:



2.4: Division a 32 bit number by 16 bit number

```
; Division a 32 bit number by 16 bit number
.model small
.stack 100h

.data

dividend_L DW 8888H
dividend_H DW 0022H
divisor dW 2222H
quotient DW ?
rem DW ?

.code
MOV AX,@data ;initialization of data segment
MOV DS,AX
MOV SI,OFFSET dividend_L

MOV AX,[SI] ; Moving LSB 16-bits of dividend from
memory to AX
MOV DX,[SI+2] ; Moving MSB 16-bits of dividend from
memory to DX
```

DIV WORD PTR [SI+4] ; Dividing DX:AX by divisor located at memory at SI+4

MOV quotient,AX ; Storing quotient in memory

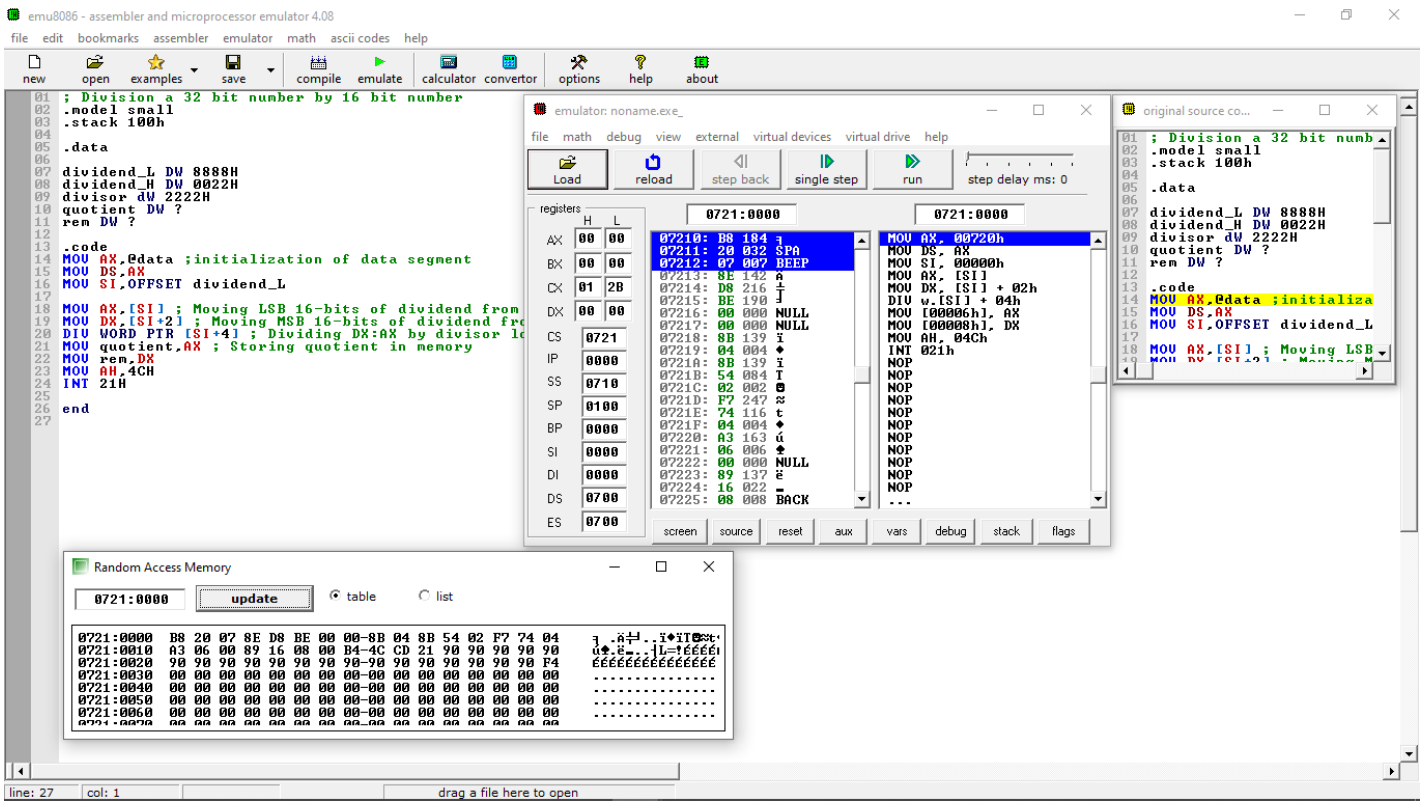
MOV rem,DX

MOV AH,4CH

INT 21H

end

Result:



Name :- Singh Sudham Dharmendra

Branch :- CSE (AI & ML)

Roll no :- AIML 51

Subject :- Microprocessor

Topic :- Experiment No. 3

Date of Submission :- 14/03/2022

Aim :- Assembly language programming for Microprocessor 8086 for finding the largest & smallest numbers of an array of numbers.

Apparatus :- 8086 Emmulator , PC.

Theory :- Explain the following instruction :-

① JNC & JC :-

JNC :- The JNC instruction transfers program controls to the specified address of the ~~memory~~ carry flag is 0. Otherwise, execution continues with the next construction . No flags are affected by this instruction.

JC :- In 8085 instruction set , we are having one memonik JC , a 16 which stands for "Jump if carry" and a 16 stands for any 16-bit address . This instruction is used to jump to the address a16 as provided in the instructions.

② LOOP label :-

The loop instruction assumes that the ECX register contains the loop count. When the loop instruction is executed , the ECX register is decremented and the control jumps to the target label , until the ECX register value . ie, the counter reaches the value zero.

3.1 Assembly language program to find the largest 16-bit numbers from the arrray of numbers stored in memory.

```
.model small
.stack 100h
.code
Mov AX, @data
Mov Ds, AX
LEA SI, words
MOV CX, cant
DEC CX
MOV AX, [SI]
up: CMP AX, [SI+2]
  JNC down
  Mov AX,[SI+2]
down: INC SI
    INC SI
    LOOP up
    Mov Largest, AX
    Mov AH, 4CH
    INT 21H
    END
```

Result: Screenshot attached

Wine Windows Program Loader

file edit bookmarks assembler help

NEW OPEN SAVE ASSEMBLE RUN

```
01 .MODEL SMALL
02 .stack 100H
03 .data
04 Num DW 2000H, 1234H, 3456H, 4567H, 9876H, 2222H, 2345H, 5555H, 6543H, 1111H
05 count EQU 0AH
06 largest DW 0000H
07 .code
08 mov AX, @data
09 mov DS, AX
10 Mov SI, OFFSET NUM
11 MOV CX, COUNT
12 DEC CX
13 MOV AX, [SI]
14 BACK: CMP AX, [SI+2]
15 JNC next
16 mov AX, [SI+2]
17 NEXT: INC SI
18 INC SI
19 LOOP BACK
20 MOV LARGEST, AX
21 MOV AH, 4CH
22 INT 21H
23 END
24
```

original source code

```
06 largest DW 0000H
07 .code
08 mov AX, @data
09 mov DS, AX
10 Mov SI, OFFSET NUM
11 MOV CX, COUNT
12 DEC CX
13 MOV AX, [SI]
14 BACK: CMP AX, [SI+2]
15 JNC next
16 mov AX, [SI+2]
17 NEXT: INC SI
18 INC SI
19 LOOP BACK
20 MOV LARGEST, AX
21 MOV AH, 4CH
22 INT 21H
```

emulator: noname.exe_

file debug view virtual devices virtual drive help

LOAD | reload | step back | single step | run | 0 | step delay ms: 0

registers

	H	L
AX	00	00
BX	00	00
CX	01	41
DX	00	00
CS	0722	
IP	0000	
SS	0710	
SP	0100	
BP	0000	
SI	0000	
DI	0000	
DS	0700	
ES	0700	

0722:0000

```
07220: B8 184 ,
07221: 20 032 SPA
07222: 07 007 BEEP
07223: 8E 142 Ž
07224: D8 216 Ø
07225: BE 190 ¾
07226: 00 000 NULL
07227: 00 000 NULL
07228: B9 185 ¹
07229: 0A 010 NEWL
0722A: 00 000 NULL
0722B: 49 073 I
0722C: 8B 139 ‹
0722D: 04 004 ♦
0722E: 3B 059 ;
0722F: 44 068 D
07230: 02 002 ☻
```

0722:0000

```
MOV AX, 00720h
MOV DS, AX
MOV SI, 00000h
MOV CX, 0000Ah
DEC CX
MOV AX, [SI]
CMP AX, [SI] + 02h
JNB 016h
MOV AX, [SI] + 02h
INC SI
INC SI
LOOP 0Eh
MOV [00014h], AX
MOV AH, 04Ch
INT 021h
NOP
...
```

screen source reset aux vars debug stack flags

Random Access Memory

0720:0000 | update | table | list

```
0720:0000  00 20 34 12 56 34 67 45-76 98 22 22 45 23 55 55   . 4♦V4gEv˜""E#UU
0720:0010  43 65 11 11 00 00 00 00-00 00 00 00 00 00 00 00   Ce♦♦............
0720:0020  B8 20 07 8E D8 BE 00 00-B9 0A 00 49 8B 04 3B 44   ¸ .ŽØ¾..¹..I♦;D
0720:0030  02 73 03 8B 44 02 46 46-E2 F4 A3 14 00 B4 4C CD   ♦s♦‹D♦FFâô£♦.´LÍ
0720:0040  21 90 90 90 90 90 90 90-90 90 90 90 90 90 90 90   !♦♦♦♦♦♦♦♦♦♦♦♦♦♦♦♦
0720:0050  90 90 90 90 90 F4 00 00-00 00 00 00 00 00 00 00   ♦♦♦♦♦ô..........
0720:0060  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
0720:0070  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
```

3.2 Assembly language program to find smallest 16-bit number from the array of numbers stored in memory.

```
.model
.stack 100h
.code
Mov AX, @data
Mov DS, AX
LEA SI, words
MOV CX, cant
DEC CX
MOV AX, [SI]
up, CMP AX, [SI+2]
JC down
Mov AX, [SI+2]
down: INC SI
      INC SI
      Loop UP
Mov largest, AX
Mov AH, 4CH
INT 21H
end.
```

Result:- Screenshot attached.

Wine Windows Program Loader

file edit bookmarks assembler help

NEW OPEN SAVE ASSEMBLE RUN

```
01 .MODEL SMALL
02 .stack 100H
03 .data
04 Num DW 2000H, 1234H, 3456H, 4567H, 9876H, 2222H, 2345H, 5555H, 6543H, 1111H
05 count EQU 0AH
06 largest DW 0000H
07 .code
08 mov AX, @data
09 mov DS, AX
10 Mov SI, OFFSET NUM
11 MOV CX, COUNT
12 DEC CX
13 MOV AX, [SI]
14 BACK: CMP AX, [SI+2]
15 JC next
16 mov AX, [SI+2]
17 NEXT: INC SI
18 INC SI
19 LOOP BACK
20 MOV LARGEST, AX
21 MOV AH, 4CH
22 INT 21H
23 END
24
```

original source code

```
01 .MODEL SMALL
02 .stack 100H
03 .data
04 Num DW 2000H, 1234H, 3456H, 4
05 count EQU 0AH
06 largest DW 0000H
07 .code
08 mov AX, @data
09 mov DS, AX
10 Mov SI, OFFSET NUM
11 MOV CX, COUNT
12 DEC CX
13 MOV AX, [SI]
14 BACK: CMP AX, [SI+2]
15 JC next
16 mov AX, [SI+2]
17 NEXT: INC SI
```

emulator: noname.exe_

file debug view virtual devices virtual drive help

LOAD reload step back single step run step delay ms: 0

registers

	H	L
AX	07	20
BX	00	00
CX	01	41
DX	00	00
CS	0722	
IP	0003	
SS	0710	
SP	0100	
BP	0000	
SI	0000	
DI	0000	
DS	0700	
ES	0700	

0722:0003

```
07220: B8 184 ¸
07221: 20 032 SPA
07222: 07 007 BEEP
07223: 8E 142 Ž
07224: D8 216 Ø
07225: BE 190 ¾
07226: 00 000 NULL
07227: 00 000 NULL
07228: B9 185 ¹
07229: 0A 010 NEWL
0722A: 00 000 NULL
0722B: 49 073 I
0722C: 8B 139 ‹
0722D: 04 004 ♦
0722E: 3B 059 ;
0722F: 44 068 D
07230: 02 002 ☻
```

0722:0003

```
MOV AX, 00720h
MOV DS, AX
MOV SI, 00000h
MOV CX, 0000Ah
DEC CX
MOV AX, [SI]
CMP AX, [SI] + 02h
JB 016h
MOV AX, [SI] + 02h
INC SI
INC SI
LOOP 0Eh
MOV [00014h], AX
MOV AH, 04Ch
INT 021h
NOP
...
```

screen source reset aux vars debug stack flags

0720:0000 update table list

```
0720:0000  00 20 34 12 56 34 67 45-76 98 22 22 45 23 55 55   . 4♦V4gEv˜""E#UU
0720:0010  43 65 11 11 00 00 00 00-00 00 00 00 00 00 00 00   Ce♦♦............
0720:0020  B8 20 07 8E D8 BE 00 00-B9 0A 00 49 8B 04 3B 44   ¸ .ŽØ¾..¹..I♦;D
0720:0030  02 72 03 8B 44 02 46 46-E2 F4 A3 14 00 B4 4C CD   ☻r♥‹D☻FFâô£¶.´LÍ
0720:0040  21 90 90 90 90 90 90 90-90 90 90 90 90 90 90 90   !♫♫♫♫♫♫♫♫♫♫♫♫♫♫♫♫
0720:0050  90 90 90 90 90 F4 00 00-00 00 00 00 00 00 00 00   ♫♫♫♫♫ô..........
0720:0060  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
0720:0070  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
```

Name:- Singh Sudham Dharmendra.

Branch:- CSE (AI & ML)

Roll no:- AIML-51

subject:- Microprocessor.

Topic:- Experiment No:- 4

Date of submission:- 21/03/2022

Aim:- Assembly language programming for Microprocessor 8086 to sort the numbers in ascending and descending order.

Apparatus:- 8086 Emulator, PC.

Theory:-

① Org - Originate

This directives indicates the assembler, that the next data/code should origin from the specified locations.

eg:- ORG 1000H; the code/data following should be stored at 1000H.

② DB - define Byte

this directives is used to intialize a variable as a byte (8-bit) variable. Hence the memory space allocates to such a variable is one byte.

eg:- X DB 5

This statement creates a byte variable named as X & the value calculated is 5.

③ DW - Define word or word

This directive is used to initialize a variable as a word (16-bit) variable. Hence the memory space allocated to such a variable is two bytes.

for eg:- X DW 5000H

This statement creates a word variable named as X & the value calculated is 5000H

④ EQU - Equate.

EQU stands for EQUAL or EQUATE

It is used to assign a value to a variable or constant.

for eg. X EQU 10.

(5) EXTERN:

The extern directive provides the assembler with a name that is not defined in the current assembly. EXTERN is very similar to IMPORT, except that name is not imported if reference to it is found in the current assembly.

(6) END:

This is placed at the END of source and it acts as the last statement of a program.

(7) ASSUME:-

This directive is used to give another name for segment registers. So as to make it easily to give the name.

example:- ASSUME CS: code, DS: Data, SS: stack.

in above eg: DS- data indicates the assembler to associate the name of data segment with DS register.

Similarly CS- Code indicates the assembler to associate the value of code segment with is register

(8) ENDS:

ENDS directive is used to indicate the end of segment.

eg:- Data ENDS.

4.1 Assembly language program to sort the 16-bit numbers located in memory in ascending order.

Code:-

```
.model small
.stack 100H
.data
num DW 1010H, 2000H, 9000H, 12000H, 0110H, 5000H, 2930H,
3912H, 3000H, 1500H
count DW 0AH
.code
MOV AX, @data
MOV DS, AX
MOV DX, Count
DEC DX
OUTER-LOOP-LEA SI, NUM
MOV CX, COUNT
DEC CX
INNER-LOOP: MOV AX, [SI]
CMP AX, [SI+2]
JC next
XCHG AX, [SI+2]
MOV [SI], AX
next: INC SI
      INC SI
LOOP INNER-LOOP
DEC DX
JNZ OUTER-LOOP
MOV AH, 4CH
INT 21H
END.
```

Activities emu8086 Mon 3:49 PM

edit: Z:\home\computer\Desktop\EMU8086\MySource\MP_EXP-4.asm

file edit bookmarks assembler help

NEW OPEN SAVE ASSEMBLE RUN

```
01 .model small
02 .stack 100h
03 .data
04 num DW 1010H,2000H,9000H,1200H,0110H,5000H,2930H,3912H,3000H,1500H
05 count DW 0AH
06 .code
07 MOV AX,@data
08 MOV DS,AX
09 MOV DX,count
10 DEC DX
11 OUTER-LOOP:LEA SI,Num
12 MOV CX,count
13 DEC CX
14 INNER-LOOP:MOV AX,[SI]
15 CMP AX,[SI+2]
16 JC next
17 XCHG AX,[SI+2]
18 MOV [SI],AX
19 next:INC SI
20      INC SI
21 LOOP INNER-LOOP
22 DEC DX
23 JNZ OUTER-LOOP
24 MOV AH,4CH
25 INT 21H
26 END
27
```

emulator: MP_EXP-4.exe

file debug view virtual devices virtual drive help

LOAD | reload | step back | single step | run | 0 step delay ms: 0

registers

	H	L
AX	4C	00
BX	00	00
CX	00	00
DX	00	00
CS	F400	
IP	0204	
SS	0710	
SP	00FA	
BP	0000	
SI	0012	
DI	0000	
DS	0720	
ES	0700	

F400:0204

```
F4200: FF 255 RES
F4201: FF 255 RES
F4202: CD 205 Í
F4203: 21 033 !
F4204: CF 207 Ï
F4205: 00 000 NULL
F4206: 00 000 NULL
F4207: 00 000 NULL
F4208: 00 000 NULL
F4209: 00 000 NULL
F420A: 00 000 NULL
F420B: 00 000 NULL
F420C: 00 000 NULL
F420D: 00 000 NULL
F420E: 00 000 NULL
F420F: 00 000 NULL
F4210: 00 000 NULL
F4211: 00 000 NULL
F4212: 00 000 NULL
F4213: 00 000 NULL
```

F400:0204

```
BIOS DI
INT 021h
IRET
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
...
```

screen source reset aux vars debug stack flags

original source code

```
15 CMP AX,[SI+2]
16 JC next
17 XCHG AX,[SI+2]
18 MOV [SI],AX
19 next:INC SI
20 INC SI
21 LOOP INNER-LOOP
22 DEC DX
23 JNZ OUTER-LOOP
24 MOV AH,4CH
25 INT 21H
```

Random Access Memory

0720:0000 update table list

```
0720:0000  10 01 10 10 00 12 00 15-00 20 30 29 00 30 12 39   □□□□.□.□
0720:0010  00 50 00 90 0A 00 00 00-00 00 00 00 00 00 00 00   .P.□........
0720:0020  B8 20 07 8E D8 8B 16 14-00 4A BE 00 00 8B 0E 14   ¸ .ŽØ‹□J¾..‹..
0720:0030  00 49 8B 04 3B 44 02 72-05 87 44 02 89 04 46 46   .I‹□;D□r□‡D.‰.FF
0720:0040  E2 F0 4A 75 E5 B4 4C CD-21 90 90 90 90 90 90 90   âðJuå´LÍ!.......
0720:0050  90 90 90 90 90 90 90 90-90 90 90 90 90 F4 00 00   □□□□□□□□□□□□□□□□
0720:0060  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
0720:0070  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
```

4.2 : Assembly language program to sort the 16-bit numbers located in memory in descending order.

code :-

```
• model small
• stack 100H
• data
num DW 1010H, 2000H, 9000H, 1200H, 0110H, 5000H, 2930H, 3912H,
    3000H, 1500H.
Count DW 0AH
.code
MOV AX, @data.
 MOV DS, AX
 MOV DX, count
 DEC DX
 Outer - LOOP : LEA SI, NUM
 MOV CX, COUNT
 DEC CX
 INNER - LOOP: MOV AX, [SI]
 CMP AX, [SI+2]
 JNC Next
 XCHG AX, [SI+2]
 MOV [SI], AX
 next: INC SI
       INC SI
 LOOP INNER - LOOP
 DEC DX
 JNZ OUTER - LOOP
 MOV AH, 4CH
 INT 21H
 END
```

Activities emu8086 Mon 3:50 PM

edit: Z:\home\computer\Desktop\EMU8086\MySource\MP_EXP-4.asm

file edit bookmarks assembler help

NEW OPEN SAVE ASSEMBLE RUN

```
01 .model small
02 .stack 100h
03 .data
04 num DW 1010H,2000H,9000H,1200H,0110H,5000H,2930H,3912H,3000H,1500H
05 count DW 0AH
06 .code
07 MOV AX,@data
08 MOV DS,AX
09 MOV DX,count
10 DEC DX
11 OUTER-LOOP:LEA SI,Num
12 MOV CX,count
13 DEC CX
14 INNER-LOOP:MOV AX,[SI]
15 CMP AX,[SI+2]
16 JNC next
17 XCHG AX,[SI+2]
18 MOV [SI],AX
19 next:INC SI
20      INC SI
21 LOOP INNER-LOOP
22 DEC DX
23 JNZ OUTER-LOOP
24 MOV AH,4CH
25 INT 21H
26 END
27
```

emulator: MP_EXP-4.exe

file debug view virtual devices virtual drive help

LOAD | reload | step back | single step | run | 0 step delay ms: 0

registers

	H	L
AX	4C	10
BX	00	00
CX	00	00
DX	00	00
CS	F400	
IP	0204	
SS	0710	
SP	00FA	
BP	0000	
SI	0012	
DI	0000	
DS	0720	
ES	0700	

F400:0204

```
F4200: FF 255 RES
F4201: FF 255 RES
F4202: CD 205 Í
F4203: 21 033 !
F4204: CF 207 Ï
F4205: 00 000 NULL
F4206: 00 000 NULL
F4207: 00 000 NULL
F4208: 00 000 NULL
F4209: 00 000 NULL
F420A: 00 000 NULL
F420B: 00 000 NULL
F420C: 00 000 NULL
F420D: 00 000 NULL
F420E: 00 000 NULL
F420F: 00 000 NULL
F4210: 00 000 NULL
F4211: 00 000 NULL
F4212: 00 000 NULL
F4213: 00 000 NULL
```

F400:0204

```
BIOS DI
INT 021h
IRET
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
...
```

screen source reset aux vars debug stack flags

original source code

```
15 CMP AX,[SI+2]
16 JNC next
17 XCHG AX,[SI+2]
18 MOV [SI],AX
19 next:INC SI
20 INC SI
21 LOOP INNER-LOOP
22 DEC DX
23 JNZ OUTER-LOOP
24 MOV AH,4CH
25 INT 21H
```

Random Access Memory

0720:0000 update table list

```
0720:0000  00 90 00 50 12 39 00 30-30 29 00 20 00 15 00 12   .□.P□9.00)
0720:0010  10 10 10 01 0A 00 00 00-00 00 00 00 00 00 00 00   □□□□.....
0720:0020  B8 20 07 8E D8 8B 16 14-00 4A BE 00 00 8B 0E 14   ¸ .ŽØ‹□J
0720:0030  00 49 8B 04 3B 44 02 73-05 87 44 02 89 04 46 46   .I‹□;D□s□
0720:0040  E2 F0 4A 75 E5 B4 4C CD-21 90 90 90 90 90 90 90   âðJuå´LÍ
0720:0050  90 90 90 90 90 90 90 90-90 90 90 90 90 F4 00 00   □□□□□□□□
0720:0060  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
0720:0070  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
```

Name :- Singh Sudhanu Dhanendra.

Branch :- CSE (AI & ML)

ROLL NO:- AIML - 51

Subject:- MICROPROCESSOR

Topic :- EXPERIMEN NO. 5

Date of Submission :- 21/03/2022.

Aim: Assembly language programming for microprocessor 8086 to convert BCD number to Hexadecimal number & Hexadecimal number to BCD number.

Apparatus: 8086 Emulator, PC

Theory:- Explain the following instruction, DDA with example

DAA ⇒ Decimal adjustment for addition.....

mnemonic ⇒ DAA

algorithm - If lower nibble of AL > 9 or AF = 1 then,

AL = AL + 06H, AF = 1

If AL > 9 FH or CF = 1 then,

AL = AL + 60H, CF = 1

Flags - It changes AF, CF, DF, ZF & SF

Address mode - implied addressing mode.

operatn - AL ← sum in adjustment to packed BCD format

- this instruction is used to make sure that result of adding two packed BCD numbers is adjusted to be a valid BCD number.
- It operates only on AL register
- If number in the lower nibble of AL register ~~of AL~~ After addition is greater the 9 or if the auxiliary carry flag is set add 6.
- If the weight nibble of AM is greater than 9 or if the carry flag is set then add 60H.

Example:- If AL = 59H valid BCD, BL = 34H valid BCD

ADD AL, BL

	0101	1001
+	0011	0100
	1000	1101
	8	0

AL = 80H invalid BC after addition of AL & BL

DAA →

	1000	1101
	0000	0110
	1001	0011
	9	3

AL = 93H BCD

5.1: Assembly language programming for microprocessor 8086 to convert ~~BDC~~ BCD numbers to Hexadecimal numbers.

Code:-

```
• model small
• stack 100H
• data
BCDNum DB 12
HEXNum DB ?
• code
MOV AX, @data
MOV DS, AX
MOV SI, OFFSET BCDNUM
MOV DI, OFFSET HEXNUM
MOV BL, [SI]
AND BL, 0FH
MOV AL, [SI]
AND AL, 0F0H
MOV CL, 04H
ROR AL, CL
MOV DL, 0AH
MUL DL
ADD AL, BL
MOV [DI], AX
MOV AH, 4CH
INT 21H
end.
```

edit: Z:\home\computer\Desktop\EMU8086\MySource\MP-4.asm

file edit bookmarks assembler help

NEW | OPEN | SAVE | ASSEMBLE | RUN

```
01 ;to convert BCD to HEX
02 .model small
03 .stack 100H
04 .data
05 BCDNUM DB 12
06 HEXNUM DB ?
07 .code
08 MOV AX,@DATA
09 MOV DS,AX
10 MOV SI,OFFSET BCDNUM
11 MOV DI,OFFSET HEXNUM
12 MOV BL,[SI]
13 AND BL,0FH
14 MOV AL,[SI]
15 AND AL,0F0H
16 MOV CL,04H
17 ROR AL,CL
18 MOV DL,0AH
19 MUL DL  ;AUTOMATICALLY MULTIPLY W
20 ADD AL,BL
21 MOV [DI],AL
22 MOV AH,4CH
23 INT 21H
24 END
```

emulator: MP-4.exe

file debug view virtual devices virtual drive help

LOAD | reload | step back | single step | run | 0 | step delay ms: 0

registers

	H	L
AX	4C	0C
BX	00	0C
CX	01	04
DX	00	0A
CS	F400	
IP	0204	
SS	0710	
SP	00FA	
BP	0000	
SI	0000	
DI	0001	
DS	0720	
ES	0700	

0723:0002 | 0723:0002

```
07232: CD 205 I   | INT 021h
07233: 21 033 !   | NOP
07234: 90 144 □   | NOP
07235: 90 144 □   | NOP
07236: 90 144 □   | NOP
07237: 90 144 □   | NOP
07238: 90 144 □   | NOP
07239: 90 144 □   | NOP
0723A: 90 144 □   | NOP
0723B: 90 144 □   | NOP
0723C: 90 144 □   | NOP
0723D: 90 144 □   | NOP
0723E: 90 144 □   | NOP
0723F: 90 144 □   | NOP
07240: 90 144 □   | NOP
07241: 90 144 □   | NOP
07242: 90 144 □   | NOP
07243: 90 144 □   | NOP
07244: 90 144 □   | NOP
07245: 90 144 □   | ...
```

screen | source | reset | aux | vars | debug | stack | flags

original source code

```
01 ;to convert BCD to HEX
02 .model small
03 .stack 100H
04 .data
05 BCDNUM DB 12
06 HEXNUM DB ?
07 .code
08 MOV AX,@DATA
09 MOV DS,AX
10 MOV SI,OFFSET BCDNUM
11 MOV DI,OFFSET HEXNUM
```

Random Access Memory

0720;000 | update | table | list

```
0720:0000  0C 0C 00 00 00 00 00 00-00 00 00 00 00 00 00 00   □□..........
0720:0010  B8 20 07 8E D8 BE 00 00-BF 01 00 8A 1C 80 E3 0F   ¸ ŽØ¾.¿...Š.€ã.
0720:0020  8A 04 24 F0 B1 04 D2 C8-B2 0A F6 E2 02 C3 88 05   Š□$ð±□ÒÈ².öâ.Ã^.
0720:0030  B4 4C CD 21 90 90 90 90-90 90 90 90 90 90 90 90   ´LÍ!□□□□□□□□□□□□
0720:0040  90 90 90 90 90 90 90 90-F4 00 00 00 00 00 00 00   □□□□□□□□ô.......
0720:0050  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
0720:0060  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
0720:0070  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
```

5.2: Assembly language Programming for microprocessor 8086 to Convert Hexa decimal number to BCD numbers.

Code:

```
.model small
.stack 100H
.data
hexnum db 0ah
bcdnum dw ?
.code
mov ax, @data.
mov ds, ax
mov si, offset hexnum
mov di , offset bcdnum
mov dl, 00h
mov al, 00h
mov bl, [SI]
back: ADD al, 01H
DAA
JNC next
INC dl
next: DEC bl
JNZ back
MOV [DI], al
MOV [DI+1], dl
MOV ah, 4CH
INT 21H
end
```

Conclusion :- Thus we have ~~spoo~~ simulated the conversion of BCD number to Hexadecimal number and Hexadecimal number to BCD number.

edit: Z:\home\computer\Desktop\EMU8086\MySource\MP-4.asm

file edit bookmarks assembler help

NEW | OPEN | SAVE | ASSEMBLE | RUN

```
01 ;to convert HEX TO BCD
02 .model small
03 .stack 100H
04 .data
05 HEXNUM DB 0AH
06 BCDNUM DW ?
07 .code
08 MOV AX,@DATA
09 MOV DS,AX
10 MOV SI,OFFSET HEXNUM
11 MOV DI,OFFSET BCDNUM
12 MOV DL,00H ;TO COLLECT CARRY
13 MOV AL,00H
14 MOV BL,[SI]
15 BACK:ADD AL,01H
16 DAA ;DECIMAL ADJUST AFTER ADDITION
17 JNC NEXT
18 INC DL
19 NEXT:DEC BL
20 JNZ BACK
21 MOV [DI],AL
22 MOV [DI+1],DL
23 MOV AH,4CH
24 INT 21H
25 END
```

emulator: MP-4.exe

file debug view virtual devices virtual drive help

LOAD | reload | step back | single step | run | 0 | step delay ms: 0

registers

	H	L
AX	4C	10
BX	00	00
CX	01	35
DX	00	00
CS	F400	
IP	0204	
SS	0710	
SP	00FA	
BP	0000	
SI	0000	
DI	0001	
DS	0720	
ES	0700	

F400:0204

```
F4200: FF 255 RES
F4201: FF 255 RES
F4202: CD 205 Í
F4203: 21 033 !
F4204: CF 207 Ï
F4205: 00 000 NULL
F4206: 00 000 NULL
F4207: 00 000 NULL
F4208: 00 000 NULL
F4209: 00 000 NULL
F420A: 00 000 NULL
F420B: 00 000 NULL
F420C: 00 000 NULL
F420D: 00 000 NULL
F420E: 00 000 NULL
F420F: 00 000 NULL
F4210: 00 000 NULL
F4211: 00 000 NULL
F4212: 00 000 NULL
F4213: 00 000 NULL
```

F400:0204

```
BIOS DI
INT 021h
IRET
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
...
```

screen | source | reset | aux | vars | debug | stack | flags

original source code

```
14 MOV BL,[SI]
15 BACK:ADD AL,01H
16 DAA ;DECIMAL ADJUST AFTE
17 JNC NEXT
18 INC DL
19 NEXT:DEC BL
20 JNZ BACK
21 MOV [DI],AL
22 MOV [DI+1],DL
23 MOV AH,4CH
24 INT 21H
```

Random Access Memory

0720:000 | update | table | list

```
0720:0000  0A 10 00 00 00 00 00 00-00 00 00 00 00 00 00 00    .□..............
0720:0010  B8 20 07 8E D8 BE 00 00-BF 01 00 B2 00 B0 00 8A    ¸ .ŽØ¾..¿..²..°..Š
0720:0020  1C 04 01 27 73 02 FE C2-FE CB 75 F5 88 05 88 55    □□□'s□þÂþËuõˆ□ˆU
0720:0030  01 B4 4C CD 21 90 90 90-90 90 90 90 90 90 90 90    □´LÍ!□□□□□□□□□□
0720:0040  90 90 90 90 90 90 90 90-90 F4 00 00 00 00 00 00    □□□□□□□□□ô......
0720:0050  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00    ................
0720:0060  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00    ................
0720:0070  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00    ................
```

Name :- Singh Sudham Dharmendra.

Branch :- CSE(AI & ML)

Roll NO :- AIML 51

Subject :- Micro processor.

Topic :- Assignment No :- (1).

Date of Submission :- 07/03/2022.

Q.1) Explain the instruction pipelining features of 8086. Give its advantages & its disadvantages.

→ (1) Fetching the next instruction while the current instruction, excutes is called pipelining.

(2) While the EU is decoding an instruction or executing an instruction, which does not require use of buses, the BIU fetches upto six instruction bytes for the following instructions.

(3) The BIU stores these prefetched bytes in a first-in-first-out register set called a queue.

(4) A new word gets fetched from memory whenever 2 bytes are vacant in queue.

Advantages:-

(1) The obvious advantage of Pipelining is that it increases efficiency

(2) Time is no longer spent on fetching as it happens alongside execution.

(3)

Disadvantages:-

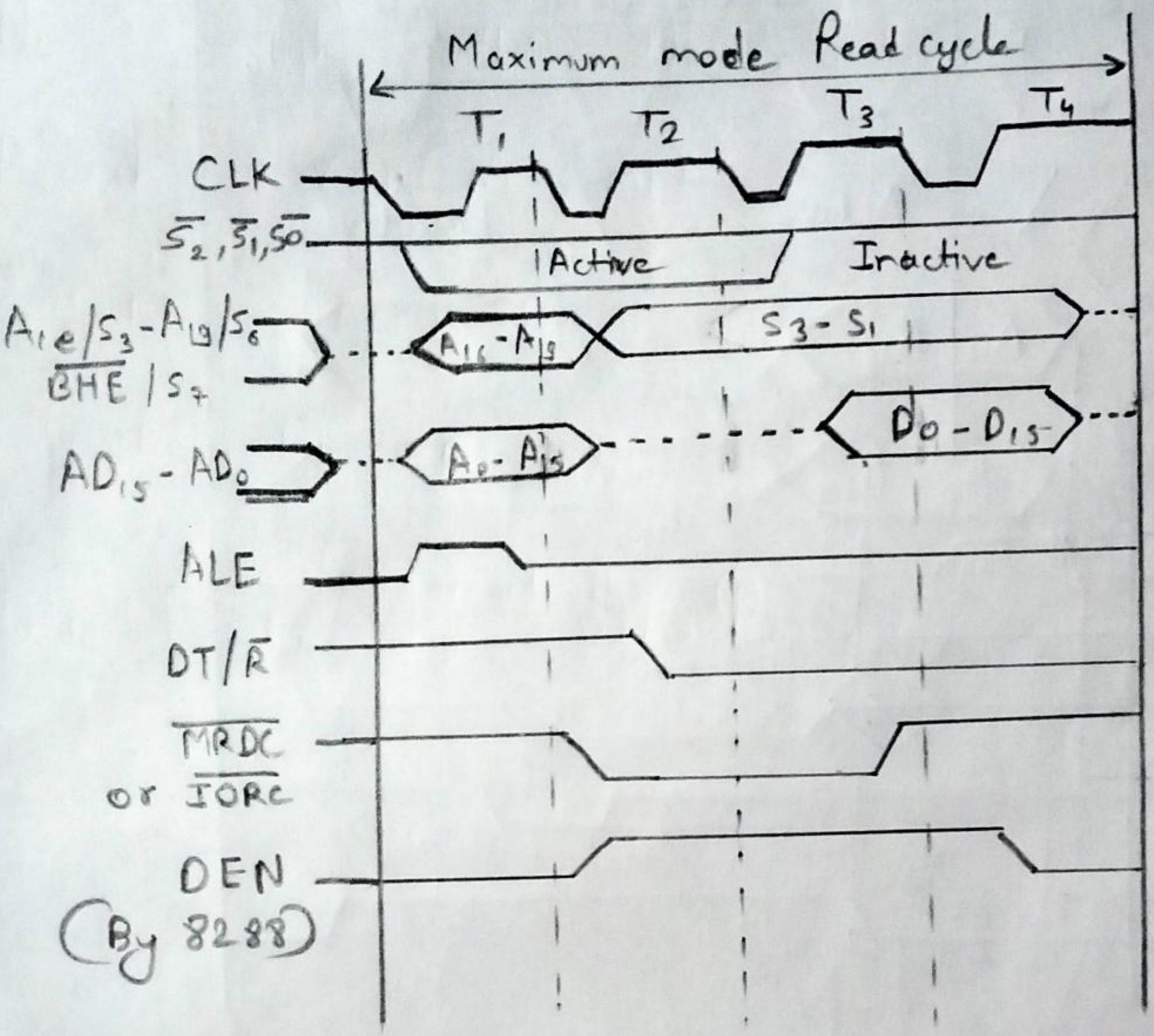
(1) Pipelining assumes the program will always be executed in a sequential manner.

(2) Hence, it fails when a branch occurs, as the pre-fetched instructions are no longer useful.

(3) As soon as 8086 detects a branch operation, it clears/discards the entire queue.

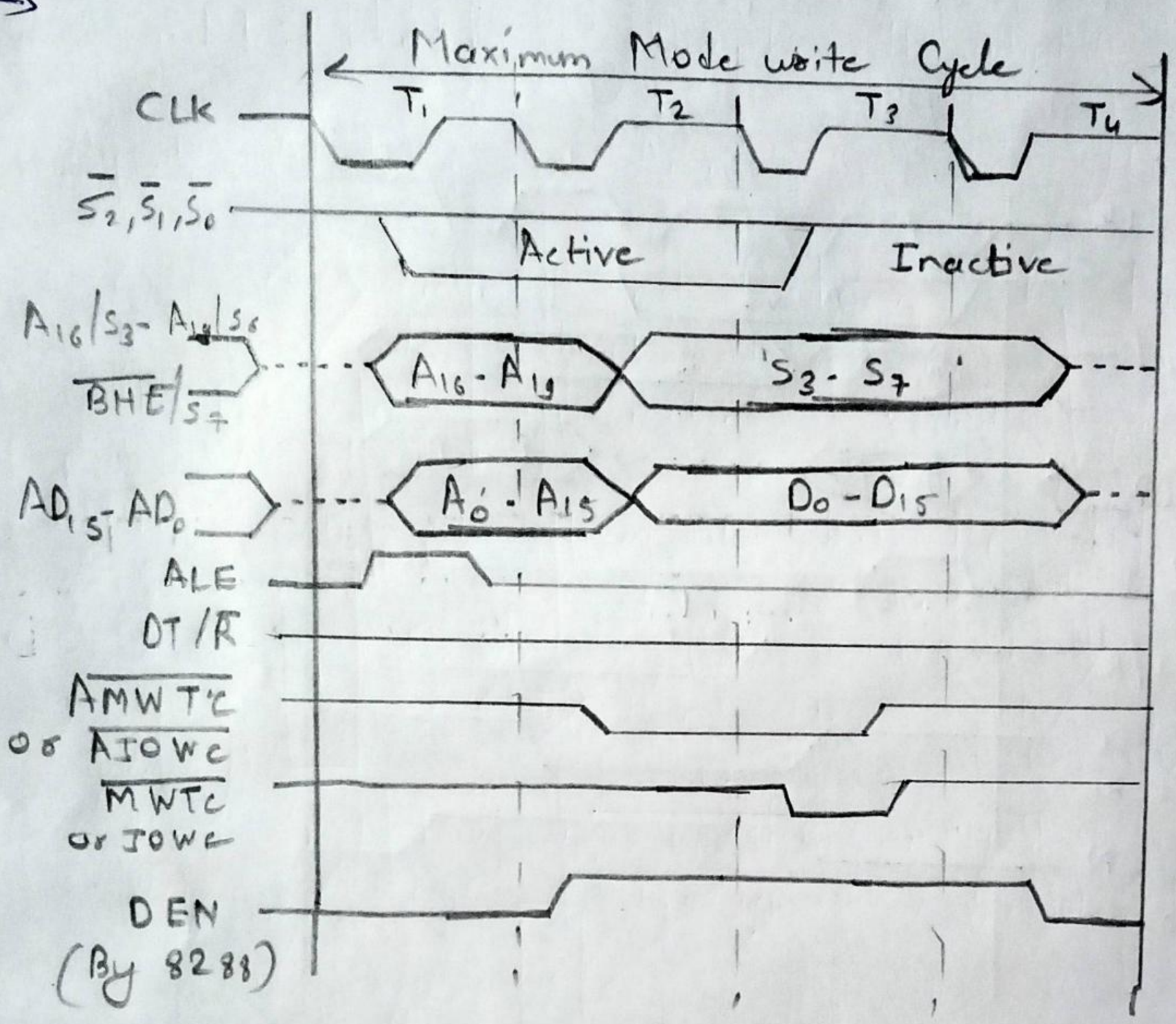
Q. 2) Draw timing diagram for read operation in maximum mode of 8086 microprocessor.

→



Q.3) Draw timing diagram for write operation in maximum mode of 8086 microprocess.

→



Q.4) Draw & Explain flag register of 8086 microprocessor.

→ ① 8086 has a 16-bit flag register.
② It has 9-flags, the rest are don't care bits denoted by the 'x' symbol
③ It is a continuation of the flag register of 8085. The lower 8-bits are exactly the same as 8085 flags.
④ These 9 flags are of two types :- 6-status flag & 3-control flag
⑤ Status flags are affected by ALU, after every arithmetic or logic operation.
⑥ They give status of the current result.
⑦ Control flags are used to control certain operations

X	X	X	X	CF	DF	IF	TF	SF	ZF	X	AF	X	PF	X	CF

Status flag:-

① Carry flag (CF)
- It indicates if the current ALU operation produced a carry out of MSB.

② Parity Flag (PF)
- It indicates the parity of the current result.

③ Auxillary Carry flag (AC)
- It indicates if the current ALU operation produced a carry from Lower ~~nibbel~~ nibble to higher nibble.

④ Zero flag :-
It indicates if the current ALU operation produced a zero result

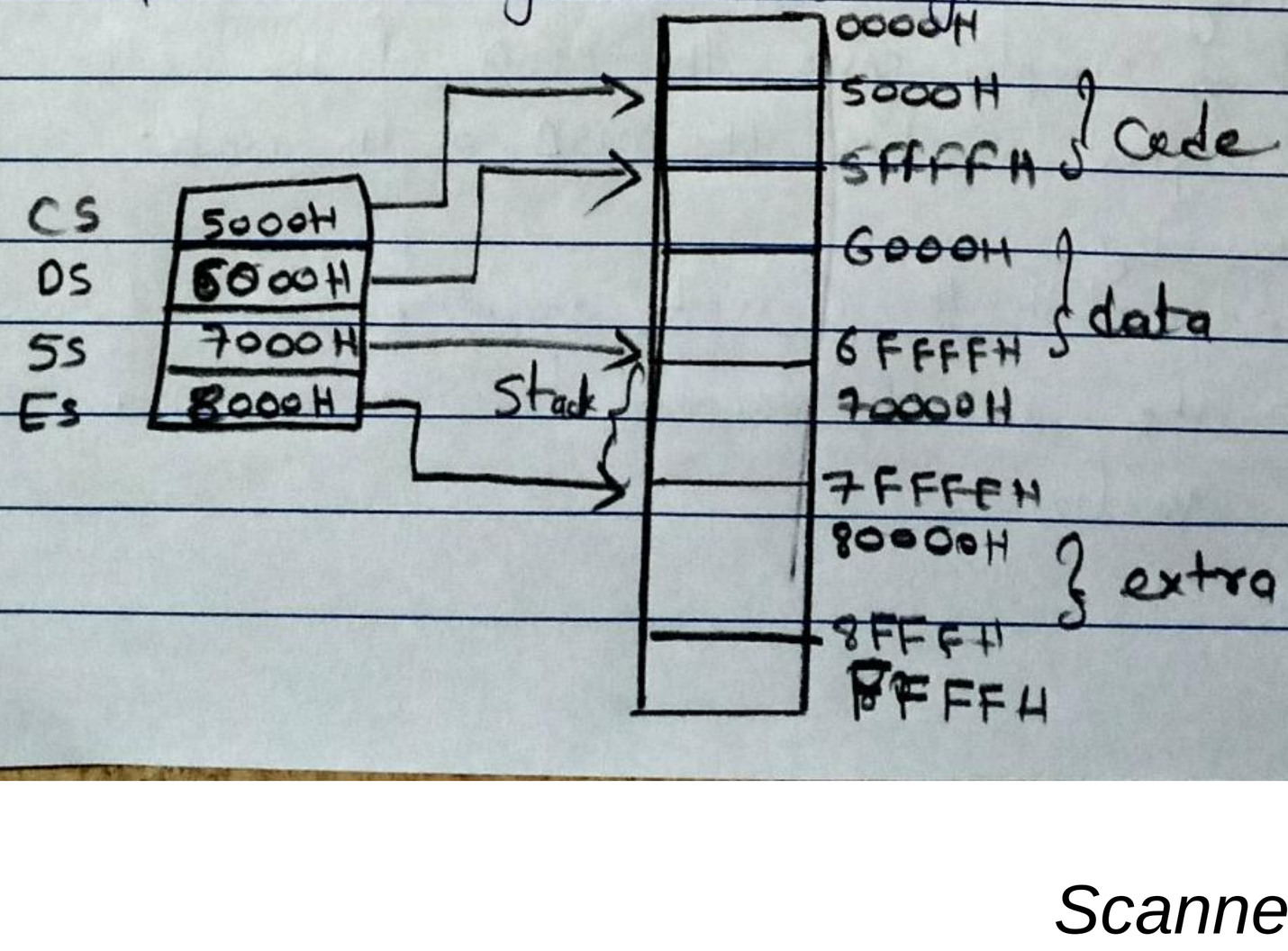
⑤ Sign flag :-
sign flag simply gives the MSB of the result.
For signed number the MSB of the number indicates its sign.

⑥ Overflow flag :-
It indicates if the current operation resulted in a signed overflow when the result exceeds this range then its is said to have overflow.

⑦ Trap flag (TF):- This flag is used if we need single step debugging in our code.
⑧ Interupt flag (IF):- This flag is used to enable the interrupt.
⑨ Direction flag (DF):- This flag is used for string instruction. If the flag is set, the string will be read from higher-order bits to lower order bits & vice versa.

Q 5] Explain Memory segmentation of 8086 microprocessor with its advantages.

→ ① 8086 has 20-bit address bus while the registers are of 16-bit.
② To access memory location, provide 20-bit address is provided while the register are 16 bit, this is made possible using implementation.
③ Segmentation in 8086 refers to division of the 1MB main memory into segments or blocks of 64kB each. This is done so as to access a memory segment using 16 bit address or pointer registers
④ There are four registers called as segment registers.
(i) The code segment (CS) register
(ii) The Stack segment (SS) register
(iii) The extra segement (ES) register
(iv) The data segment (DS) register



Advantages:-

① The programmer can access a memory that required 20-bit address, by using 16-bit register only.
② The programs, data & stack are ~~stored in~~ stored in separate blocks in memory and hence the three are organized in modular fashion.
③ It also help in object oriented programming to store data of an object.
④ Sharing of data or passing of data from one to another program.

Q.6) Write a program to display "I SE AIML" on IBM PC. Use INT 21H Function, AH = 09 with string of message terminated by '$'

→
- Stack 100H
- DATA

```
msg DB 'SE AIML', '$'
```

- CODE

```
MAIN PROC
    MOV AX, @DATA
    MOV DS, AX
    LEA DX, msg
    MOV AH, 09H
    INT 21H
    END MAIN
ret
```

Q.7] Write an assembly language program to no. of determine no. of positive & negative numbers from a series of ten 8-bit signed numbers stored from 2000H: 3000H. Store the counts in consecutive locations in memory after data.

→ Program:-

```
Code    SEGMENT
        ASSUME  CS: Code
        MOV AX, 2000H
        MOV DS, AX
        MOV SI, 0000H
        MOV CX, 000AH
        mov AH, 00H
Back:   MOV AL, [SI]
        RCL AL, 01H
        JNC  skip
        INC  AH
skip:   INC  SI
        INC  SI
        Loop Back
        MOV [SI], AH
        INT 3
Code    ENDS
        END
```

Q.8) Write a short note on string instructions of 8086 microprocessor

→ ① A string is a series of bytes stor sequentially in memory
② String with such instruction operates on 'strings'.
② The source string is at a location pointed by SI in the data segment.
③ The Destination string is at a location pointed by DI in the extra segment.
④ The count for string operations is always given by CX.
⑤ Since CX is a 16-bit register we can transfer max 64kB Using a string instruction.

- **Move**: MOVSB/MOVSW (Move String)
  - It is used to transfer a word/byte from data segment to extra segment.
  - The offset of the source in data segment is in SI
  - The offset of the destination in extra segment is in DI
  - SI & DI are incremented/decremented depending upon the direction flag.

- LODS: LODSB/LODSW (Load string)
  - It is used to load AL(or AX) register with a byte (or word) from data segment.
  - The offset of the source in data segment is in SI
  - SI is incremented / decremented depending upon the direction flag (DF)

- STOS: STOSB/STOSW (store string)
  It is used in storing of segment.

7). g]

→ Differentiate between procedure & Macro.

Procedure	Macro.
① Accessed by CALL & RET mechanism during program execution.	① Accessed by name gives to macro when defined during assembly
② Machine code for instructions only put in memory ~~space~~ once.	② Machine code generated for instruction each time called.
③ Parameters are passed in registers, memory locations or stack	③ Parameters passed as part of statement which calls macro.
④ Procedure uses stack	④ Macro does not utilize stack.
⑤ A procedure can be defined anywhere in program using the directives PROC & ENDP	⑤ A macro can be defined anywhere in program using the directives MACRO & ENDM.

Q.10] Explain Operations:-

→ (i) PUSPF

- This instruction decrements the stack pointer by 2.
- It copies contents of flag register to the memory location pointed by stack pointer.

SS: [SP-1] ← $Flag_H$, SS [SP-2] ← $Flag_L$ SP ← SP-2

(ii) Loop :

Jump to specified label if CX not equal to 0; and decremented CX.

```
Eg:        MOV CX, 40H
Back:      MOV AL, BL
           ADD AL, BL
           :
           :
           MOV BL, AL
           LOOP BACK      ; Do CX ← CX-1
                          Go to back if CX not equal to 0
```

(iii) JNZ address :

It transfer execution control to address 'Label', if ZF = 0

(iv) XCHG :-

Exchanges a byte/word between the source and the destination specified in the instruction.

Source: Register, Memory location.

Destination:- Register, Memory Location.

(v) CLD :-

This instruction clears the direction flag. No other flags are affected.

(vi) NOP :-

- There is no operation performed while executing this instruction.
- 8086 requires 3T-states for this instruction.
- It is mainly used to insert time delays, and can also be used while debugging.