NAME : PRATHAMESH CHIKANKAR

BRANCH : CSE-(AI&ML)

ROLL NO. : AIML08

SUBJECT : MICROPROCESSOR

TOPIC : EXPERIMENT NO. 01

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.

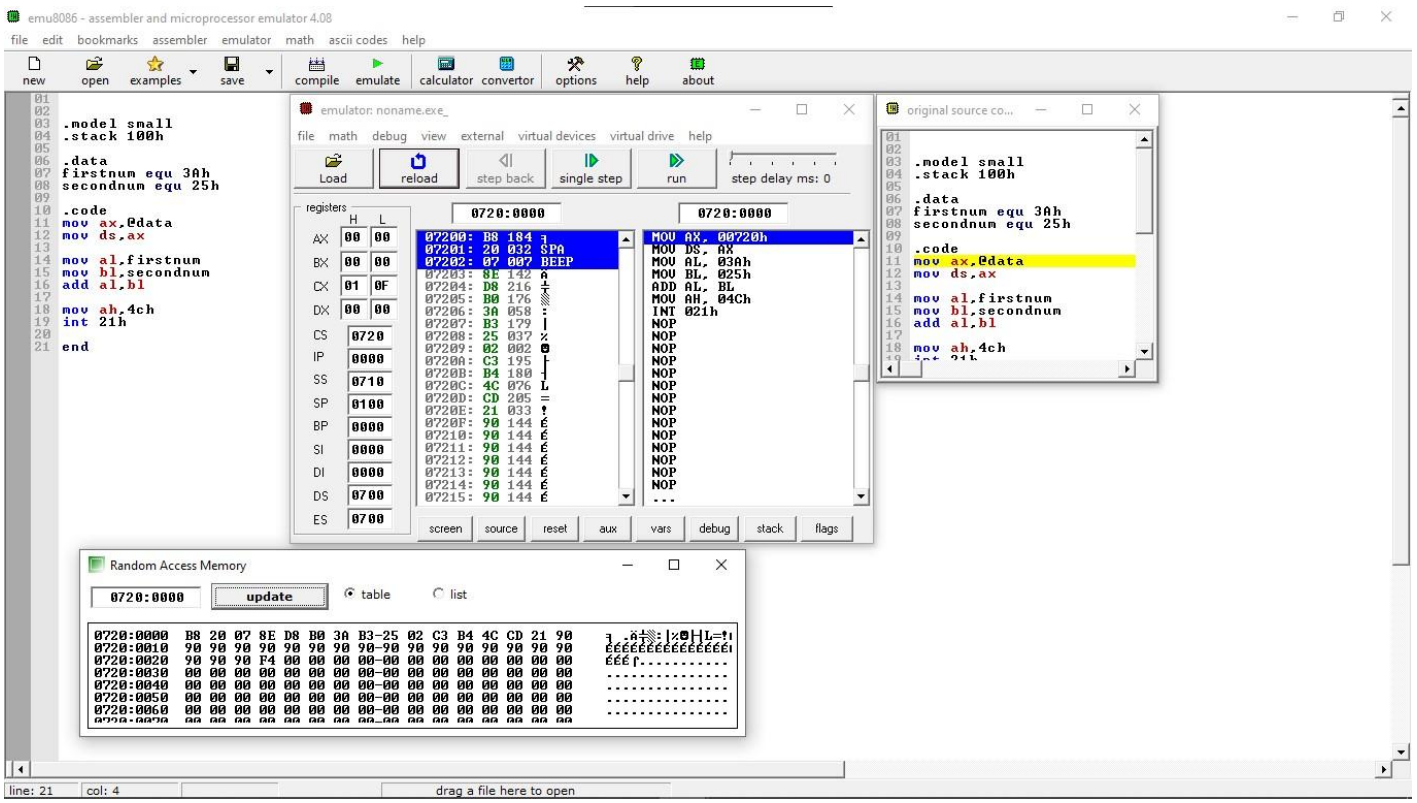
Example : JMP SHORT LABEL(8-bit)

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
```

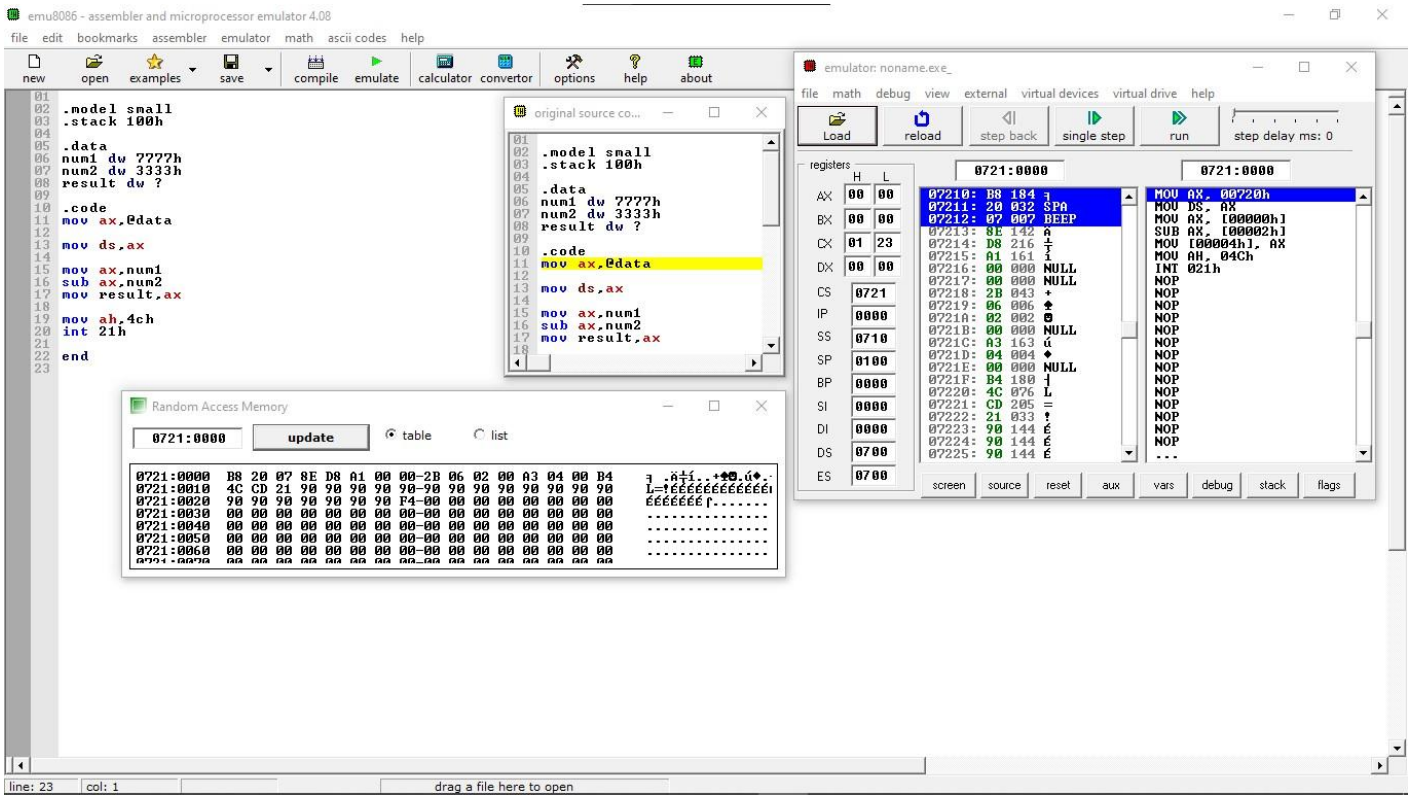
Result:


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 : PRATHAMESH CHIKANKAR

BRANCH : CSE-(AI&ML)

ROLL NO. : AIML08

SUBJECT : MICROPROCESSOR

TOPIC : EXPERIMENT NO. 02

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

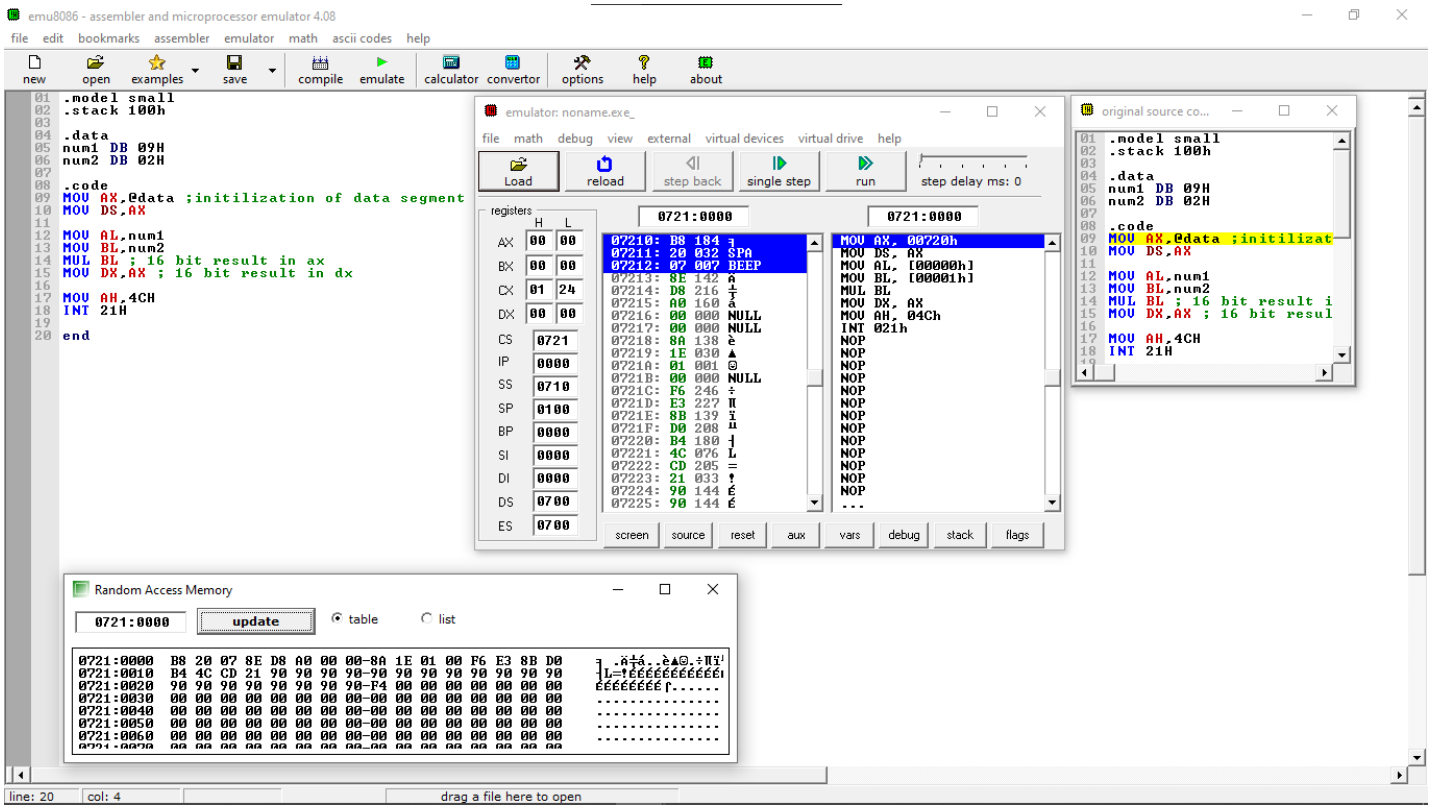
; Write assembly language program for 8086 microprocessor for multiplication of
; two 8 bits numbers, resulting in 16 bits with direct addressing modes.

```
.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

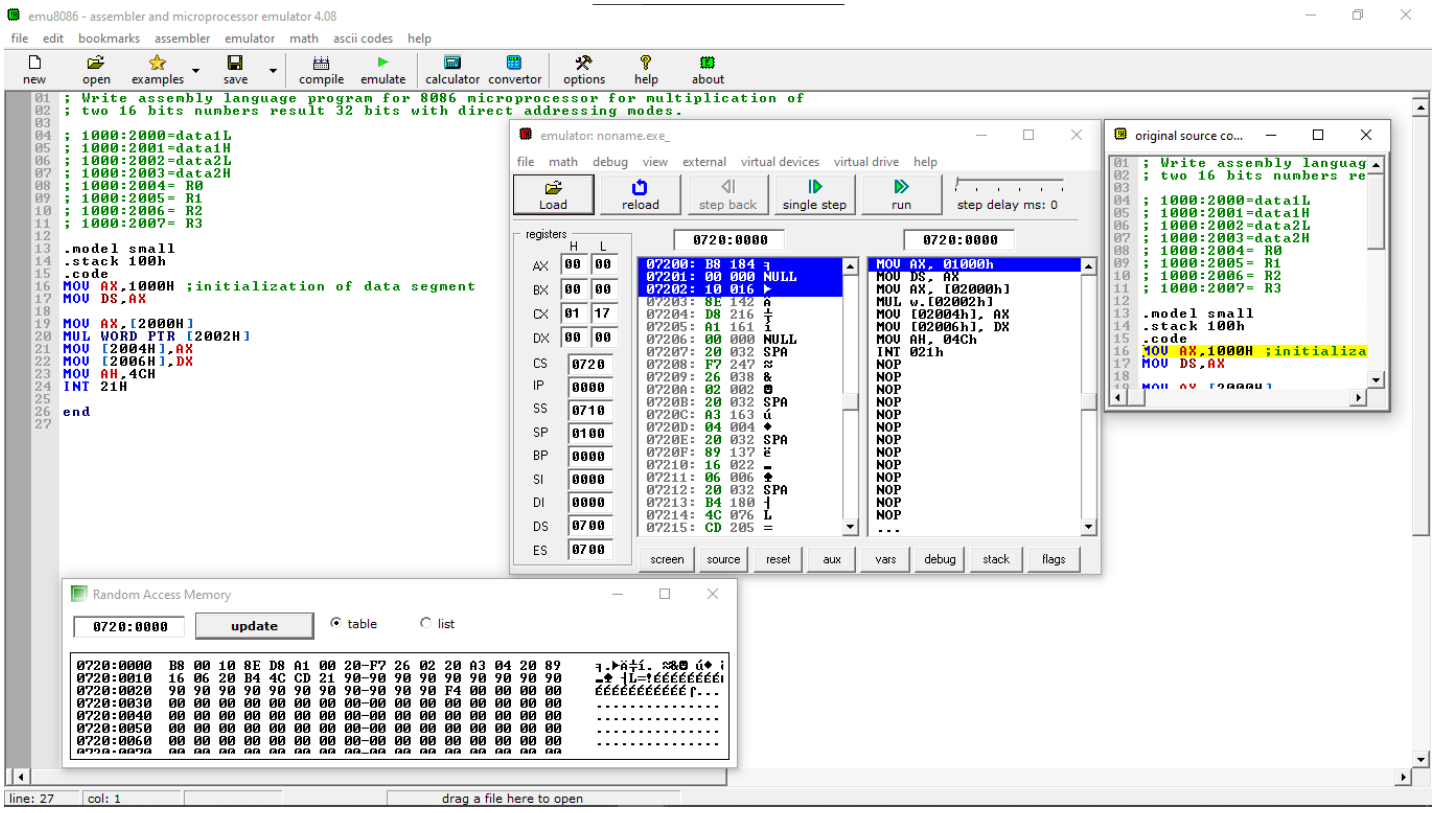
MOV [2006H],DX

MOV AH,4CH

INT 21H

end

Result:



2.3: Division a 16 bit number by 8 bit number

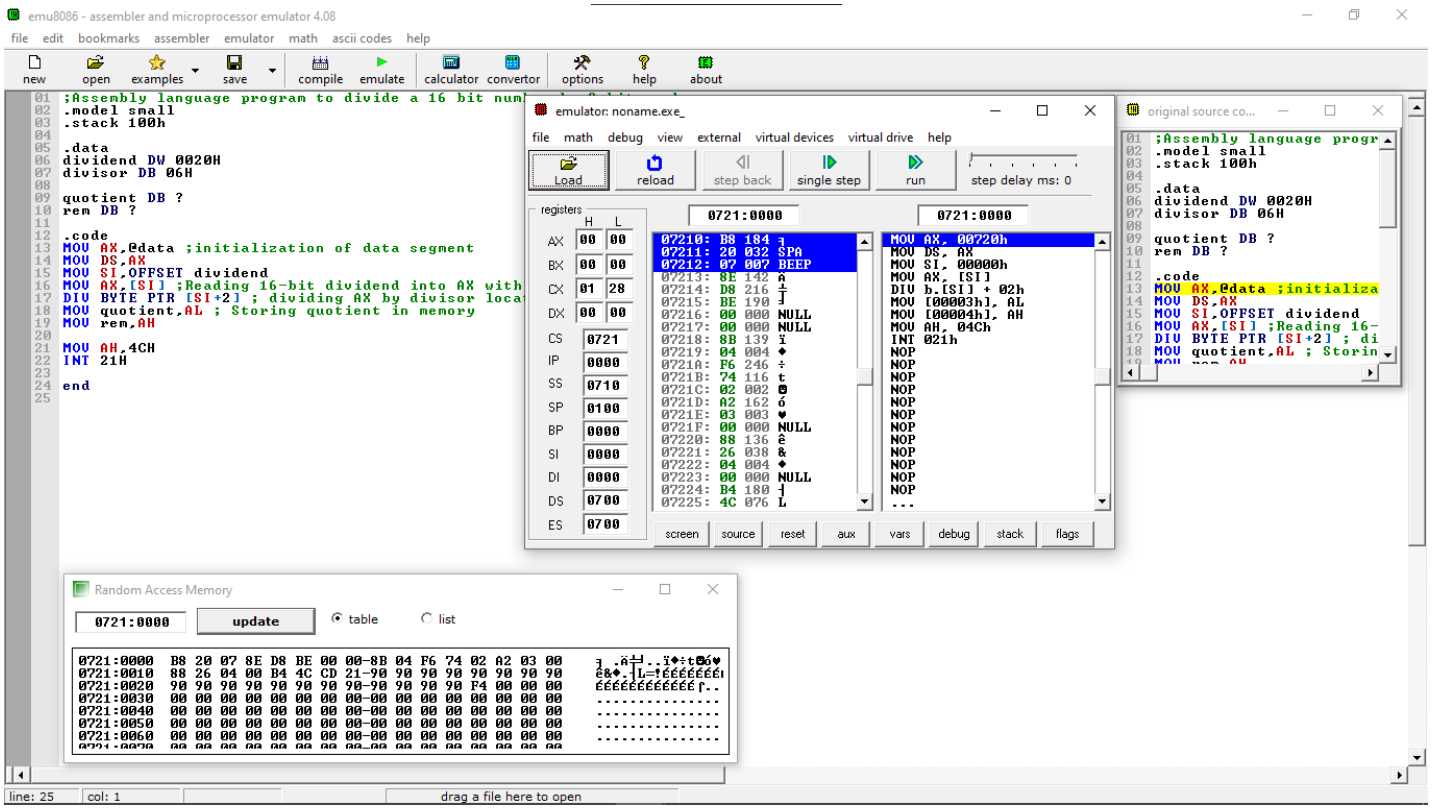
```
;Assembly language program to divide a 16 bit number by 8 bit number
.model small
.stack 100h

.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

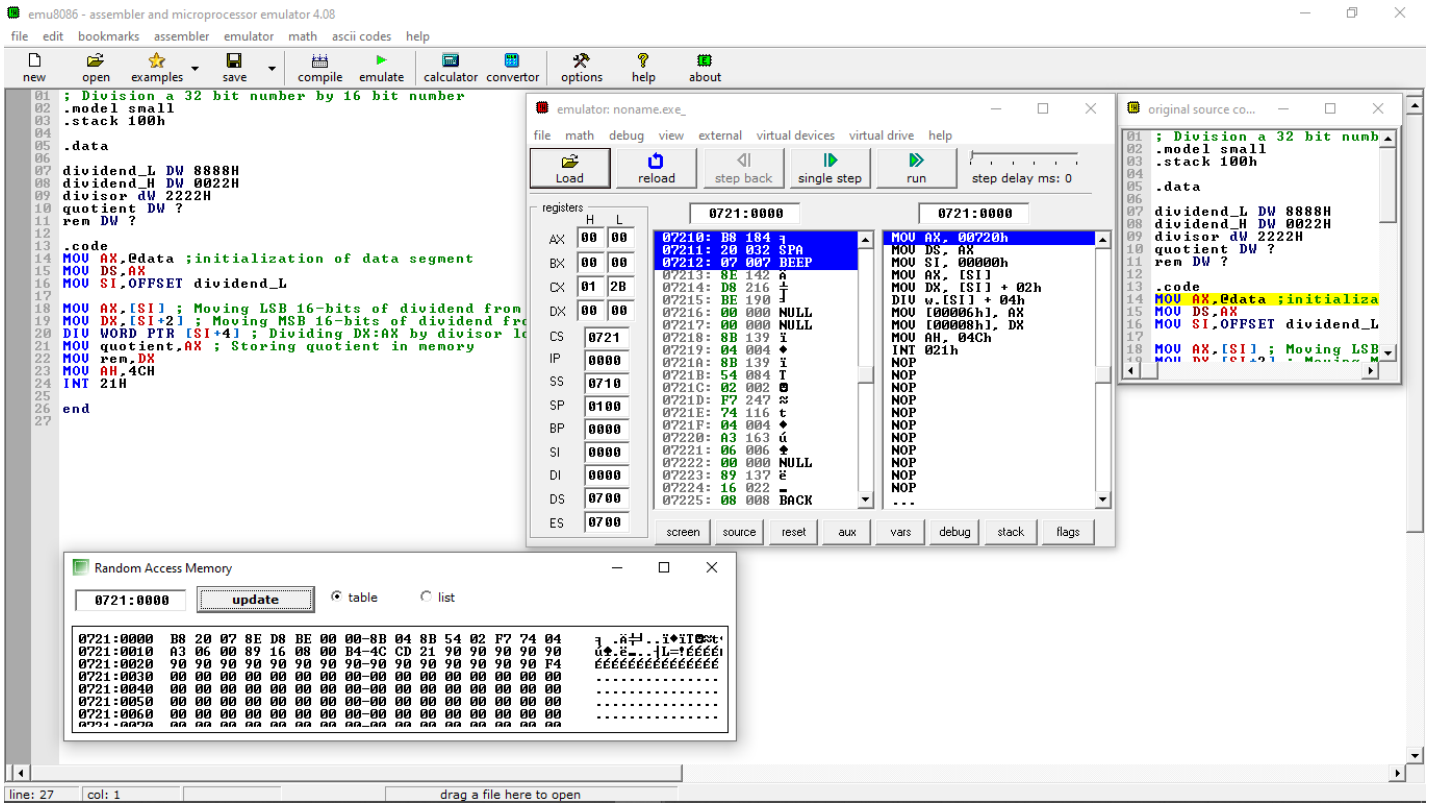
MOV rem,DX

MOV AH,4CH

INT 21H

end

Result:



Name : Prathamesh Chikankar
Branch : CSE-(AI & ML)
Roll No. : AIML08
Subject : Microprocessor
Topic : Experiment No. 03
Sign : Prathu 14th March '22

Exp: No. 3

Aim: Assembly language programming for Microprocessor 8086 for finding the largest and smallest numbers out of an array of Numbers.

Apparatus: 8086 Emulator, PC.

Theory:

Explain the following instructn:

1) JNC and JC

JNC: Jump if Not carry

Transfer execution control to address 'label' if CF=0.

JNC instructn transfer program control to the specified address if the carry flag is zero. Otherwise, executn continuous with the next instructn. No flags are affected by this instructn.

JC: Jump if carry

Transfers execution control to address 'label' if CF=1.

This instructn is used to jump the address as provided in instructn.

2) Loop label

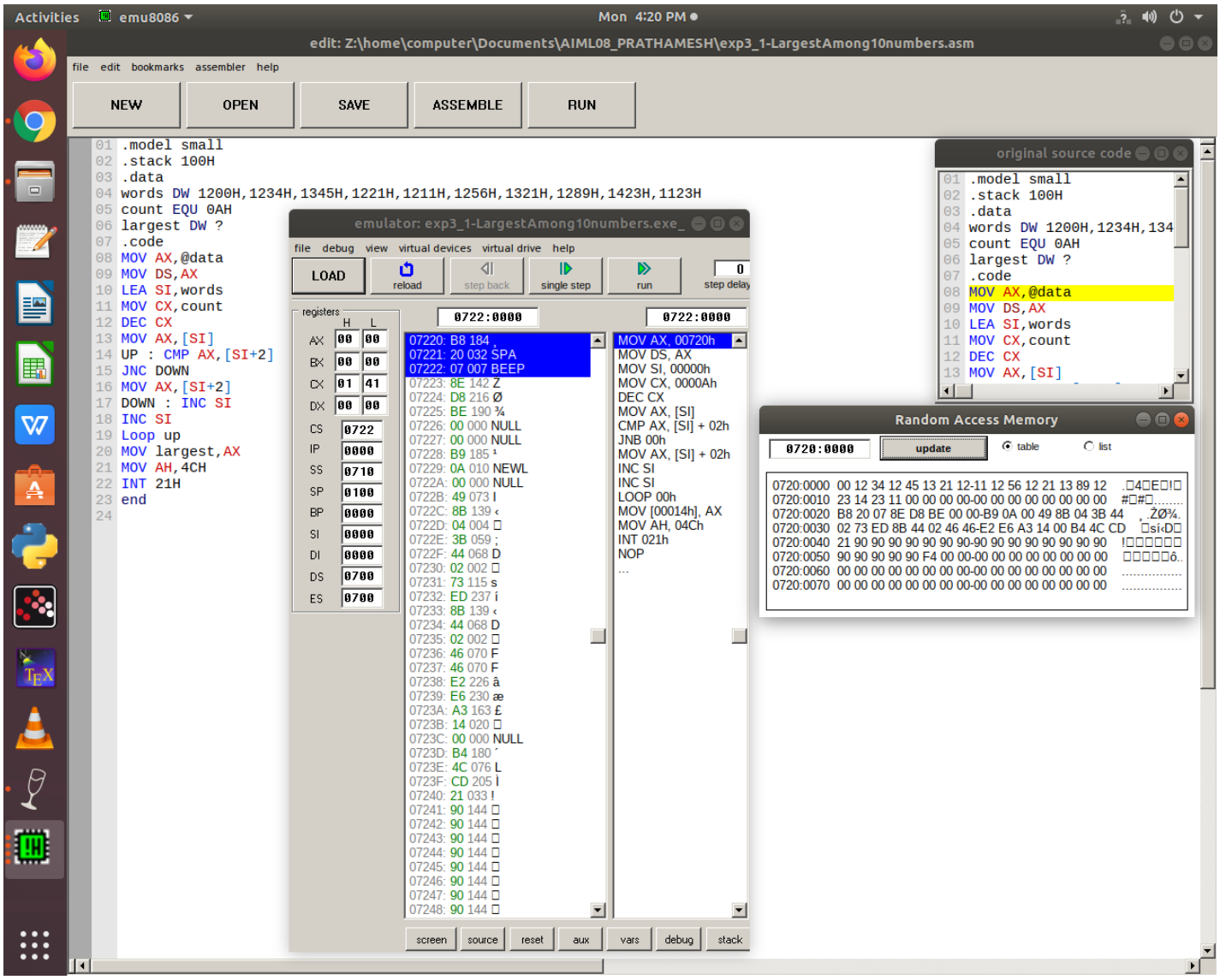
Jump to specified label if CX not equal to 0;
and decrement CX.

```
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.
```

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

```
.model small
.stack 100h
.code
MOV AX, @data
MOV DS, AX      ; initialization of segment data
LEA SI, words   ; loading effective address of word series in SI
MOV CX, count   ; Number of words in CX
DEC CX  ; for count no. of words & comparison iteration count-1.
MOV AX, [SI]  ; loading data word from memory to AX
up: CMP AX, [SI+2]; comparing word in AX with next word in memory
JNC down  ; if carry flag=0 then jump to down
MOV AX, [SI+2] ; If carry flag=1 then move larges word to AX
down: INC SI
INC SI   ; increment points by 2 for next word
LOOP UP  ; If CX not equal to 0 then go for next comparison
MOV largest, AX  ; Move largest no. to memory after data words.
MOV AH, 4Ch
INT 21H
end
```

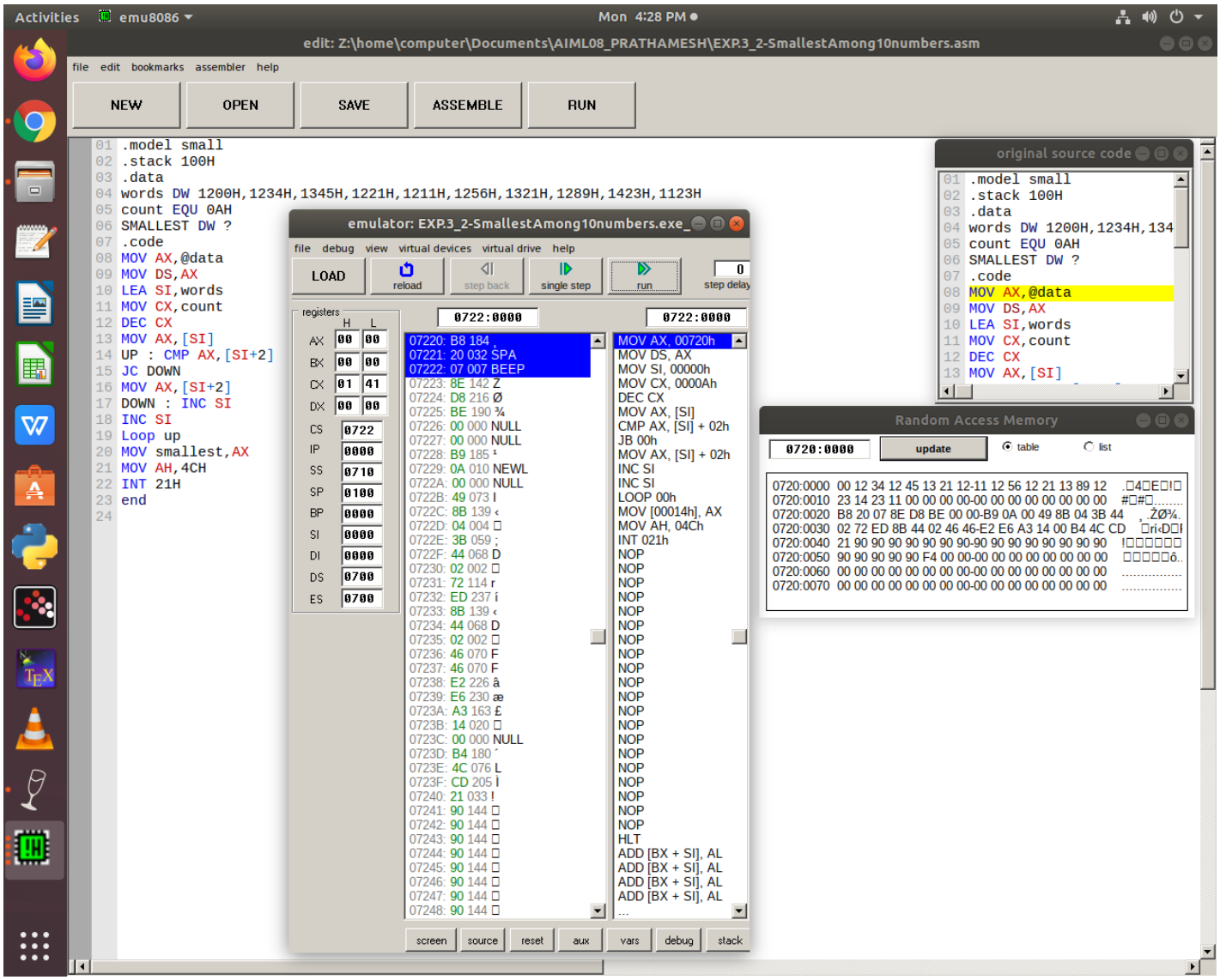
Result :


Activities
emu8086
Mon 4:20 PM
edit: Z:\home\computer\Documents\AIML08_PRATHAMESH\exp3_1-LargestAmong10numbers.asm
file edit bookmarks assembler help
NEW
OPEN
SAVE
ASSEMBLE
RUN
01 .model small
02 .stack 100H
03 .data
04 words DW 1200H,1234H,1345H,1221H,1211H,1256H,1321H,1289H,1423H,1123H
05 count EQU 0AH
06 largest DW ?
07 .code
08 MOV AX,@data
09 MOV DS,AX
10 LEA SI,words
11 MOV CX,count
12 DEC CX
13 MOV AX,[SI]
14 UP : CMP AX,[SI+2]
15 JNC DOWN
16 MOV AX,[SI+2]
17 DOWN : INC SI
18 INC SI
19 Loop up
20 MOV largest,AX
21 MOV AH,4CH
22 INT 21H
23 end
24
emulator: exp3_1-LargestAmong10numbers.exe_
file debug view virtual devices virtual drive help
LOAD
reload
step back
single step
run
step delay
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
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 □
07231: 73 115 s
07232: ED 237 í
07233: 8B 139 ‹
07234: 44 068 D
07235: 02 002 □
07236: 46 070 F
07237: 46 070 F
07238: E2 226 â
07239: E6 230 æ
0723A: A3 163 £
0723B: 14 020 □
0723C: 00 000 NULL
0723D: B4 180 ´
0723E: 4C 076 L
0723F: CD 205 Í
07240: 21 033 !
07241: 90 144 □
07242: 90 144 □
07243: 90 144 □
07244: 90 144 □
07245: 90 144 □
07246: 90 144 □
07247: 90 144 □
07248: 90 144 □
MOV AX, 00720h
MOV DS, AX
MOV SI, 00000h
MOV CX, 0000Ah
DEC CX
MOV AX, [SI]
CMP AX, [SI] + 02h
JNB 00h
MOV AX, [SI] + 02h
INC SI
INC SI
LOOP 00h
MOV [00014h], AX
MOV AH, 04Ch
INT 021h
NOP
...
screen
source
reset
aux
vars
debug
stack
original source code
01 .model small
02 .stack 100H
03 .data
04 words DW 1200H,1234H,134
05 count EQU 0AH
06 largest DW ?
07 .code
08 MOV AX,@data
09 MOV DS,AX
10 LEA SI,words
11 MOV CX,count
12 DEC CX
13 MOV AX,[SI]
Random Access Memory
0720:0000
update
table
list
0720:0000 00 12 34 12 45 13 21 12-11 12 56 12 21 13 89 12 .□4□E□!□
0720:0010 23 14 23 11 00 00 00 00-00 00 00 00 00 00 00 00 #□#□.....
0720:0020 B8 20 07 8E D8 BE 00 00-B9 0A 00 49 8B 04 3B 44 ¸.ŽØ¾..
0720:0030 02 73 ED 8B 44 02 46 46-E2 E6 A3 14 00 B4 4C CD □sí‹D□
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 the smallest 16-bit number from the array of numbers stored in memory.

```
.model small
.stack 100h
.code
MOV AX, @data
MOV DS, AX ;initialization of data segment
LEA SI, words ;loading effective address of word series in SI
MOV CX, count ;Number of words in CX
DEC CX ;for count no. of words & comparison, count-1
MOV AX, [SI] ;loading data word from memory to AX
up: CMP AX, [SI+2] ; comparing word in AX with next word
JC down ; If carry flag=1 then jump to down
MOV AX, [SI+2] ; if carry flag=0, then move larger word to AX
down: INC SI
INC SI ; increment pointer by 2 for next word
LOOP up ; if CX not equal to 0 then go for next comparison
MOV largest, AX ; Move largest NO. to memory after data words
MOV AH, 4Ch
INT 21H
end
```

Result :


Activities
emu8086
Mon 4:28 PM
edit: Z:\home\computer\Documents\AIML08_PRATHAMESH\EXP.3_2-SmallestAmong10numbers.asm
file edit bookmarks assembler help
NEW
OPEN
SAVE
ASSEMBLE
RUN
.model small
.stack 100H
.data
words DW 1200H,1234H,1345H,1221H,1211H,1256H,1321H,1289H,1423H,1123H
count EQU 0AH
SMALLEST DW ?
.code
MOV AX,@data
MOV DS,AX
LEA SI,words
MOV CX,count
DEC CX
MOV AX,[SI]
UP : CMP AX,[SI+2]
JC DOWN
MOV AX,[SI+2]
DOWN : INC SI
INC SI
Loop up
MOV smallest,AX
MOV AH,4CH
INT 21H
end
emulator: EXP.3_2-SmallestAmong10numbers.exe_
file debug view virtual devices virtual drive help
LOAD
reload
step back
single step
run
step delay
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
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 □
07231: 72 114 r
07232: ED 237 í
07233: 8B 139 ‹
07234: 44 068 D
07235: 02 002 □
07236: 46 070 F
07237: 46 070 F
07238: E2 226 â
07239: E6 230 æ
0723A: A3 163 £
0723B: 14 020 □
0723C: 00 000 NULL
0723D: B4 180 ´
0723E: 4C 076 L
0723F: CD 205 Í
07240: 21 033 !
07241: 90 144 □
07242: 90 144 □
07243: 90 144 □
07244: 90 144 □
07245: 90 144 □
07246: 90 144 □
07247: 90 144 □
07248: 90 144 □
MOV AX, 00720h
MOV DS, AX
MOV SI, 00000h
MOV CX, 0000Ah
DEC CX
MOV AX, [SI]
CMP AX, [SI] + 02h
JB 00h
MOV AX, [SI] + 02h
INC SI
INC SI
LOOP 00h
MOV [00014h], AX
MOV AH, 04Ch
INT 021h
NOP
screen
source
reset
aux
vars
debug
stack
original source code
.model small
.stack 100H
.data
words DW 1200H,1234H,134
count EQU 0AH
SMALLEST DW ?
.code
MOV AX,@data
MOV DS,AX
LEA SI,words
MOV CX,count
DEC CX
MOV AX,[SI]
Random Access Memory
0720:0000
update
table
list
0720:0000 00 12 34 12 45 13 21 12-11 12 56 12 21 13 89 12
0720:0010 23 14 23 11 00 00 00 00-00 00 00 00 00 00 00 00
0720:0020 B8 20 07 8E D8 BE 00 00-B9 0A 00 49 8B 04 3B 44
0720:0030 02 72 ED 8B 44 02 46 46-E2 E6 A3 14 00 B4 4C CD
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
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: Prathamesh Chikankar
Branch: CSE-(AI & ML)
ROLL NO.: AIML08
Subject: Microprocessor
Topic: Experiment No. 04
Sign: Prathy 14th March '22

Exp: No. 4

Aim: Assembly language programming for microprocessor 8086 to sort the Numbers in ascending and Descending order..

Apparatus: 8086 Emulator, PC.

Theory:

Explain the following assembler directives:

1. ORG - Originate

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

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

2. DB - define Byte

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

for eg. X DB 5.

this statement creates a byte variable named as x and the value initialized 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 and the value initialized 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 no reference to it is

formed in the current assembly.

6. END

this is placed at the end of a 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 easy 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 name of code segment with CS register.

8. ENDS

ENDS directive is used to indicate the end of segment.

eg:-

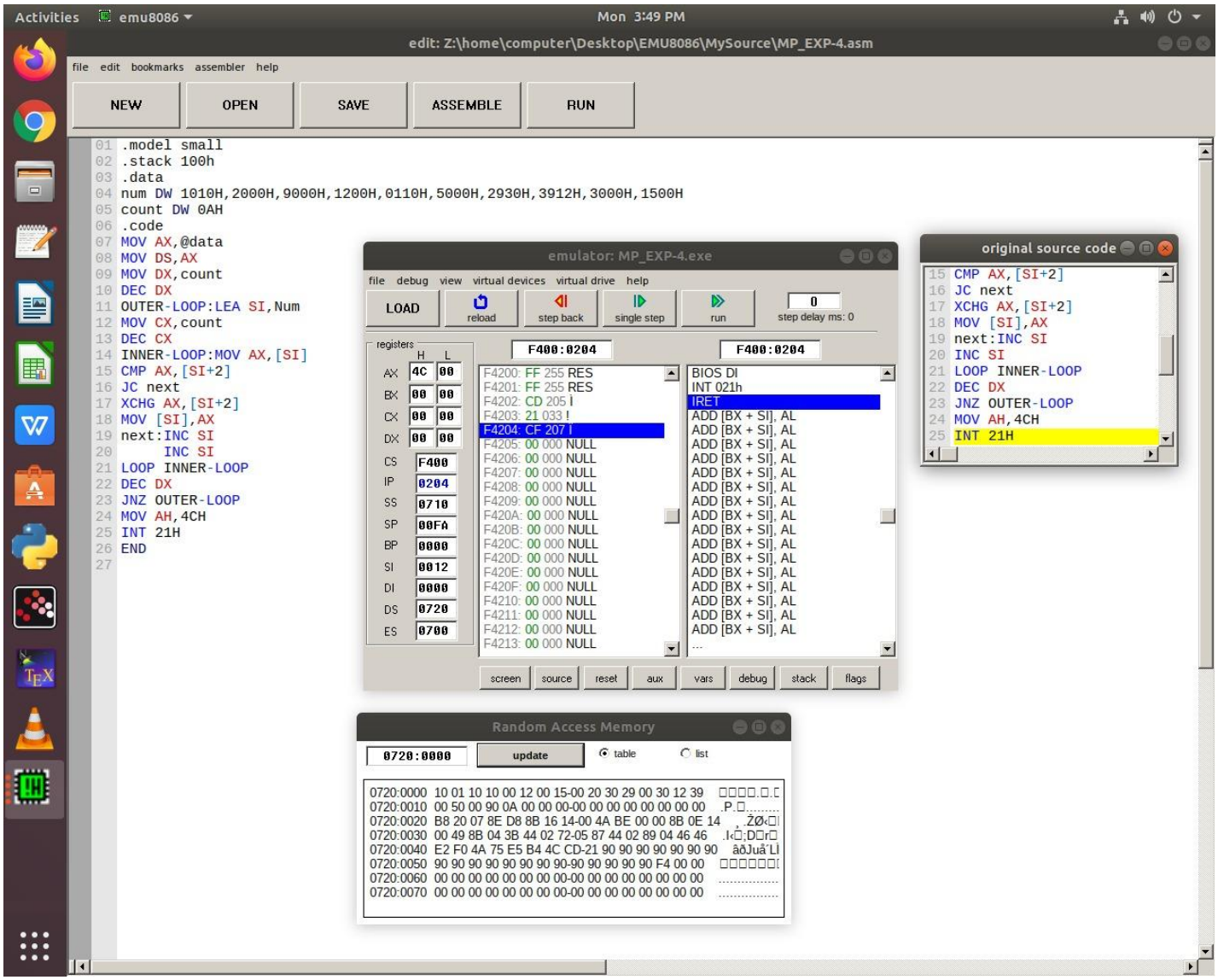
```
DATA ENDS.
```

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

```
.model small
.stack 100H
.data
num DW 1010H, 2000H, 9000H, 1200H, 0110H, 5000H, 2930H, 3912H, 1500H
                                                              3000H,
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
```

Result :

4.1:

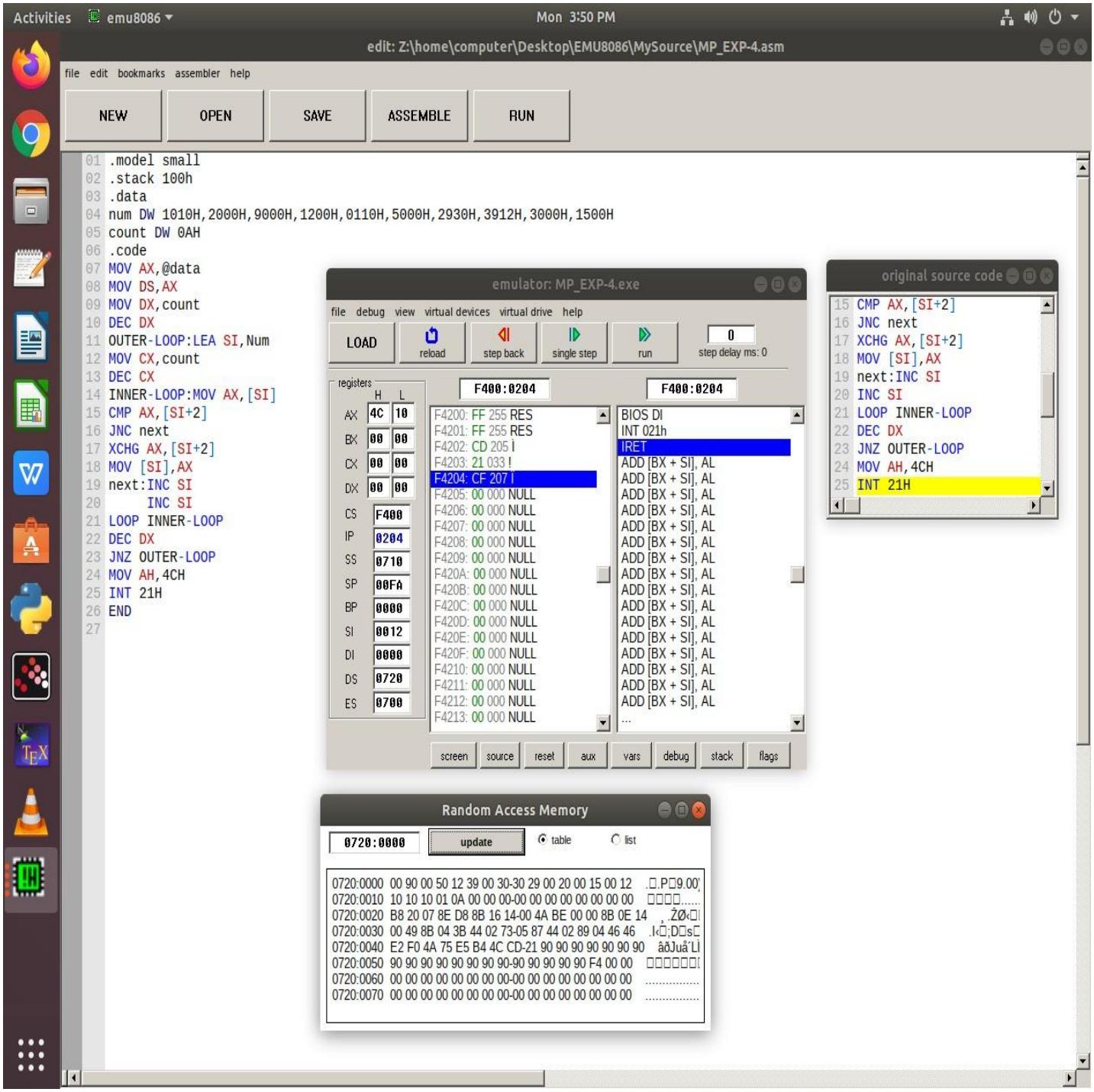

Activities
emu8086
Mon 3:49 PM
edit: Z:\home\computer\Desktop\EMU8086\MySource\MP_EXP-4.asm
file edit bookmarks assembler help
NEW
OPEN
SAVE
ASSEMBLE
RUN
01 .model small
02 .stack 100h
03 .data
04 num DW 1010H,2000H,9000H,1200H,0110H,5000H,2930H,3912H,3000H,1500H
05 count DW 0AH
06 .code
07 MOV AX,@data
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
step delay ms: 0
registers
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
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
BIOS DI
INT 021h
IRET
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
0720:0020 B8 20 07 8E D8 8B 16 14-00 4A BE 00 00 8B 0E 14
0720:0030 00 49 8B 04 3B 44 02 72-05 87 44 02 89 04 46 46
0720:0040 E2 F0 4A 75 E5 B4 4C CD-21 90 90 90 90 90 90 90
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 orders.

```
.model small
.stack 100h
.data                                                    3000H,
num DW 1010H, 2000H, 9000H, 1200H, 0110H, 3000H, 2930H, 3912H, 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          ; one less than no. of elements
INNER-LOOP : MOV AX, [SI]
CMP AX, [SI+2]      ; compare consecutive locatn
JNC next
XCHG AX, [SI+2]
MOV [SI], AX
next : INC SI
       INC SI
LOOP INNER-LOOP  ; repeat for next consecutive memory locatn
DEC DX           ; decrement outer counter
JNZ OUTER-LOOP   ; repeat all comparison if counter is not zero.
MOV AH, 4CH      ; Terminate program
INT 21H
END              ; end
```

Result :

4.2:


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
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
F400:0204
F4200: FF 255 RES
F4201: FF 255 RES
F4202: CD 205 I
F4203: 21 033 !
F4204: CF 207 I
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
BIOS DI
INT 021h
IRET
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
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
0720:0030 00 49 8B 04 3B 44 02 73-05 87 44 02 89 04 46 46
0720:0040 E2 F0 4A 75 E5 B4 4C CD-21 90 90 90 90 90 90 90
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 : Prathamesh chikankar
BRANCH : CSE-(AI & ML)
ROLL NO. : AIML08
Subject : Microprocessor
Topic : Experiment No. 05
SIGN : Prathu 20th March '22

Exp: No. 5

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

Apparatus: 8086 Emulator, PC

Theory: Explain the following instruction, DAA 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, PF, ZF and SF

Address mode - implied addressing mode

Operation - AL ← sum in adjustment to packed BCD format

- this instruction is used to make sure that the 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 after addition is greater than 9 or if the auxiliary carry flag is set add 6.
- if the higher nibble of AH is greater than 9 or if the carry flag is set then add 60H.

Example: if AL = 59 H valid BCD, BL = 34H valid BCD

ADD AL, BL

```
  0 1 0 1   1 0 0 1
+ 0 0 1 1   0 1 0 0
-----------------------
  1 0 0 0   1 1 0 1
```

8 D

AL = 8 DH invalid BCD after addition of AL and BL

DAA →

```
  1 0 0 0   1 1 0 1
  0 0 0 0   0 1 1 0
-----------------------
  1 0 0 1   0 0 1 1
```

9 3

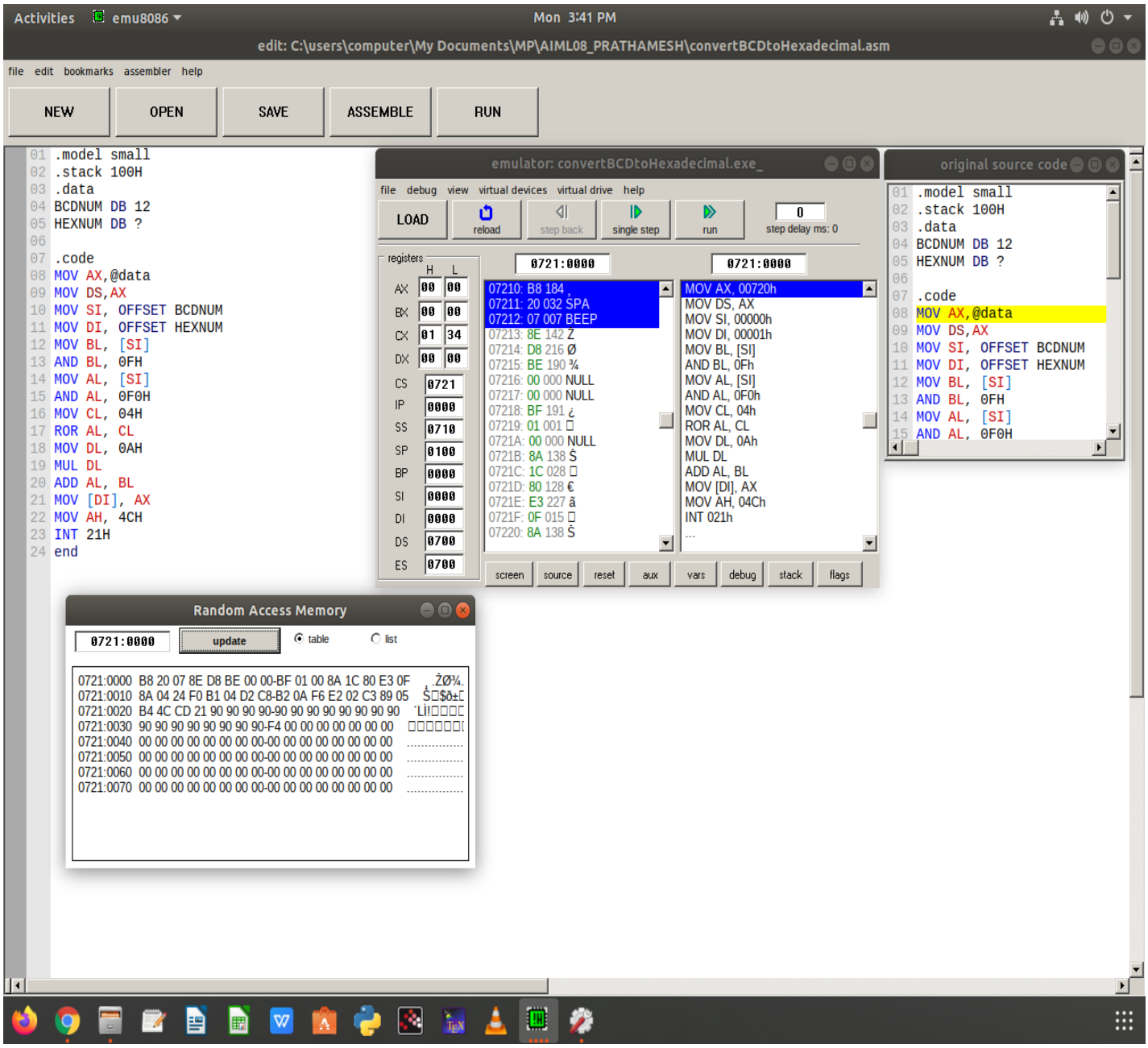
∴ AL = 93 H BCD

5.1.: Assembly language programming for Microprocessor 8086 to Convert BCD numbers to Hexadecimal numbers.

```
.model small
.stack 100H
.data
BCDNUM DB 12
HEXNUM DB ?
.code
MOV AX, @data
MOV DS, AX
MOV SI, OFFSET BCDNUM ; initialize SI with memory offset- BCDNUM
MOV DI, OFFSET HEXNUM ; initialize DI with memory offset- HEXNUM
MOV BL, [SI]          ; load data from memory to BL
AND BL, OFH ; BL=02   AND OFH with BL
MOV AL, [SI]          ; load data from memory to AL
AND AL, OFOH ; AL=010 AND FOH with AL
MOV BX, 0 CL, 04H     ; load CL with 4
ROR AL, CL ; AL=01    Rotate AL to right
MOV DL, 0AH           ; load DL with 0AH
MUL DL                ; multiply DL with AL
ADD AL, BL            ; Add AL and BL
MOV [DI], AL ; store AL into memory
MOV AH, 4CH
INT 21H
END                   ; terminate the program
```

Result :

5.1:

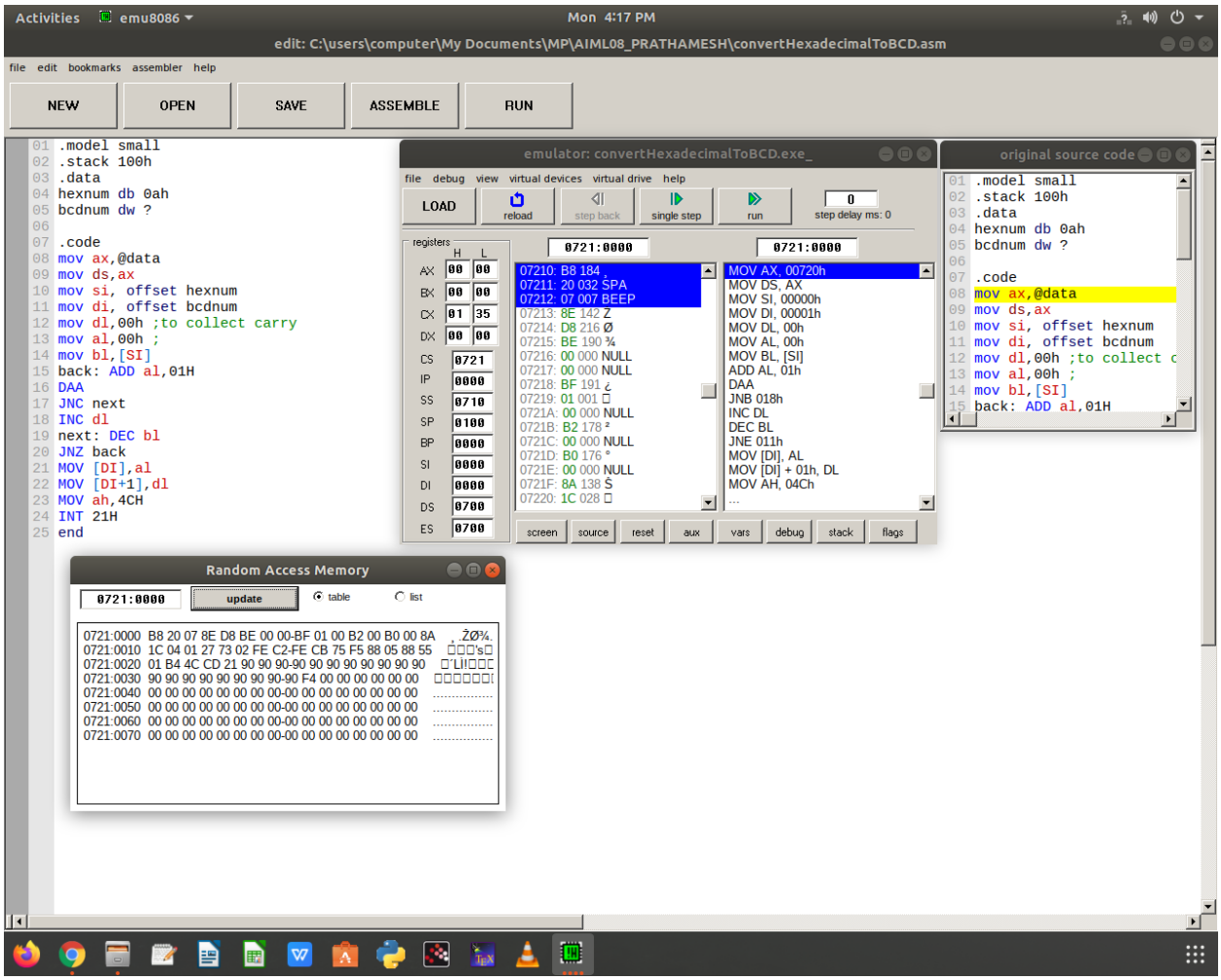

Activities
emu8086
Mon 3:41 PM
edit: C:\users\computer\My Documents\MP\AIML08_PRATHAMESH\convertBCDtoHexadecimal.asm
file edit bookmarks assembler help
NEW
OPEN
SAVE
ASSEMBLE
RUN
01 .model small
02 .stack 100H
03 .data
04 BCDNUM DB 12
05 HEXNUM DB ?
06
07 .code
08 MOV AX,@data
09 MOV DS,AX
10 MOV SI, 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
20 ADD AL, BL
21 MOV [DI], AX
22 MOV AH, 4CH
23 INT 21H
24 end
emulator: convertBCDtoHexadecimal.exe_
file debug view virtual devices virtual drive help
LOAD
reload
step back
single step
run
step delay ms: 0
registers
H L
AX 00 00
BX 00 00
CX 01 34
DX 00 00
CS 0721
IP 0000
SS 0710
SP 0100
BP 0000
SI 0000
DI 0000
DS 0700
ES 0700
0721:0000
07210: B8 184 ¸
07211: 20 032 SPA
07212: 07 007 BEEP
07213: 8E 142 Ž
07214: D8 216 Ø
07215: BE 190 ¾
07216: 00 000 NULL
07217: 00 000 NULL
07218: BF 191 ¿
07219: 01 001 □
0721A: 00 000 NULL
0721B: 8A 138 Š
0721C: 1C 028 □
0721D: 80 128 €
0721E: E3 227 ã
0721F: 0F 015 □
07220: 8A 138 Š
0721:0000
MOV AX, 00720h
MOV DS, AX
MOV SI, 00000h
MOV DI, 00001h
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, 04Ch
INT 021h
...
screen source reset aux vars debug stack flags
original source code
01 .model small
02 .stack 100H
03 .data
04 BCDNUM DB 12
05 HEXNUM DB ?
06
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
Random Access Memory
0721:0000
update
table
list
0721:0000 B8 20 07 8E D8 BE 00 00-BF 01 00 8A 1C 80 E3 0F ¸.ŽØ¾.
0721:0010 8A 04 24 F0 B1 04 D2 C8-B2 0A F6 E2 02 C3 89 05 Š□$ð±Ò
0721:0020 B4 4C CD 21 90 90 90 90-90 90 90 90 90 90 90 90 ´LÍ!□□□□
0721:0030 90 90 90 90 90 90 90 90-F4 00 00 00 00 00 00 00 □□□□□□□ô
0721:0040 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ....
0721:0050 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ....
0721:0060 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ....
0721:0070 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 Hexadecimal number to BCD number.

```
.model small
.stack 100H
.data
HEXNUM DB 0AH
BCDNUM DW ?
.code
MOV AX, @data
MOV DS, AX
MOV SI, OFFSET HEXNUM ; initialize memory offset
MOV DI, OFFSET BCDNUM ; SI with BCD NUM
MOV BL, [SI]          ; load data from memory to BL
MOV DL, 00H ; To collect the carry bits generated in addition process
MOV AL, 00H
BACK: ADD AL, 1H ; Hexadecimal addition answer will come in AL
      DAA        ; Decimal addition answer is now in AL
      JNC NEXT
      INC DL
NEXT: DEC BL
      JNZ BACK
      MOV [DI], AL
      MOV [DI+1], DL
      MOV AH, 4CH ; terminate
      INT 21H
      END         ; terminate the program end
```

Result :

5.2:


Activities
emu8086
Mon 4:17 PM
edit: C:\users\computer\My Documents\MP\AIML08_PRATHAMESH\convertHexadecimalToBCD.asm
file edit bookmarks assembler help
NEW
OPEN
SAVE
ASSEMBLE
RUN
01 .model small
02 .stack 100h
03 .data
04 hexnum db 0ah
05 bcdnum dw ?
06
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
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: convertHexadecimalToBCD.exe_
file debug view virtual devices virtual drive help
LOAD
reload
step back
single step
run
step delay ms: 0
registers
H L
AX 00 00
BX 00 00
CX 01 35
DX 00 00
CS 0721
IP 0000
SS 0710
SP 0100
BP 0000
SI 0000
DI 0000
DS 0700
ES 0700
0721:0000
0721:0000
07210: B8 184 ¸
07211: 20 032 SPA
07212: 07 007 BEEP
07213: 8E 142 Ž
07214: D8 216 Ø
07215: BE 190 ¾
07216: 00 000 NULL
07217: 00 000 NULL
07218: BF 191 ¿
07219: 01 001 □
0721A: 00 000 NULL
0721B: B2 178 ²
0721C: 00 000 NULL
0721D: B0 176 °
0721E: 00 000 NULL
0721F: 8A 138 Š
07220: 1C 028 □
MOV AX, 00720h
MOV DS, AX
MOV SI, 00000h
MOV DI, 00001h
MOV DL, 00h
MOV AL, 00h
MOV BL, [SI]
ADD AL, 01h
DAA
JNB 018h
INC DL
DEC BL
JNE 011h
MOV [DI], AL
MOV [DI] + 01h, DL
MOV AH, 04Ch
...
screen source reset aux vars debug stack flags
original source code
01 .model small
02 .stack 100h
03 .data
04 hexnum db 0ah
05 bcdnum dw ?
06
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 c
13 mov al,00h ;
14 mov bl,[SI]
15 back: ADD al,01H
Random Access Memory
0721:0000
update
table
list
0721:0000 B8 20 07 8E D8 BE 00 00-BF 01 00 B2 00 B0 00 8A ¸ ŽØ¾ ¿ ² °Š
0721:0010 1C 04 01 27 73 02 FE C2-FE CB 75 F5 88 05 88 55 's þÂþËuõ U
0721:0020 01 B4 4C CD 21 90 90 90-90 90 90 90 90 90 90 90 ´LÍ!
0721:0030 90 90 90 90 90 90 90 90-90 F4 00 00 00 00 00 00 ô
0721:0040 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
0721:0050 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
0721:0060 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
0721:0070 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00

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

Name : Prathamesh S. Chikankar

Roll No. : AIML08

Year | Branch : SE | CSE-(AI & ML)

Subject | Topic : MP / Assignment No. 01

Signature : Prathu

6th March 2022

1. Explain the instruction pipelining features of 8086. Give its advantages and its disadvantages.

Ans:- - pipelining is a special feature of 8086 that increases the speed of the processor. In earlier processors, the process of fetching the instruct^n and executing them was one at a time or not overlapping. This made the execut^n unit wait until the instruct^n is fetched and also made the fetching unit wait when the instruct^n is under execut^n.

- In 8086, pipelining refers to overlapping the process of fetching the instruct^n when the previous instruct^n is in execut^n. This is possible because of the prefetch queue.
- 8086 has a 6 byte prefetch queue. The size of the queue is kept to be 6-byte so as to accomodate the largest instruct^n of 8086 which is of 6-byte.
- As the name says, 8086 (BIU) fetches instruct^n into this queue prior to execut^n.
- BIU fills in the queue, until the entire queue is full i.e. all six bytes are filled.
- In case, of a branching instruct^n like jump or call, the sequence of execut^n of the program has to be changed. In such a condit^n, the instruct^n in the queue are of no use. These are to be removed from the queue and the process is called flushing the queue. The instruct^n are then to be filled in the queue from the target locat^n or branching locat^n.

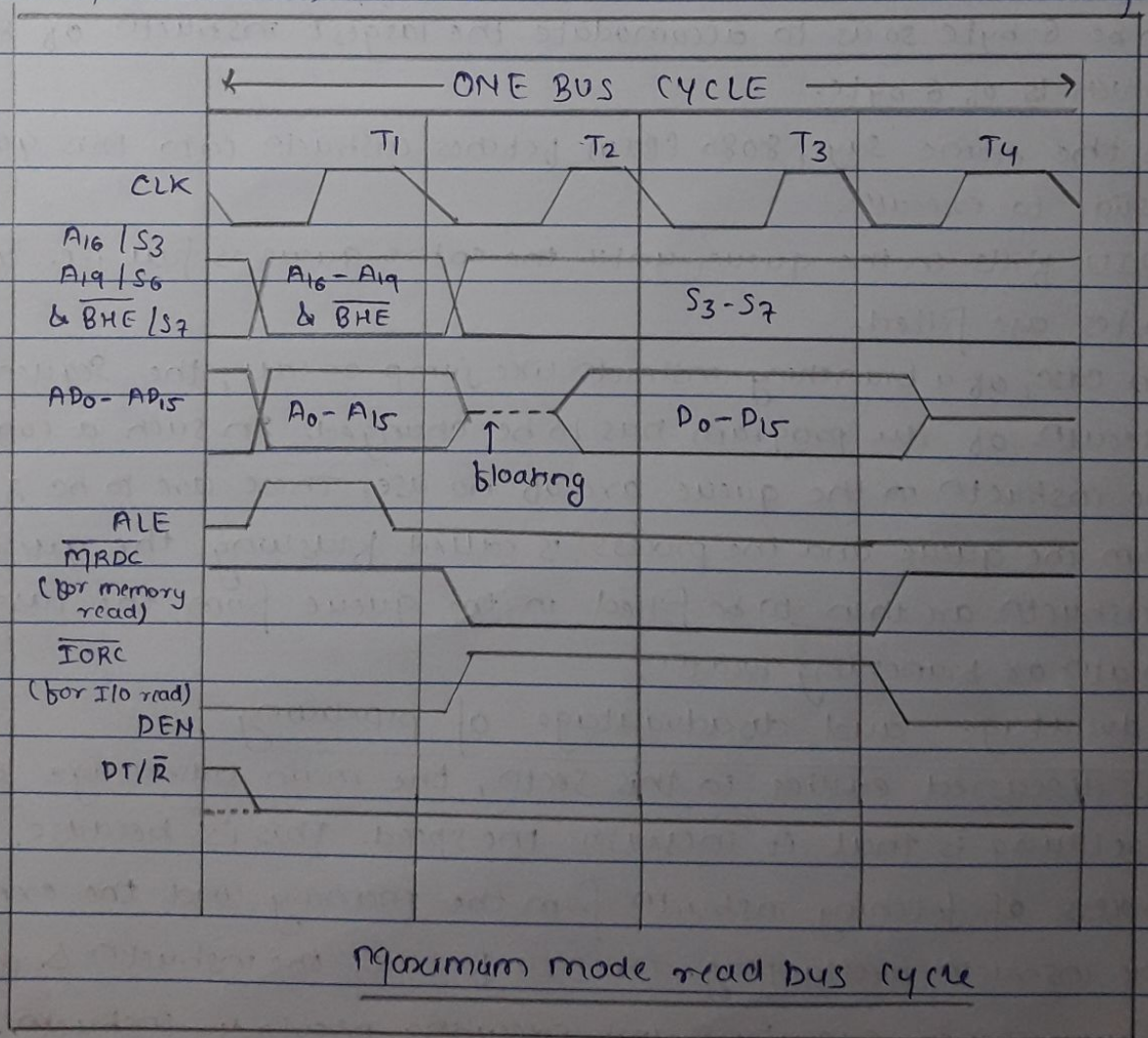
• Advantage and disadvantage of pipelining

- As discussed earlier in this sect^n, the main advantage of pipelining is that it increases the speed. This is because, the process of fetching instruct^n from the memory and the execut^n of the instruct^n overlapping. The BIU fetches the instruct^n & puts them in queue when executing unit executing previous instruct^n.

- The only disadvantage of pipelining is that there is requirement of extra hardware in the processor. These are some hardware units that are required for fetching the instruction as well as executing unit. These units are implemented doubly for both fetching unit as well as execution unit, thereby increasing overall cost of the processor.

2. Draw timing diagram for Read operation in maximum mode of 8086 microprocessor.

Ans:- fig. shows read cycle in a maximum mode of 8086. The cycle is identical to read cycle in 8086 in minimum mode of operation. The few differences we have those are as follows:



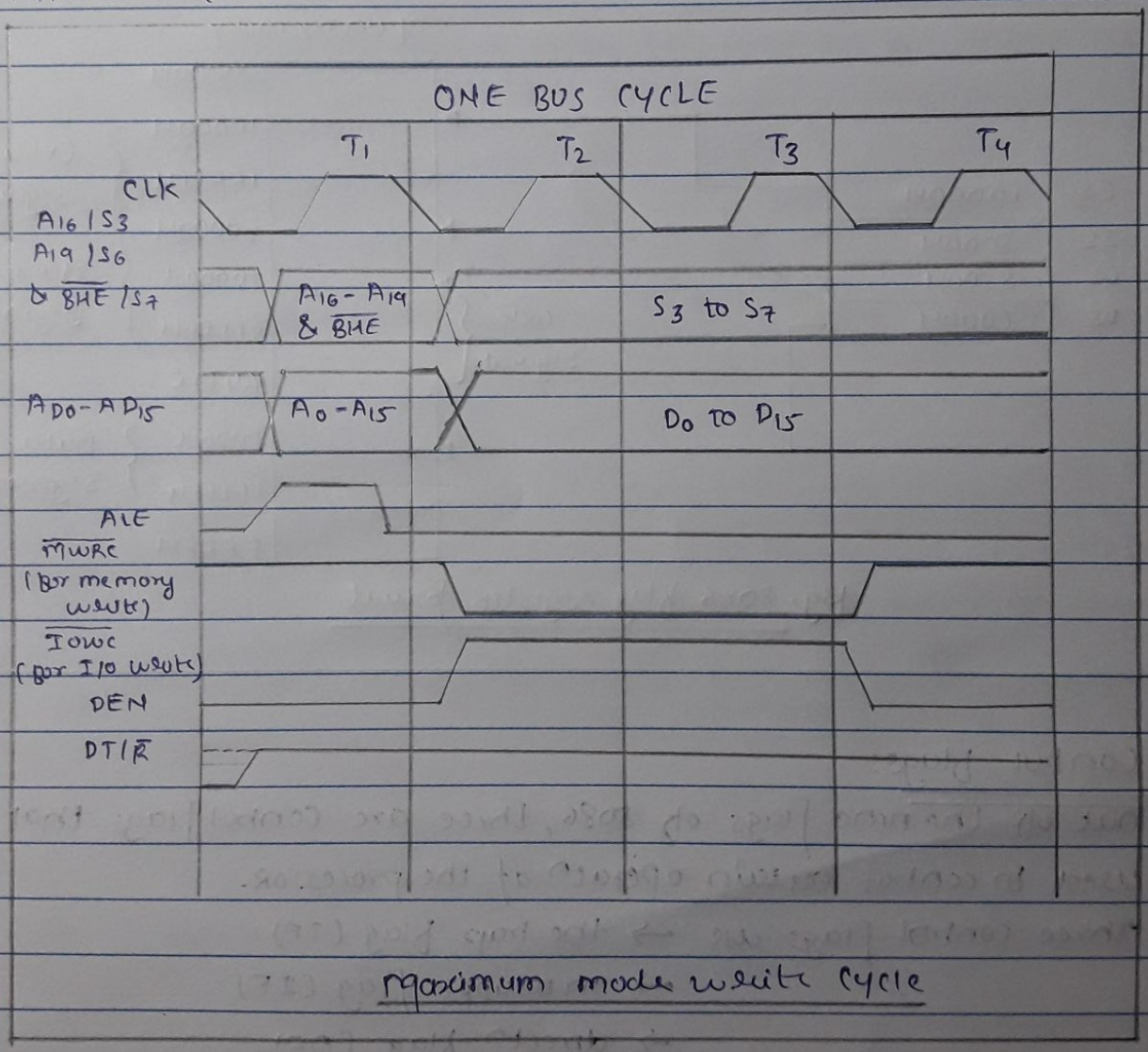
Maximum mode read bus cycle

1) Status line $\overline{S_2}$ $\overline{S_0}$ lines are taken into account. These lines are active for T_1 and T_2 cycle. After that they are inactive.

2) ALE, Memory Read, I/O read, DT/$\overline{R}$ read, DEN are generated by 8288 bus controller. They aren't generated by µP directly.

Q. Draw timing diagram for write operation in maximum mode of 8086 microprocessor.

Ans:- WRITE CYCLE



Maximum mode write cycle

- write cycle of 8086 in maximum mode is again similar to write cycle in minimum mode.

4. Draw and explain flag registers of 8086 microprocessor.

Ans:- - It is a set of flip-flops that gives the presence or absence of a particular attribute. The flip-flops together make a 16-bit register.

- Although there only 9 flip-flops, but they are arranged in such a form so as to make a 16-bit register. Hence, 7 are unused while only 9 are used.

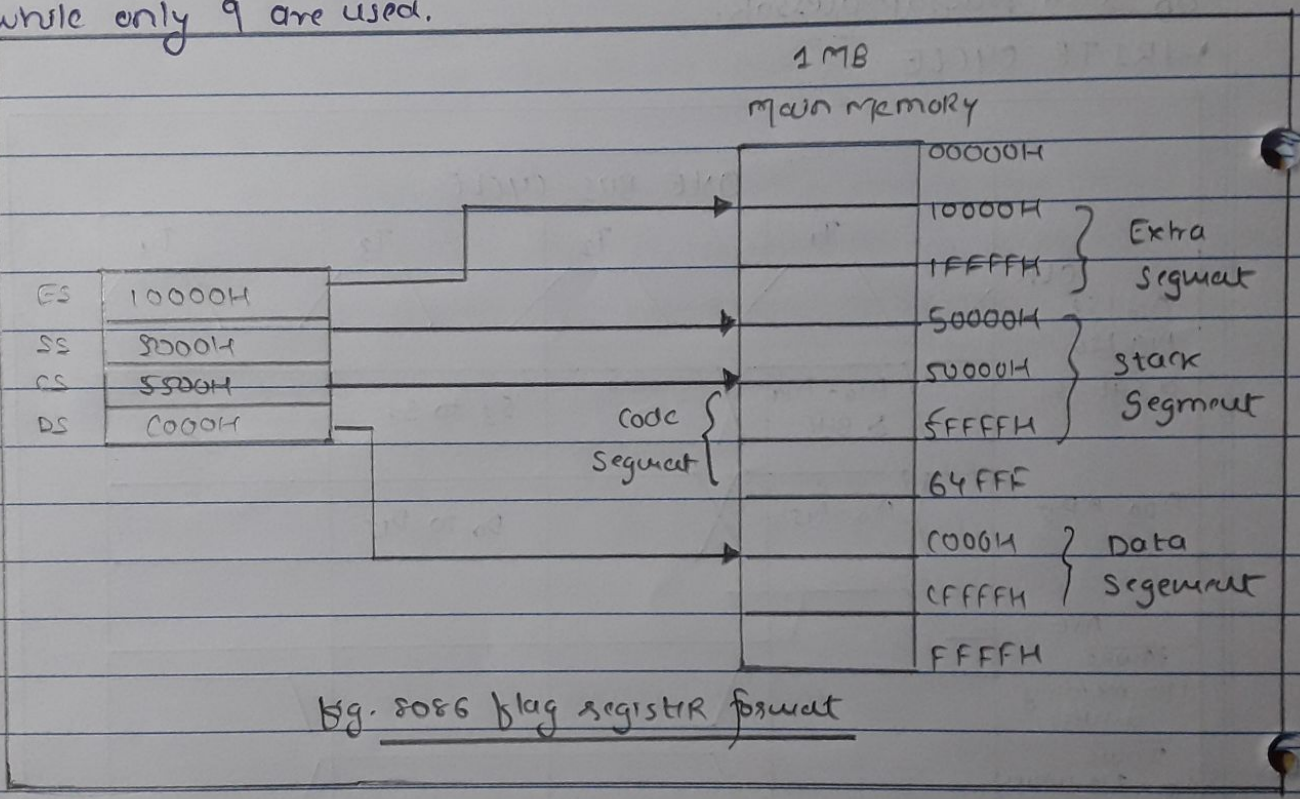


fig. 8086 flag register format

- Control flags:

Out of the nine flags of 8086, three are control flags that are used to control certain operation of the processor.

Three control flags are → the trap flag (TF)
→ interrupt flag (IF)
→ direction flag (DF)

1) The trap flag (TF): this flag is used for debugging a program. To see the stepwise execution of a program, this flag should be set to '1'.

TF = '1' - single stepping on,
TF = '0' - single stepping off

2) The interrupt flag (IF): 8086 has maskable as well as non-maskeable interrupts. Interrupt Enable flag is used to enable or disable the maskable interrupt i.e. INTR
IF = '1' - INTR enabled
IF = '0' - INTR disabled

3) The Direction flag (DF): 8086 supports string instructn. These are a special type of instructn that can work on an entire array of data. DF = '1' - Up, DF = '0' - Down.

- Status flags: Thus we have discussed three control flags, the remaining six are called as status flag or conditional flags.

The six conditional flags are:

↳ parity flag
↳ zero flag
↳ sign flag
↳ Auxiliary flag
↳ carry flag
↳ overflow flag.

1) The parity flag (PF): Refers to number of '1's in a data.
PF = '1' - indicates that the lower byte has an even parity
PF = '0' - indicates that the lower byte has odd parity.

2) The zero flag (ZF): This flag indicates whether the ALU result is zero or non-zero.
ZF = '1' - indicates that the result is 'zero
ZF = '0' - indicates that the result is non-zero.

3) The sign register (SF): This flag is set, when MSB (most significant bit of the result is 1.
SF = '1' - MSB is 1 (-ve for signed operatn)
SF = '0' - MSB is 0 (+ve for signed operatn)

4) The Auxiliary carry flag (AF): Carry from lower digit to upper digit.

AF = '1' = carry out from bit 3 on addtn or borrow into bit 3 on substractn.
AF = '0' - no carry out from bit 3 on addtn or borrow into bit 3 on substractn.

5) The carry flag (CF) : also be called as a final carry.
CF = '1' - indicates that there is carry generated by MSB,
CF = '0' - no carry out from MSB.

6) The overflow flag (OF) : it indicates an overflow from the magnitude to the sign bit of result.
OF = '1' - indicates overflow occured in case it is signed operatn.
OF = '0' - indicates that no overflow occured.

- Hence, we can say for the following bit structure of data, the flags will be affected as indicated in the Table. 4.

D_{15}	D_{14}	D_{13}	D_{12}	D_{11}	D_{10}	D_9	D_8	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0

Table 4.

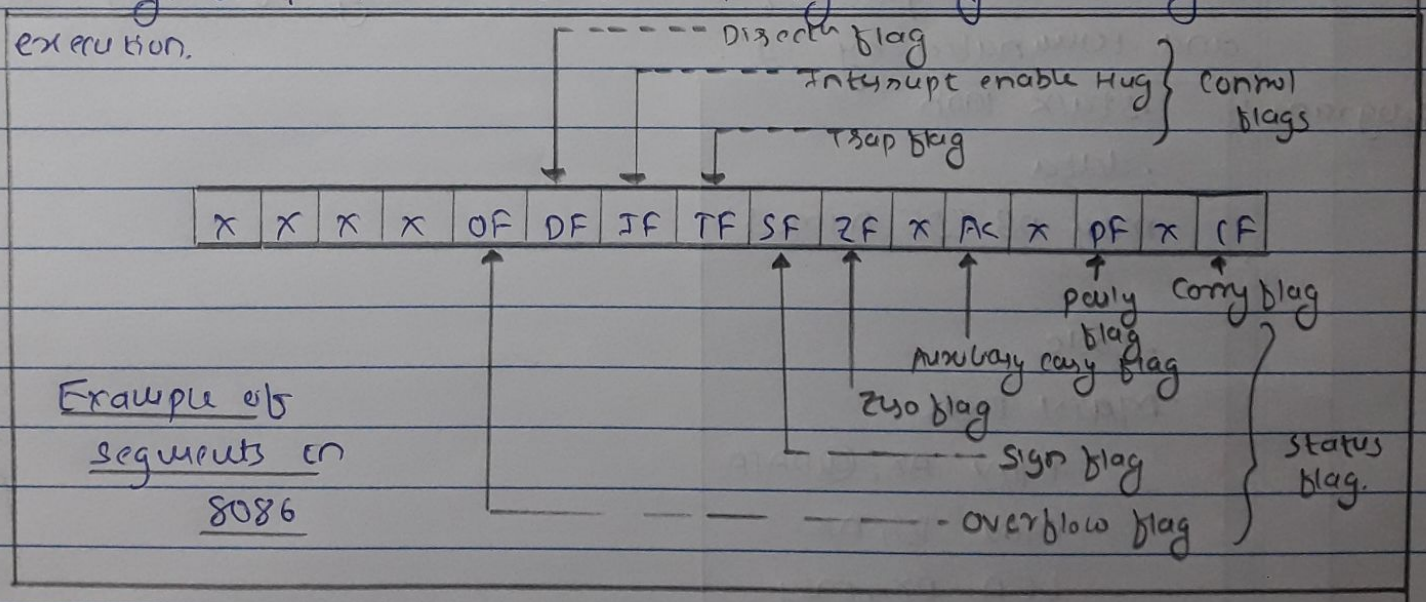
flag name	8-bit operatn	16-Bit operatn
Auxiliary carry	D_3 to D_4	D_3 to D_4
Carry	D_7 to D_8	D_{15} to D_{16}
overflow	D_6 to D_7	D_{14} to D_{15}
sign	Copy of D_7	Copy of D_{15}

5. Explain Memory Segmentatn of 8086 microprocessor with its advantages.

Ans:- - 8086 has 20-bit address bus while the registers are of 16-bit To access a memory locatn, we thus need to provide 20-bit address while the registers are 16-bit, this is made possible using segmentatn.
- Segmentatn 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 segment of the memory using the 16-Bit address or pointer registers.

- There are four registers that point to the beginning of a segment and are called as segment registers. They are:
  1. The code Segment (CS) Registry
  2. The stack segment (SS) Registry
  3. The Extra Segment (ES) Registry and
  4. The Data segment (DS) Registry.
- As per the names code Segment CSS points to the beginning of specific Segments. Code segment (that stores the program or codes) stack Segment (SS) points to the beginning of stack Segment Data Segment (DS) and Extra segment (ES) are used to point to data segments, where in ES is used only during the string instruction execution.



Important points

- In non-overlapping, there can be 16 segments in all, each of 64KB (since, 1MB/64KB = 16)
- But, the segments can overlap each other. Hence there can be many segments.
- A segment can begin at any location with the only condition that the

starting address must be a multiple of 10H.
- using segment override prefix, one can change the above default segment registry assignments.

• Advantages of segmentation:
- The most important advantage is that the programmer can access a memory that required 20-bit address, by using 16-bit registry only.
- The programs, data and stack are stored in separate blocks in memory and hence the three are organized in modular fashion.
- It also helps in object oriented programming to store data of object.
- Finally the sharing of data or passing of data from one program to another is easily possible due to segmentation.

6. Write a program to display message "SE AIML" on IBM PC. Use INT 21H function, AH=09 with string of message at DS:DX and terminated by "$".

Program:

```
.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
```

7. Write an assembly language program to determine number of positive and 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 :

8. Write a short note on string instruction of 8086 microprocessor.

Ans:-
- 8086 provides special instruction which performs some string related activities.
- We have five Basic string operation, called primitive or string primitives. These primitives allow string to bytes or words to be operated on, one element (byte or word) at a time.
- String of upto 64K bytes may be manipulated with these instruction
- Instruction are available to move, compare and scan for a values, as well as for moving string elements to and from the accumulator.
- The five basic operation primitives (MOV, compare, scan, load and store), may appear in one of the following form:
  1: operation B OR 2: operation W.
  The first and second form explicitly determines B (byte) and W(word) operation respectively.
- The string instruction may have a: 1. source operand.
  2. Destination operand 3. or both (source & destination)
- It is assumed that source string resides in the current data.
- A destination string must be in the current extra segment.

REP	Repeat
REPE / REPZ	Repeat while equal / zero
REPNE / REPNZ	Repeat while not equal / not zero
MOVSB / MOVSW	Move byte or word string
CMPSB / CMPSW	Compare byte or word string
SCASB / SCASW	Scan byte or word string
LODSB / LODSW	Load byte or word string
STOSB / STOSW	Store byte or word string

String Instruction

- SI, DI, CX, AL/AX, DF, DF, ZF...

9. Differentiate between procedure and macros.

Difference:

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

10. Explain the operation of the following instruction:

a. PUSHF b. LOOP c. JNZ address

d. XCHG e. CLD f. NOP

Ans:- a. PUSH F

- Push value of Flag Register into stack and decrement the stack value points by 2.

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

b. LOOP

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

```
Eg :  MOV CX, 4014
      MOV AL, BL
      ADD AL, BL
       :
      MOV BL, AL
      LOOP BACK     ; DO CX←CX-1
                    ; GO to BACK if CX not equal to 0.
```

c. JNZ address

- Transfer execution control to address 'Label', if ZF=0 then Jump.

d. XCHG

- Exchange : source and destination
  source - Registers, Memory location
  Destination - Registers, Memory location
  Even here both operands cannot be memory location.

```
Eg :  XCHG  CX, BX   ;  CX ↔ BX
      XCHG  BL, CH   ;  BL ↔ CH
```

e. CLD

- clears direction flag, no other flags are affected.

f. NOP

- No operation performed while executing
- 8086 requires 3T-states for this instruction
- insert time delays, and can also be used while debugging.