Md. Nur-E-Azam
ID: 1512268042

Sub :

Day

Time : Date : / /

LAB-4 Homework: 04

8086 INSTRUCTION SET :

① mov - mov Destination, Source

The 'mov' instruction copies a word or byte of data from a specified source to a specified source to a specified desination. The destination can be a a register or a memory location. The source can be a register, a memory location or an immediate number.

```
mov CX, 037AH
mov BL, [437AH]
mov AX, BX
mov DL, [BX]
mov DS, BX
mov RESULT[BP], AX
```

put immediate number 037AH to CX
copy byte in DS at offset 437AH to
copy content of register BX to AX
copy byte from memory at [BX] to
Copy Word from BX to DS register
copy AX to two memory location

Sub :

Day

Time : Date : / /

MOV ES: RESULTS[BP], AX same as the above instruction, but physical address = EA+ES because of the segment override prefix ES

LEA - LEA Register, Source

This instruction determines the offset of the variable or memory location named as the source and puts this offset in the indicated 16-bit register. LEA does not affect any flag.

LEA BX, PRICES Load BX with offset of PRICE in DS

LEA BP, SS: STACK-TOP Load BP with offset of STACK-TOP in SS

LEA CX, [BX][DI] Load CX with EA = [BX] + [DI]

Sub : Day Time : Date : / /

ADD - ADD Destination, Source

ADC - ADC Destination, Source

ADD AL, 74 H — Add immediate number 74H to content of AL, Result in AL

ADC CL, BL — Add content of BL plus carry status to contents of CL

ADD DX, BX — Add content of BX to content of DX

ADD DX, [SI] — Add word from memory at offset [SI] in DS to content of DX Add byte from effective address ~~PRICES [BX]~~

ADC AL, PRICES [BX] — Add byte from effective address PRICES [BX]

ADD AL, PRICES [BX] — Add content of memory at effective address PRICES [BX] to AL

Sub :

Day

Time : Date : / /

SUB - SUB Destination, Source

SUB - SBB Destination, Source

SUB CX, BX — CX - BX; Result in CX

SBB CH, AL — Subtract content of AL and content of CF from content of CH. Result in CH

SUB AX, 3427H — subtract immediate number 3427H from AX

SBB BX, [3427H] — subtract word at displacement 3427H in DS and content of CF

SUB PRICES [BX], 04H — subtract 04 from byte at effective address PRICES[BX]

SBB CX, TABLE [BX] — Subtract word from effective address TABLE [BX] and status of CF from CX.

SBB TABLE [BX], CX — Subtract CX and status of CF from word in memory at

Sub : _____ Day | | | | | | | |

Time : Date : / /

MUL – MUL source

```
MUL BH
MUL CX
MUL BYTE PTR [BX]
MUL FACTOR [BX]
MOV AX, MCAND_16
MOV CL, MPLIER_8
MOV CH, 00H
MUL CH
```

DIV – DIV source

DIV BL — Divide word in AX by byte in BL; Quotient in AL, remainder in AH

Div CX — Divide down word in DX and AX by word in CX; Quotient in AX, and remainder in DX

DIV SCALE [BX] — AX / byte at effective address SCALE [BX] if SCALE [BX] is of

Sub :

Day

Time : Date : / /

INC - INC Destination :

```
INC BL
INC CX
INC BYTE PTR [BX]
INC WORD PTR [BX]
INC TEMP
INC PRICES [BX]
```

DEC - DEC Destination

```
DEC CL
DEC BP
DEC BYTE PTR [BX]
DEC WORD PTR [BP]
DEC COUNT
```

Sub : _____ Day | | | | | | | |

Time : Date : / /

DAA (DECIMAL ADJUST AFTER BCD ADDITION)

Let AL = 59 BCD, and BL = 35 BCD

ADD AL, BL — AL = 8E H; Lower nibble > 9, add 06H to AL

DAA — AL = 94 BCD, CF = 0

Let AL = 88 BCD, and BL = 49 BCD

ADD AL, BL — AL = D1H; AF = 1, add 60H to AL

AL = 37 BCD, CF = 1

The DAA instruction updates AF, CF, SF, PF, and ZF ; but OF is undefined.

AAA (ASCII ADJUST FOR ADDITION)

Let AL = 0011 0101 (ASCII 5), and BL = 0011 1001 (ASCII 9)

ADD AL, BL — AL = 0110 1110 (6EH, which is incorr BCD

AAA — CF = 1 indicates answer is 14 decimal

The AAA instruction works only on the AL register. The AAA instruction updates AF and CF; but OF, PF, SF and ZF are left undefined.

Sub :

Day

Time : Date : / /

LOGICAL INSTRUCTIONS

AND - AND Destination, Source

```
AND CX, [SI]
AND BH, CL
AND BX, 00FFH
```

OR - OR Destination, Source

```
OR AH, CL
OR BP, SI
OR SI, BP
OR BL, 80H
OR CX, TABLE [SI]
```

XOR - XOR Destination, Source

```
XOR CL, BH
XOR BP, 01
XOR WORD PTR [BX], 00FFH
```

NOT - NOT Destination

```
NOT BX
NOT BYTE PTR [BX]
```

Sub: _____ Day | Time: Date: / /

CMP - CMP Destination, Source

	CF	ZF	SF	
CX = BX	0	1	0	Result of sub is 0
CX > BX	0	0	0	No borrow req, so CF=
CX < BX	1	0	1	Subtraction req borrow, so CF

CMP AL, 01H — compare immediate number 01H with byte in AL

CMP BH, CL — compare byte in CL with byte in BH

CMP CX, TEMP — compare word in DS at

CMP PRICES[BX], 40H

TEST - TEST Destination, Source

TEST AL, BH — AND BH with AL. No result stored

TEST CX, 0001H — AND CX with immediate

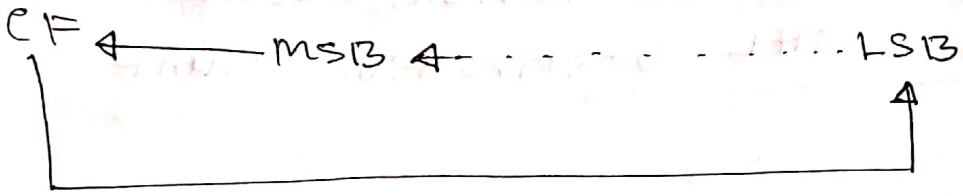
TEST BP, [BX][DI]

Sub :

Day

Time : Date : / /

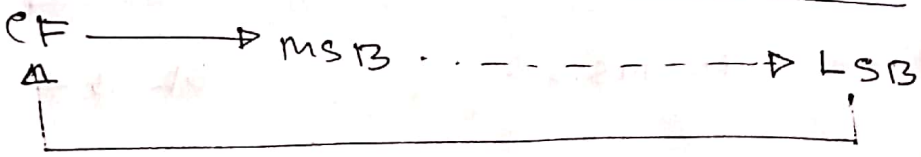
RCL – RCL Destination, count



For multi-bit rotates, CF will contain the bit most recently rotated out of the MSB.

```
RCL DX, 1          word in DX 1 bit left, MSB to CF. CF to LSB
MOV CL, 4
RCL SUM[BX], CL
```

RCR – RCR Destination, count



For multi-bit rotate, CF will contain the bit most recently rotated out of the LSB.

```
RCR BX, 1
MOV CL, 4
RCR BYTE PTR [BX], 4
```

Sub :

Day

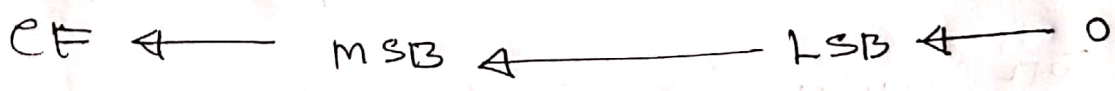
Time :

Date : / /

ROL - ROL Destination, count

SAL - SAL Destination, count

SHL - SHL Destination, count



~~SAR - SAR Destination, count~~

```
SAL BX, 1
MOV CL, 02h
SAL BP, CL
SAL BYTE PTR [BX], 1
```

SAR - SAR Destination, count


```
SAR DX, 1
MOV CL, 02H
SAR WORD PTR [BP], CL
```

Sub :

Day

Time : Date : / /

SHR - SHR Destination, count

0 ⟶ MSB . . . . . ⟶ LSB ⟶ CF

SHR BP, 1

MOV CL, 03H

SHR BYTE PTR [BX]

JUMP (UNCONDITIONAL JUMP TO SPECIFIED DESTINATION :

JMP CONTINUE

JMP BX

JMP WORD PTR [BX]

JMP DWORD PTR [SI]

JBE / JNA (JUMP IF BELOW OR EQUAL / JUMP IF NOT ABOVE :

CMP AX, 4371H

JBE NEXT

CMP AX, 4371H

JNA NEXT

Sub : Day Time : Date : / /

JG/ JNLE (JUMP IF GREATER/JUMP IF NOT LESS THAN OR EQUAL

```
CMP BL, 39H
JG NEXT

CMP BL, 39H
JNLE NEXT
```

JL/ JNGE (JUMP IF LESS THAN/ JUMP IFI NOT GREATER THAN OR EQUAL

```
CMP BL, 39H
JL AGAIN
CMP BL, 39H
JNGE AGAIN
```

JLE/ JNG

```
CMP BL, 39H
JLE NEXT
CMP BL, 39H
JNG NEXT
```

Sub :

Day

Time : Date : / /

JE / JZ

```
CMP BX, DX
JE DONE
IN AL, 30H
SUB AL, 30H
JZ START
```

JNE / JNZ

```
IN AL, 0F8H
CMP AL, 72
JNE NEXT
ADD AX, 0002H
DEC BX
JNZ NEXT
```

Sub: _____ Day | Time: Date: / /

STACK RELATED INSTRUCTIONS

PUSH – PUSH source

```
PUSH  BX
PUSH  DS
PUSH  BL
PUSH  TABLE[BX]
```

POP – POP Destination

```
POP  DX
POP  DS
POP  TABLE[BX]
```

IN – IN Accumulator, port

```
IN   AL, 0C8H    Input a byte from port 0C8H to AL
IN   AX, 34H     Input a word from port 34H to AX

MOV  DX, 0FF78H
IN   AL, DX
IN   AX, DX
```

Sub :

Day

Time : Date : / /

OUT - OUT port, Accumulator

The out instruction copies a byte from AL or a word from AX to the specified port. The OUT instruction has two possible forms, fixed port and variable port.

```
OUT   313H, AL
OUT   2CH, AX
MOV   BX, 0FFF8H
OUT   DX, AL
OUT   DX, AX
```

The OUT instruction does not affect any flag.

Sub : _____ Day | | | | | | | |

Time : Date : / /

ENDS (END SEGMENT)

This directive is used with the name of a segment to indicate the end of that logical segment.

```
CODE SEGMENT
CODE ENDS
```

END (END PROCEDURE)

The END directive is put after the last statement of a program to tell the assembler that this is the end of the program module. The assembler will ignore any statements after an END directive, so you should make sure to use only END directive at the very of your program module. A carriage return is required after the END directive.

Sub :

Day

Time : Date : / /

DW (DEFINE WORD)

WORD DW 12341H, 3456H

STORAGE DW 100 DUP(0)

STORAGE DW 100 DUP(?)

ENDP (END PROCEDURE)

The directive is used along with the name of the procedure to indicate the end of a procedure to the assembler. The directive, together with the procedure directive, PROC, is used to "bracket" a procedure.

SQUARE_ROOT PROC start of procedu

SQUARE_ROOT ENDP End of procedur

Sub : _____ Day | Time : Date : / /

LABEL :

```
ENTRY_POINT LABEL FAR
NEXT: MOV AL, BL
```

The following example shows how we use the label directive for a data reference.

```
STACK_SEG SEGMENT STACK
DW 100 DUP(0)
STACK_TOP LABEL WORD
STACK_SEG ENDS
```

INCLUDE (INCLUDE SOURCE CODE FROM FILE)

This directive is used to tell the assembler to insert a block of source code from the named file into the current source module