Addition :

Label	Mnemonics	Comments:
	MOV AH, [2000]	AH ← [2000]
	MOV BH, [2001]	BH ← [2001]
	MOV CH, 00	CH ← 00
	ADD AH, BH	AH ← AH + BH
	JNC label	Jump if CF ≠ 1
	INC CH	CH ← CH + 1
label	MOV [2002], AH	[2002] ← AH
	MOV [2003], CH	[2003] ← CH
	HLT	Halt the program.

Subtraction

Label	Mnemonics	Comments.
	MOV AH, [2000]	AH ← [2000]
	MOV BH, [2001]	BH ← [2001]
	MOV CH, 00	CH ← 00
	SUB AH, BH	AH ← AH − BH
	JNC label	jump if CF ≠ 1
	INC CH	CH ← CH + 1
	NEG AH	2's complement of AH
label	MOV [2002], AH	[2002] ← AH
	MOV [2003], CH	[2003] ← CH
	HLT	Halt the program.

Assignment - 1 | 23/08/2022

8 BIT ARITHMETIC OPERATIONS IN 8086

Aim: To perform basic 8-bit arithmetic operations such as addition, subtraction, multiplication and division in 8086 microprocessor.

Apparatus required: 8086 microprocessor kit

Algorithms:

1) **Addition:**
   i. Move the contents in 2000 to AH
   ii. Move the contents in 2001 to BH
   iii. Move the value 00 to CH
   iv. Add the contents of AH and BH and store it in AH
   v. Jump to (vii) if carry is not set
   vi. Increment CH
   vii. Move the contents of AH to 2002 location
   viii. Move the contents of BH to 2003 location
   ix. Halt the program.

Sample input and output:

Input:	[2000]-	19	[2000]-	87
	[2001]-	66	[2001]-	99
Output	[2002]-	7F	[2002]-	20
	[2003]-	00	[2003]-	01

2) **Subtraction:**
   i. Move the contents in 2000 to AH
   ii. Move the contents in 2001 to BH
   iii. Move the value 00 to CH
   iv. Subtract BH from AH and store in AH
   v. Jump to (viii) if carry is not set
   vi. Increment CH
   vii. Takes 2's complement of AH
   viii. Move the contents of AH to 2002 location
   ix Move the contents of BH to 2003 location.
   x. Halt the program.

Sample Input and Output:

Input:	[2000]-	59	[2000] -	21
	[2001]-	40	[2001] -	55
Output	[2002]-	19	[2002] -	34
	[2003]-	00	[2003] -	01

Multiplication

Label	Mnemonics	Comments:
	MOV AL, [2000]	AL ← [2000]
	MOV BL, [2001]	BL ← [2001]
	MUL BL	AX ← AL * BL
	MOV [2002], AX	[2002] ← AX
	HLT	Halt the program.

Division.

Label	Mnemonics	Comments:
	MOV AH, 00	AH ← 00
	MOV AL, [2000]	AL ← [2000]
	MOV BL, [2001]	BL ← [2001]
	DIV BL	AX ← AX ÷ BL
	MOV [2002], AX	[2002] ← AX
		AL ← quotient
		AH ← remainder
	HLT	Halt the program.

3) Multiplication

i. Move the contents of 2000 to AL
ii. Move the contents of 2001 to BL
iii. Multiply BL with AL and store it in AX
iv. Move the contents of AX to 2002
v. Halt the program.

Sample Input and Output:

Input	[2000] –	29	[2000] –	FF
	[2001] –	46	[2001] –	FF
Output	[2002] –	36	[2002] –	01
	[2003] –	0B	[2003] –	FE

4) Division:

i. Move the contents of 2000 to AL
ii. Move the contents of 2001 to BL
iii. Move 00 to AH
iv. Divide BH from AX
v. Move the contents of AX to [2002]
vi. Halt the program.

Sample Input and Output

Input:	[2000] –	FF	[2000] –	81
	[2001] –	FF	[2001] –	09
Output:	[2002] –	01	[2002] –	06
	[2003] –	00	[2003] –	03

Result: The 8 bit arithmetic operations have been implemented in 8086

Addition.

Label	Mnemonics	comments.
	MOV AX, [2010]	AX ← [2010]
	MOV BX, [2012]	BX ← [2012]
	MOV CH, 00	CH ← 00
	ADD AX, BX	AX ← AX + BX
	JNC label	jump if no carry
	INC CH	CH ← CH + 1
label	MOV [2014], AX	[2014] ← AX
	MOV [2016], CH	[2016] ← CH
	HLT	

Subtraction

Label	Mnemonics	Comment
	MOV AX, [2010]	AX ← [2010]
	MOV BX, [2012]	BX ← [2012]
	MOV CH, 00	CH ← 00
	SUB AX, BX	AX ← AX − BX
	JNC label	Jump if not carry
	INC CH	CH ← CH + 1
	NEG AX	2's complement of AX
label	MOV [2014], AX	[2014] ← AX
	MOV [2016], CH	[2016] ← CH
	HLT	

Assignment - 2

16 - BIT ARITHMETIC OPERATIONS IN 8086

Aim: To perform 16 bit arithmetic operations like addition, subtraction, multiplication, division on 8086 microprocessor

Aparatus required: 8086 microprocessor kit

Algorithms:

1) **Addition:**
   i. Transfer the value in 2010 to AX and 2012 to BX
   ii. Initialize CH to 00H
   iii Add AX, BX and result is stored in AX
   iv. In case of carry increment CH
   v. Transfer the value in AX to 2014 and CH to 2016
   vi) Halt the program

Sample Input and Output

Input		Output	
	[2010] - A5		[2014] - C5
	[2011] - 10		[2015] - 11
	[2012] - 20		[2016] - 00
	[2013] - 01		

Input		Output	
	[2010] - 53		[2014] - 15
	[2011] - A6		[2015] - 1C
	[2012] - C2		[2016] - 01
	[2013] - 75		

2) **Subtraction**
   i. Transfer the value in 2010 to AX and 2012 to BX
   ii. Initialize CH to 00H
   iii. Subtract BX from AX and store in AX
   iv. If in case of borrow increment CH and do a 2's complement of AX
   v. Halt the program after transfering the value in AX to 2014 and value in CH to 2016

Sample Input and Output:

Input		Output:	
	[2010] - 60		[2014] - F6
	[2011] - C5		[2015] - 71
	[2012] - 6A		[2016] - 00
	[2013] - 53		

Multiplication

Label	Mnemonics	Comments.
	MOV AX, [2010]	AX ← [2010]
	MOV BX, [2012]	BX ← [2012]
	MUL BX	DX AX ← AX • BX
	MOV [2014], AX	[2014] ← AX
	MOV [2016], DX	[2016] ← DX
	HLT	

Division.

Label	Mnemonics	Comments.
	MOV AX, [2010]	AX ← [2010]
	MOV BX, [2012]	BX ← [2012]
	DIV BX	DX. AX ← AX ÷ BX
	MOV [2014], AX	[2014] ← AX
	MOV [2016], DX	[2016] ← DX
	HLT	

Input : [2010] - 2B
[2011] - 46
[2012] - 01
[2013] - D4

Output : [2014] - 06
[2015] - 80
[2016] - 01

3) Multiplication :
i. Transfer the value in 2010 to AX and 2012 to BX
ii. Multiply AX by BX
iii Transfer the value in AX to 2014 and DX to 2016
iv. Halt the program.

Sample I/O :
Input : [2010] - AC
[2011] - 32
[2012] - 11
[2013] - 9B

Output : [2014] - 6C
[2015] - 41
[2016] - 86
[2017] - 08

Input [2010] - 05
[2011] - 00
[2012] - 09
[2013] - 00

Output : [2014] - 0A
[2015] - 00
[2016] - 00
[2017] - 00

4) Division :
i. Transfer the value in 2010 to AX and 2012 to BB
ii. Divide AX by BX store quotient in AX and remainder in DX
iii. Transfer the value in AX to 2014 and DX to 2016
iv. Halt the program

Sample I/O :
Input : [2010] - 00
[2011] - 5B
[2012] - C0
[2013] - 12

Output : [2014] - 04
[2015] - 00
[2016] - 00
[2017] - 10

Result : Hence 16 bit arithmetic operations have been performed using 8086 microprocessor

Move a string of bytes.

Label	Mnemonics	Comments.
	MOV CX, [2200]	CX ← [2200]
	MOV SI, 2000	SI ← 2000
	MOV DI, 2100	DI ← 2100
	CLD	Clear direction flag
	REP MOVSB	If CX != 0 the repeat MOVSB else exit
	HLT	

Compare a string.

Label	Mnemonics	Comments.
	MOV CX, [2200]	CX ← [2200]
	MOV SI, 2000	SI ← 2000
	MOV DI, 2100	DI ← 2100
	CLD	
	REPE CMPSB	If CX != 0 and ZF = 1 repeat CMPSB
	JZ label	jump to label if ZF = 1
	INC CX	CX ← CX + 1
label	MOV [2202], CX	[2202] ← CX
	HLT	

Assignment - 3

STRING OPERATIONS

Aim: To perform string manipulations using REP instructions in 8086 microprocessor

Apparatus required: 8086 microprocessor

Algorithms:

1) Moving a string of bytes.
   i. Initialise the pointer SI to 2000
   ii. Initialise the pointer DI to 2100
   iii. Initialise the CX count register to no. of bytes [2200]
   iv. Clear the direction flag.
   v. Repeat move string bytes until CX=0
   vi Halt the program.

Sample I/O:

Before execution: [2000] = AB, [2001] = CD, [2200] / CX = 02, [2100] = 00, [2101] = 00

After execution: [2000] = AB, [2001] = CD, [2100] = AB, [2101] = CD

2) Compare 2 strings.
   i. Initialise the pointer SI to 2000
   ii. Initialise the pointer DI to 2100
   iii. Initialise CX to value in 2200
   iv. Clear direction flag.
   v. Repeat compare string byte if CX!=0 and ZF=1
   vi. If ZF!=0 increment CX
   vii. Move the value in CX to 2202
   viii Halt the program.

Sample I/O:

Before execution: [2000] - AB, [2002] - CD, [2200] - 02, [2100] - AB, [2101] - AB

After execution: [2202] - 01

Scan a string byte.

Label	Mnemonics	Comments.
	MOV CX,[2200]	CX ← [2202]
	MOV AL, [2000]	AL ← [2000]
	MOV DL, 2100	DI ← 2100
	MOV SI, 2000	SI ← 2000
	CLD	
	REPNE SCASB	If CX != 0 and ZF = 0 then repeat SCASB
	JNZ label	
	INC CX	CX ← CX + 1
label	MOV [2202], CX	[2202], CX
	HLT	

Move a string without using string commands.

Label	Mnemonics	Comments.
	MOV SI,2000	SI ← 2000
	MOV DI, 2100	DI ← 2100
	MOV CX,[2200]	CX ← [2200]
label	MOV AL, [SI]	AL ← [SI]
	MOV [DI], AL	[DI] ← AL
	INC SI	SI ← SI + 1
	INC DI	DI ← DI + 1
	DEC CX	CX ← CX − 1
	JNZ label	jump if CX != 0
	HLT	

3) Scan a string byte

i. Initialize the CX register to value in 2200
ii. Initialise DI to 2100
iii. Initialise SI to 2000
iv. Clear the direction flag
v. Repeat scan string byte if CX != 0 and ZF = 0
vi Move the contents of CX to 2202
vii. Halt the program.

Sample I/O:

Before execution:

[2000] - BB
[2100] - AA
[2101] - BB
[2102] - CC
[2103] - DD
[2200] - 04

After execution:

[2202] - 03
[2203] - 00

4) Move a string byte without string commands.

i. Initialize CX with value in 2200
ii. Initialize the SI pointer with value 2000
iii. Initialize the DI pointer with value 2100
iv. Move the contents pointed by SI to AL
v. Move the contents pointed by AL to DI
vi. Increment SI and DI
vii. Decrement the count register CX
viii Jump to step 5 if ZF != 0 or CX != 0
ix. Halt the program.

Sample I/O:

Before execution:

[2000] - AA
[2001] - BB
[2002] - CC
[2200] - 03
[2201] - 00

After execution

[2100] - AA
[2101] - BB
[2102] - CC

Result: The string manipulating operations have been done.

Label	Mnemonics	Comments
	MOV AL, [2000]	AL ← [2000]
	MOV BL, AL	BL ← AL
	MOV AH, 00	AH ← 00
	AND BL, 0F	BL ← BL & 0F
	AND AL, 0F0	AL ← AL & F0
	MOV CL, 04	CL ← 04
	ROR, AL, CL	Rotate AL by 4 bit.
	MOV DL, 0A	DL ← 0A
	MUL DL	AX ← AL x DL
	ADD AL, DL	AL ← AL + DL
	MOV [2500], AL	[2500] ← AL
	HLT	

Assignment - 4

CODE CONVERSION

Aim: To perform BCD to hexadecimal and hexadecimal to BCD code conversion using 8086 microprocessor.

Apparatus required - 8086 microprocessor kit

Algorithm:

1) BCD → hexadecimal conversion.
   i. Move the contents of [2000] to AL
   ii. Move the contents of AL to BL
   iii Move the value "00" to AH
   iv. Bitwise AND the value in BL and "0F"
   v. Bitwise AND the value in AL and "0F0"
   vi. Move the value "04" to CL
   vii. Rotate the value in AL by 4 bit
   viii. Move the value "0A" to DL
   ix. Multiply DL with AL
   x Move the value in AL to [2500]
   xi Halt the program.

Sample I/O:

Input: [2000] - 12 [2000] - 21

Output: [2500] - 0C [2500] - 15

Label	Mnemonics	Comments.
	MOV AH,00	AH←00
	MOV AL, [2000]	AL←[2000]
	MOV BL, 64	BL←64
	DIV BL	AX←AX÷BL
	MOV DL, AL	DL←AL
	MOV AL, AH	AL←AH
	MOV AH, 00	AH←00
	MOV BL, 0A	BL←0A
	DIV BL	AX←AX÷BL
	MOV CL,04	CL←04
	ROR AL,CL	
	ADD AL,AH	AL←AL+AL
	MOV [2100],DL	[2100]←DL
	MOV [2101], AL	[2101]←AL
	HLT	

2) Hexadecimal → BCD conversion

i. Move the value "00" to AH
ii. Move the value in [2000] to AL
iii. Move the value "64" to BL
iv. Divide AX by BL
v. Move the value in AL to DL
vi. Move the value in AH to AL
vii. Move the value "00" to AH
viii Move "0A" to BL
ix Divide AX by BL
x Move the value "04" to CL
xi Rotate AL by 4 bits.
xii Add AL and AH
xiii Move the value in DL to [2100]
xiv Move the value in AL to [2101]
xv. Halt the program.

Sample I/O

Input : [2000] = FF [2000] = 0A

Output: [2100] = 02 [2100] = 00
[2101] = 55 [2101] = 10

Result : Thus a BCD code is converted to hexadecimal and hexadecimal code is converted to BCD equivalent using 8086 microprocessor

Label	Mnemonics	Comments.
	MOV AL,[2000]	AL ← [2000]
	MOV BL, [2001]	BL ← [2001]
	MOV CL, [2002]	CL ← [2002]
	MOV DL, [2003]	DL ← [2003]
	CMP AL, BL	compare AL and BL
	JNZ label1	
	CMP CL, BL	compare CL, BL
	JNZ label1	
	MUL DL	AX ← AL x DL
	MOV CX, AX	CX ← AX
	MOV SI, 3000	SI ← 3000
	MOV DI, 3500	DI ← 3500
	MOV BX, 3600	BX ← 3600
label2	MOV AL, [SI]	AL ← [SI]
	ADD AL, [DI]	AL ← AL + [DI]
	MOV [BX], AL	[BX] ← [BX] + AL
	INC SI	SI ← SI + 1
	INC DI	DI ← DI + 1
	INC BX	BX ← BX + 1
	LOOP label2	
label1	HLT	

Assignment -5

MATRIX OPERATIONS.

Aim: To perform matrix addition and subtraction using 8086 microprocessor

Apparatus required - 8086 microprocessor kit

Algorithm:

1) Matrix addition.

i. Move the contents of [2000] to AL
ii. Move the contents of [2001] to BL
iii. Move the contents of [2002] to CL
iv. Move the contents of [2003] to DL
v. Compare AL and BL
vi. If ZF = 0 then jump to exit
vii. Compare BL and DL
viii If ZF = 0 then jump to exit
ix. Multiply AL and DL
x Move Ax to CX
xi Move 3000 to SI
xii Move 3500 to DI
xiii. Move the value in SI to AL
xiv Add the value in AL to value of the memory pointed by DI
xv. Move the value in AL to the memory pointed by SI
xvi Increment SI and DI
xvii loop to step 13 until Cx = 0
xviii Halt the program.

Sample I/O:

Before execution:

[2000] = 02
[2001] = 02
[2002] = 02
[2003] = 02

[3000] = 04
[3001] = 03
[3002] = 02
[3003] = 01
[3500] = 01
[3501] = 02
[3502] = 03
[3503] = 04

After execution:

[3600] = 05
[3601] = 05
[3602] = 05
[3603] = 05

Label	Mnemonics	Comments.
	MOV AL, [2000]	AL ← [2000]
	MOV BL, [2001]	BL ← [2001]
	MOV BL, [2002]	CL ← [2002]
	MOV DL, [2003]	DL ← [2003]
	CMP AL, CL	
	JNZ label1	if ZF=0 jump
	CMP BL, DL	
	JNZ label2	if ZF=0 jump.
	MUL DL	AX ← AL X DL
	MOV CX, AX	CX ← AX
	MOV SI, 3000	SI ← 3000
	MOV DI, 3500	DI ← 3500
	MOV BX, 3600	BX ← 3600
label1	MOV AL, [SI]	AL ← [SI]
	SUB AL, [DI]	AL ← AL − [DI]
	MOV BX, [AL]	BX ← [AL]
	INC SI	SI ← SI + 1
	INC DI	DI ← DI + 1
	INC BX	BX ← BX + 1
	LOOP label2	
label1	HLT	

2) Matrix Subtraction:

i. Move the contents in [2000] to AL
ii. Move the contents in [2001] to BL
iii. Move the contents in [2002] to CL
iv. Move the contents in [2003] to DL
v. Compare AL and CL
vi. If ZF=0 the jump to exit
vii. Compare BL and DL
viii. If ZF=0 the jump to exit
ix Multiply AL with DL
x Move the value in AX to CX
xi. Move the value 3000 to SI
xii. Move the value 3500 to DL
xiii Move the value 3600 to BX
xiv Move the value pointed by SI to AL
xv. Subtract AL by the value pointed by DI
xvi. Increment SI, DI and BX
xvii. Loop to step [xiv] till count becomes 0
xix Halt the program.

Sample I/O:

Input: [2000] = 02 [3000] = 04 [3500] = 01
[2001] = 02 [3001] = 03 [3501] = 01
[2002] = 02 [3002] = 02 [3502] = 01
[2003] = 02 [3003] = 01 [3503] = 01

Output: [3600] = 03
[3601] = 02
[3602] = 01
[3603] = 00

Result: The matrix addition and subtraction has been implemented in 8086 microprocessor.

Label	Mnemonics	Comments.
	MOV DL, [3000]	DL ← [3000]
	DEC DL	DL ← DL − 1
L1 :	MOV SI, 2000	SI ← 2000
	MOV CL, DL	CL ← DL
L2:	MOV AL, [SI]	AL ← [SI]
	CMP AL, [SI+1]	compare AL with [SI+1]
	JNC label	
	XCHG [SI+1], AL	[SI+1] ↔ AL
	XCHG [SI], AL	[SI] ↔ AL
	INC SI	SI ← SI+1
label:	LOOP L2	goto label L2
	DEC DL	DL ← DL − 1
	JNZ L1	Jump to label L1
	HLT	Halt the program.

Assignment - 6

SORTING

Aim: To perform sorting in ascending and descending order using 8086 microprocessor.

Apparatus required: 8086 microprocessor kit

Algorithm:

Descending:

i. Move the number of elements in the list to DL register
ii. Decrement DL
iii. Copy DL to CL
iv. Load SI with the starting address of list of elements
v. Move the contents of [SI] to AL
vi. Compare AL with [SI+1]
vii Jump if no carry to step 10
viii Exchange AL, [SI]
ix Exchange AL, [SI+1]
x Increment the SI pointer
xi Loop to step 5 till CX=0
xii Decrement DL
xiii Jump to point 3 if DL != 0
xiv Halt the program.

Sample I/O:

Input: [3000] - 05
[3001] - 00
[2000] - 01
[2001] - 05
[2002] - 03
[2003] - 04
[2004] - 02

Output:

[2000] - 05
[2001] - 04
[2002] - 03
[2003] - 02
[2004] - 01

Label	Mnemonics	Comments :
	MOV DL, [2000]	DL ← [3000]
	DEC DL	DL ← DL - 1
L1:	MOV SI, 2000	SI ← 2000
	MOV CL, DL	CL ← DL
L2:	MOV AL, [SI]	AL ← [SI]
	CMP AL, [SI+1]	compare AL [SI+1]
	JC label	
	XCHG [SI+1], AL	[SI+1] ↔ AL
	XCHG [SI], AL	[SI] ↔ AL
label	INC SI	SI ← SI+1
	LOOP L2	
	DEC DL	DL ← DL - 1
	JNZ L1	
	HLT	Halt the program

Ascending:

i. Move the number of elements to DL
ii. Decrement DL
iii. Copy DL to CL
iv. Load SI with the starting address of the list
v. Move the contents of [SI] to AL
vi. Compare AL to [SI+1]
vii. If there is a carry i.e SI > SI+1 then goto step 9
viii. else exchange [SI+1] with [SI] using AL
ix increment SI pointer
x loop to step 5
xi decrement DL register
xii jump if CX is not zero.
xiii Halt the program.

Sample I/O:

Input: [3000] - 05
[3001] - 00
[2000] - 01
[2001] - 05
[2002] - 04
[2003] - 02
[2004] - 03

Output:

[2000] - 01
[2001] - 02
[2002] - 03
[2003] - 04
[2004] - 05

Result: Therefore sorting has been done on an array of elements in both ascending and descending order using 8086 microprocessor.

Label	Mnemonics	Comments:
	MOV AL,[2000]	AL ← [2000]
	MOV BL,[2001]	BL ← [2001]
	ADD AL, BL	AL ← AL + BL
	DAA	decimal adjust after addition.
	MOV [2002], AL	[2002] ← AL
	~~ADC~~MOV AL, 00	AL ← 00
	ADC AL,AL	AL ← AL + AL + CF
	MOV [2003], AL	[2003] ← AL
	HLT	halt the program.

Assignment – 7

BCD OPERATIONS (ADDITION and SUBTRACTION)

Aim: To perform BCD addition and subtraction using 8086 microprocessor.

Prerequisite: Store digits less than or equal to 9 only.

Apparatus required: 8086 microprocessor kit

Algorithm:

Addition

1. Move the contents of [2000] to AL
2. Move the contents of [2001] to BL
3. Add the contents of register AL and register BL and store the result in AL.
4. Decimally adjust AL after addition.
5. Move the contents of AL to [2002]
6. Move 00 to AL
7. Add the contents of register AL with carry and store the result in location [2003]

Sample I/O:

Input: [2000] – 15
[2001] – 17

Output: [2002] – 32
[2003] – 00

Label	Mnemonics	Comments.
	MOV AL, [2000]	AL ← [2000]
	MOV BL, [2001]	BL ← [2001]
	MOV CL, 00	CL ← 00
	SUB AL, BL	AL ← AL − BL
	DAS	decimal adjust after subtraction.
	JNC label	
	MOL DL, 99	DL ← 99
	SUB DL, AL	DL ← DL − AL
	MOV AL, DL	AL ← DL
	INC AL	AL ← AL + 1
	DAA	decimal adjust after addition
	INC CL	CL ← CL + 1
label	MOV [2002], AL	[2002] ← AL
	MOV [2003], CL	[2003] ← CL
	HLT	Halt the program.

Subtraction:

i. Move the contents of [2000] to AL
ii. Move the contents of [2001] to BL
iii. Subtract the contents of register BL from AL
iv. Decimally adjust AL after subtraction.
v. Move the contents of AL to 2002.
v. If carry is not set jump to step 10
vi. Move the value 99H to DL
vii Subtract value of AL from DL
viii Move DL to AL ~~and perform DAA~~
ix Increment AL and perform DAA. Increment CL
x Move the contents of DL to [2002]
xi Move the contents of CL to [2003]
xii. Halt the program

Sample I/O

Input: [2000] = 45
[2001] = 19

Output: [2002] - 26
[2003] - 00

Input: [2000] = 20
[2001] = 25

Output = [2002] - 05
[2003] - 01.

Result: BCD addition and subtraction has been performed using 8086 Microprocessor.

Aim: To implement a given set of operations through masm assembly programming.

Procedure:

1. Mount the directory containing the folder masm into C:\ as.

   mount C: D:\masm

2. To compile masm program:

   masm filename.asm

3. After compilation, debug errors if any.
4. To link the object file the syntax is:

   link filename.obj;;;

5. Then to debug:

   debug filename.exe

6. The various instruction used are:

   u - dissassemble
   d - to display values. (ex: d 076A:0000)
   e - to update values
   g - to terminate program
   q - quit.

```
ASSUME CS:code, DS:data
data segment
count equ 08h
data ends
code segment
start:MOV AX,data
      MOV DS,AX
      MOV CX,count
L1:   MOV AH,01h
      INT 21H
      CMP AL,60H
      JNC upper
      ADD AL,20H
      JMP skip
upper:SUB AL,20H
skip:MOV AH,02H
      MOV DL,AL
      INT 21H
      LOOP L1
      MOV AH,4CH
      INT 21H
code ends
end start
```

```
D:\>debug ass8.exe
-u
076A:0000 B86A07      MOV     AX,076A
076A:0003 8ED8        MOV     DS,AX
076A:0005 B90800      MOV     CX,0008
076A:0008 B401        MOV     AH,01
076A:000A CD21        INT     21
076A:000C 3C60        CMP     AL,60
076A:000E 7304        JNB     0014
076A:0010 0420        ADD     AL,20
076A:0012 EB02        JMP     0016
076A:0014 2C20        SUB     AL,20
076A:0016 B402        MOV     AH,02
076A:0018 8AD0        MOV     DL,AL
076A:001A CD21        INT     21
076A:001C E2EA        LOOP    0008
076A:001E B44C        MOV     AH,4C
```

```
-g
qQwWeEtTuUrRoOpP
Program terminated normally
```

Assignment 8

CASE CONVERSION

Aim: Get letters through keyboard and print (display) case converted letters on the fly.

Algorithm:

i. A character is read from the keyboard.
ii. If the value is greater than 60H, its value is decremented by 20H
    A - Z = 41 - 5A
    a - z = 61 - 7A
iii. Otherwise it is incremented by 20H
iv. The character stored in DL is now sent to the standard output device.
v. Repeats steps 2 through 4 until user wants to quit

Interrupts used: (DOS functions)

01 H - Reads the keyboard with echo & stores in AL

02H - writes character stored in DL to standard output device (monitor)

Execution steps:

i) mount c: d:\masm
ii) d:\
iii) masm case.asm
iv) link case.obj;;
v) debug case.exe
→ -g:
FfLLuUWw

Result: Conversion of case on the fly using a masm program is executed.

```
ASSUME CS:CODE, DS:DATA
DATA SEGMENT
      ORG 00H
      X DD 20.4375
      ORG 10H
      Y DD 20.4375
      ORG 20H
      SUM DD ?
DATA ENDS
CODE SEGMENT
start:      MOV AX,DATA
      MOV DS,AX
      FINIT
      FLD X
      FLD Y
      FADD ST(0),ST(1)
      FST SUM
      MOV AH,4CH
      INT 21H
CODE ENDS
END START
```

```
D:\>debug ass9.exe
-u
076D:0000 B86A07      MOV     AX,076A
076D:0003 8ED8        MOV     DS,AX
076D:0005 9B          WAIT    FINIT
076D:0006 DBE3
076D:0008 9B          WAIT    FLD     DWORD PTR [0000]
076D:0009 D9060000
076D:000D 9B          WAIT    FLD     DWORD PTR [0010]
076D:000E D9061000
076D:0012 9B          WAIT    FADD    ST,ST(1)
076D:0013 D8C1
076D:0015 9B          WAIT    FST     DWORD PTR [0020]
076D:0016 D9162000
076D:001A B44C        MOV     AH,4C
076D:001C CD21        INT     21
076D:001E F8          CLC
076D:001F B700        MOV     BH,00
```

```
-d 076a:0000
076A:0000  00 80 A3 41 00 00 00 00-00 00 00 00 00 00 00 00   ...A............
076A:0010  00 80 A3 41 00 00 00 00-00 00 00 00 00 00 00 00   ...A............
076A:0020  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
076A:0030  B8 6A 07 8E D8 9B DB E3-9B D9 06 00 00 9B D9 06   .j..............
076A:0040  10 00 9B D8 C1 9B D9 16-20 00 B4 4C CD 21 F8 B7   ........ ..L.!..
076A:0050  00 8A 87 48 2F D0 D8 73-17 E8 B6 00 8A 5E F8 B7   ...H/..s.....^..
076A:0060  00 8A 87 48 2F D0 D8 73-07 53 B0 01 50 E8 73 01   ...H/..s.S..P.s.
076A:0070  A0 B6 2C 3A 46 F8 74 7E-C7 46 FA 00 00 8A 46 F8   ..,:F.t~.F....F.
-g

Program terminated normally
-d 076a:0000
076A:0000  00 80 A3 41 00 00 00 00-00 00 00 00 00 00 00 00   ...A............
076A:0010  00 80 A3 41 00 00 00 00-00 00 00 00 00 00 00 00   ...A............
076A:0020  00 80 23 42 00 00 00 00-00 00 00 00 00 00 00 00   ..#B............
076A:0030  B8 6A 07 8E D8 9B DB E3-9B D9 06 00 00 9B D9 06   .j..............
076A:0040  10 00 9B D8 C1 9B D9 16-20 00 B4 4C CD 21 F8 B7   ........ ..L.!..
076A:0050  00 8A 87 48 2F D0 D8 73-17 E8 B6 00 8A 5E F8 B7   ...H/..s.....^..
076A:0060  00 8A 87 48 2F D0 D8 73-07 53 B0 01 50 E8 73 01   ...H/..s.S..P.s.
076A:0070  A0 B6 2C 3A 46 F8 74 7E-C7 46 FA 00 00 8A 46 F8   ..,:F.t~.F....F.
-
```

Assignment 9

FLOATING POINT OPERATIONS

Aim: To execute floating point operations - addition & subtraction using masm assembly programs

Algorithm:

I **Addition**

1. Define the data segment to be used in program
2. The code segment contains actual code to execute
2. 2 operands in [0000] and [0010] are added and stored in [0020]
4. Program is terminated using int 21H (ah = 4CH)

Sample I/O:

Input: [0000] - 00 [0010] - 00
[0001] - 80 [0011] - 80
[0002] - A3 [0012] - A3
[0003] - 41 [0013] - 41

Output: [0020] - 00
[0021] - 80
[0022] - 23
[0023] - 42

Execution steps:

1) mount d: c:\masm
2. d:\
3. masm floatadd.asm
4. link floatadd.obj,,,;
5 debug floatadd.exe
→ -d 076A:0000

0000 : 00 80 A3 41 (20.4375)
0010 : 00 80 A3 41 (20.4375)

→ -g

Program terminated normally

→ -d 076A:0000

0000 : 00 80 A3 41
0010 : 00 80 A3 41
0020 : 00 80 23 42

→ -q

```
ASSUME CS:CODE, DS:DATA
DATA SEGMENT
ORG 00H
X DD 20.4375
ORG 10H
Y DD 0.125
ORG 20H
SUM DD ?
DATA ENDS
CODE SEGMENT
start: MOV AX,DATA
MOV DS,AX
FINIT
FLD Y
FLD X
FSUB ST(0),ST(1)
FST SUM
MOV AH,4CH
INT 21H
CODE ENDS
END START
```

```
D:\>debug assb9.exe
-u
076D:0000 B86A07        MOV     AX,076A
076D:0003 8ED8          MOV     DS,AX
076D:0005 9B            WAIT
076D:0006 DBE3          FINIT
076D:0008 9B            WAIT
076D:0009 D9061000      FLD     DWORD PTR [0010]
076D:000D 9B            WAIT
076D:000E D9060000      FLD     DWORD PTR [0000]
076D:0012 9B            WAIT
076D:0013 D8E1          FSUB    ST,ST(1)
076D:0015 9B            WAIT
076D:0016 D9162000      FST     DWORD PTR [0020]
076D:001A B44C          MOV     AH,4C
076D:001C CD21          INT     21
076D:001E F8            CLC
076D:001F B700          MOV     BH,00
-
```

```
-d 076a:0000
076A:0000  00 80 A3 41 00 00 00 00-00 00 00 00 00 00 00 00   ...A............
076A:0010  00 00 00 3E 00 00 00 00-00 00 00 00 00 00 00 00   ...>............
076A:0020  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
076A:0030  B8 6A 07 8E D8 9B DB E3-9B D9 06 10 00 9B D9 06   .j..............
076A:0040  00 00 9B D8 E1 9B D9 16-20 00 B4 4C CD 21 F8 B7   ........ ..L.!..
076A:0050  00 8A 87 48 2F D0 D8 73-17 E8 B6 00 8A 5E F8 B7   ...H/..s.....^..
076A:0060  00 8A 87 48 2F D0 D8 73-07 53 B0 01 50 E8 73 01   ...H/..s.S..P.s.
076A:0070  A0 B6 2C 3A 46 F8 74 7E-C7 46 FA 00 00 8A 46 F8   ..,:F.t~.F....F.
-g

Program terminated normally
-d 076a:0000
076A:0000  00 80 A3 41 00 00 00 00-00 00 00 00 00 00 00 00   ...A............
076A:0010  00 00 00 3E 00 00 00 00-00 00 00 00 00 00 00 00   ...>............
076A:0020  00 80 A2 41 00 00 00 00-00 00 00 00 00 00 00 00   ...A............
076A:0030  B8 6A 07 8E D8 9B DB E3-9B D9 06 10 00 9B D9 06   .j..............
076A:0040  00 00 9B D8 E1 9B D9 16-20 00 B4 4C CD 21 F8 B7   ........ ..L.!..
076A:0050  00 8A 87 48 2F D0 D8 73-17 E8 B6 00 8A 5E F8 B7   ...H/..s.....^..
076A:0060  00 8A 87 48 2F D0 D8 73-07 53 B0 01 50 E8 73 01   ...H/..s.S..P.s.
076A:0070  A0 B6 2C 3A 46 F8 74 7E-C7 46 FA 00 00 8A 46 F8   ..,:F.t~.F....F.
-
```

II Subtraction:

1. Define the data segment to the used in the program
2. The code segment contains the actual code.
3. The 2 operands in [0000] and [0010] are subtracted and stored in [0020]
4. Program is terminated using int 21H (ah = 4CH)

Sample I/O:

Input: [0000] - 00
[0001] - 80
[0002] - A3
[0004] - 41
[0010] - 00
[0011] - 00
[0012] - 00
[0013 - 3E

Output: [0020] - 00
[0021] - 80
[0022] - A2
[0023] - 41

Execution steps:

1. mount d: c:\masm
2. d:\
3. masm floatsub.asm
4. link floatsub.obj,,;
5. debug floatsub.exe

→ -d 076A:0000

0000 : 00 80 A3 41 (20.4375)
0010 : 00 00 00 3E (0.125)

→ -g

Program terminated normally

→ -d 076A:0000

0000: 00 80 A3 41
0010: 00 00 00 3E
0020: 00 80 A2 41

→ -q

Result: floating point operations - additions and subtraction have been executed in masm assembly program.

```
ASSUME CS:CODE,DS:DATA
DATA SEGMENT
MESSAGE DB "THIS IS THE STRING$"
DATA ENDS
CODE SEGMENT
start:      MOV AX,DATA
      MOV DS,AX
      MOV AH,09h
      MOV DX,OFFSET MESSAGE
      INT 21H
      MOV AH,4CH
      INT 21H
CODE ENDS
END start
```

```
D:\>debug ass10.exe
-u
076C:0000 B86A07        MOV     AX,076A
076C:0003 8ED8          MOV     DS,AX
076C:0005 B409          MOV     AH,09
076C:0007 BA0000        MOV     DX,0000
076C:000A CD21          INT     21
076C:000C B44C          MOV     AH,4C
076C:000E CD21          INT     21
076C:0010 B86A07        MOV     AX,076A
076C:0013 8ED8          MOV     DS,AX
076C:0015 9B            WAIT
076C:0016 DBE3                  FINIT
076C:0018 9B            WAIT
076C:0019 D9061000              FLD     DWORD PTR [0010]
076C:001D 9B            WAIT
076C:001E D9060000              FLD     DWORD PTR [0000]
-
```

```
-g
THIS IS THE STRING
Program terminated normally
-
```

Assignment 10

DISPLAY A STRING

Aim: Display a string to the prompt using interrupts in masm assembly programming for 8086. (DOS function)

Algorithm:

1. Define data segment with necessary variables.
2. The message to be displayed is enclosed with quotes and appended with '$'.
3. The DOS function 09 is used to display the string (in DX)
4. Terminate program using int 21H (ah=4CH)

DOS function used:

09H - display string in DX with offset.

Execution steps:

1. mount d: c:\masm
2. d:\
3. masm dispstring.asm
4. link dispstring.obj;;;
5. debug dispstring.exe

→ -g

THIS IS THE STRING

program terminated normally.

Result: A string is printed in the prompt using masm program dos function

```
assume cs:code,ds:data
data segment
day db 01 dup(?)
month db 01 dup(?)
year db 02 dup(?)
data ends
code segment
 org 0100h
start: mov ax,data
 mov ds,ax
 mov ah,2ah
 int 21h
mov si,offset day
 mov [si],dl
mov si,offset month
 mov [si],dh
mov si,offset year
 mov [si],cx
mov ah,4ch
 int 21h
code ends
end start
```

```
D:\>DEBUG ASS11.EXE
-U
076B:0100 B86A07      MOV     AX,076A
076B:0103 8ED8        MOV     DS,AX
076B:0105 B42A        MOV     AH,2A
076B:0107 CD21        INT     21
076B:0109 BE0000      MOV     SI,0000
076B:010C 8814        MOV     [SI],DL
076B:010E BE0100      MOV     SI,0001
076B:0111 8834        MOV     [SI],DH
076B:0113 BE0200      MOV     SI,0002
076B:0116 890C        MOV     [SI],CX
076B:0118 B44C        MOV     AH,4C
076B:011A CD21        INT     21
076B:011C FF7701      PUSH    [BX+01]
076B:011F 40          INC     AX
-
```

```
-D 076A:0000
076A:0000  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
076A:0010  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
076A:0020  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
076A:0030  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
076A:0040  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
076A:0050  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
076A:0060  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
076A:0070  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
-G

Program terminated normally
-D 076A:0000
076A:0000  0B 0A E6 07 00 00 00 00-00 00 00 00 00 00 00 00   ................
076A:0010  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
076A:0020  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
076A:0030  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
076A:0040  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
076A:0050  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
076A:0060  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
076A:0070  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
```

Assignment 11

SYSTEM DATE AND TIME

Aim: To get the system date and system time and store it in the memory

Algorithm:

I System date:

1. Define data segment with necessary variables
2. The DOS function 2A is used to input the system date
3. The day is stored in DL, month in DH and year in CH (CX)
4. Move the contents in DL, DH and CH to [0000], [0001] and [0002]
5. Terminate the program using int 21H (ah = 4CH)

DOS function used:

2A : gets system and stores month in DH, day in DL and year in CH(CX)

Execution steps:

1. mount d: c:\masm
2. d:\
3. masm sysdate.asm
4. link sysdate.obj,,;
5. debug sysdate.exe.

```
-d 076A:0000
     0000:  00  00  00  00
-g
     Program terminated successfully.
-d   076A:0000
     0000:  0B  0A  E6  07   (11/10/2022)
```

```
assume cs:code,ds:data
data segment
hour db 01 dup(?)
minute db 01 dup(?)
second db 02 dup(?)
data ends
code segment
 org 0100h
start: mov ax,data
mov ds,ax
mov ah,2ch
 int 21h
mov si,offset hour
 mov [si],ch
mov si,offset minute
 mov [si],cl
mov si,offset second
 mov [si],dh
mov ah,4ch
 int 21h
code ends
end start
```

```
D:\>debug assb11.exe
-u
076B:0100 B86A07      MOV     AX,076A
076B:0103 8ED8        MOV     DS,AX
076B:0105 B42C        MOV     AH,2C
076B:0107 CD21        INT     21
076B:0109 BE0000      MOV     SI,0000
076B:010C 882C        MOV     [SI],CH
076B:010E BE0100      MOV     SI,0001
076B:0111 884C        MOV     [SI],CL
076B:0113 BE0200      MOV     SI,0002
076B:0116 8834        MOV     [SI],DH
076B:0118 B44C        MOV     AH,4C
076B:011A CD21        INT     21
076B:011C FF7701      PUSH    [BX+01]
076B:011F 40          INC     AX
-
```

```
-d 0761:0000
0761:0000  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
0761:0010  00 0D 61 73 73 62 31 31-2E 65 78 65 0D 00 00 00   ..assb11.exe....
0761:0020  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
0761:0030  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
0761:0040  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
0761:0050  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
0761:0060  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
0761:0070  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
-g

Program terminated normally
-d 076a:0000
076A:0000  0E 1A 1D 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
076A:0010  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
076A:0020  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
076A:0030  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
076A:0040  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
076A:0050  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
076A:0060  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
076A:0070  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
-
```

II System time:

1. Define the data segment with necessary variable.
2. The DOS function 2C is used to input system time.
3. Hour is stored in CH, minute in CL and seconds in DH(DX)
4. Move the contents of CH, CL, DH to [0000], [0001] and [0002]
5. Terminate the program using int 21H(ah = 4CH)

DOS function used

2C - Used to get time from system and store hour in CH, minutes in CL, and seconds in DH

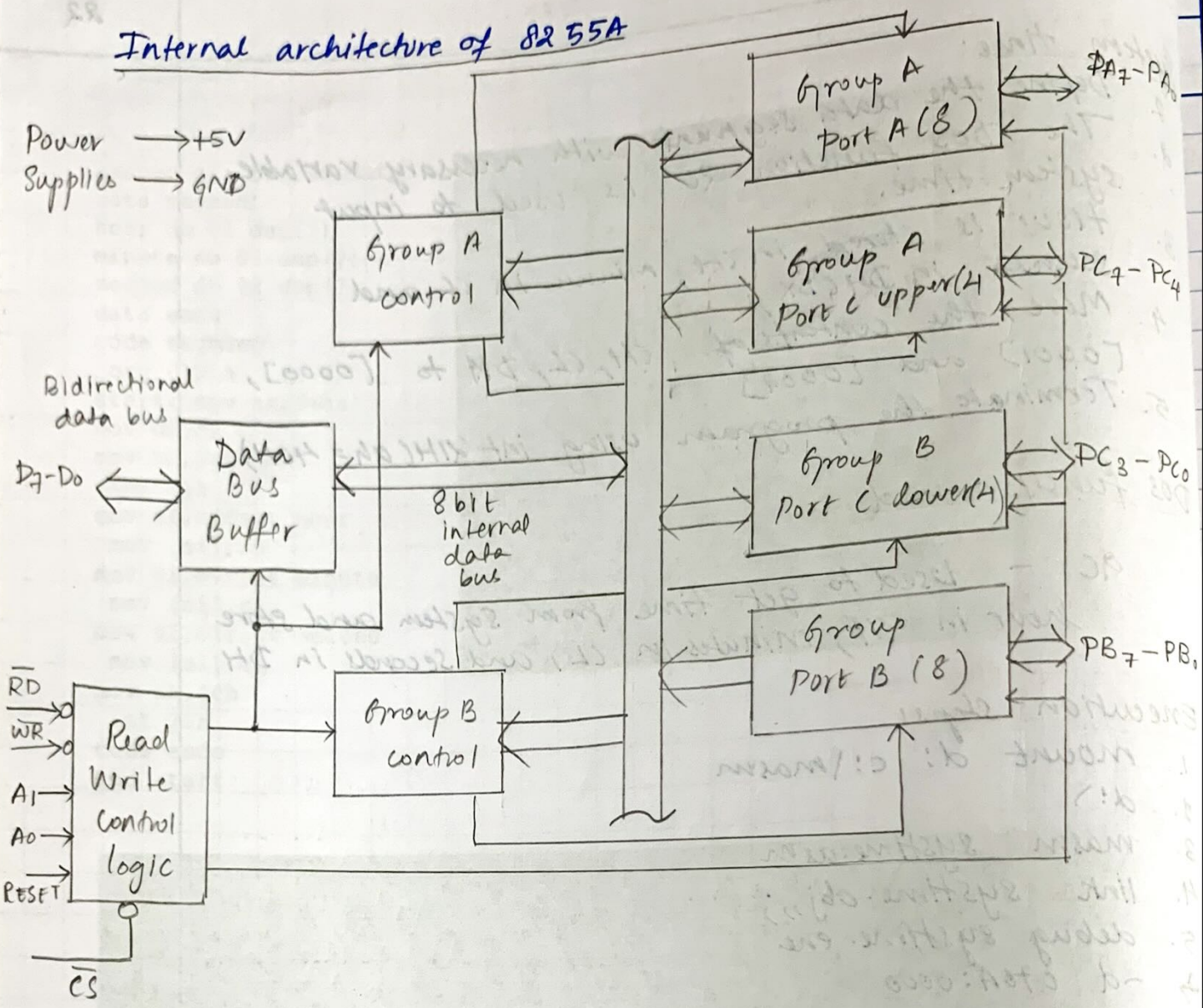
Execution steps:

1. mount d: c:\masm
2. d:\
3. masm systime.asm
4. link systime.obj,,;
5. debug systime.exe

```
-d 076A:0000
0000 : 00 00 00 00
-g
Program terminated successfully
-d 076A:0000
0000 : 0E 1A 1D 00   (14 : 26 : 13)
-q
```

Result: The system time and date have been successfully stored in the memory.

Internal architecture of 8255A



Port addresses:

C6 – control register

C0 – PortA

C2 – Port B

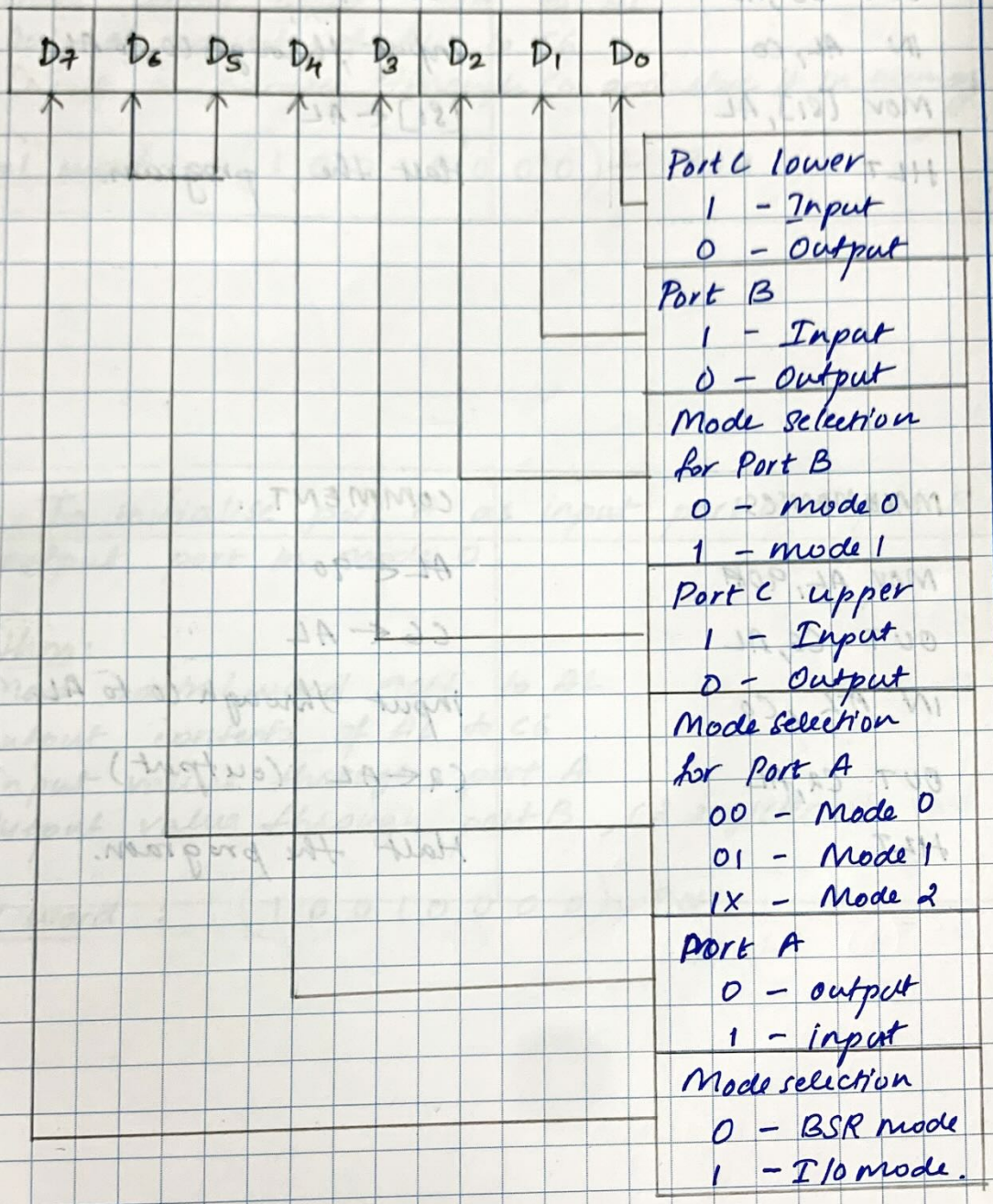
CA – Port C

Assignment 12

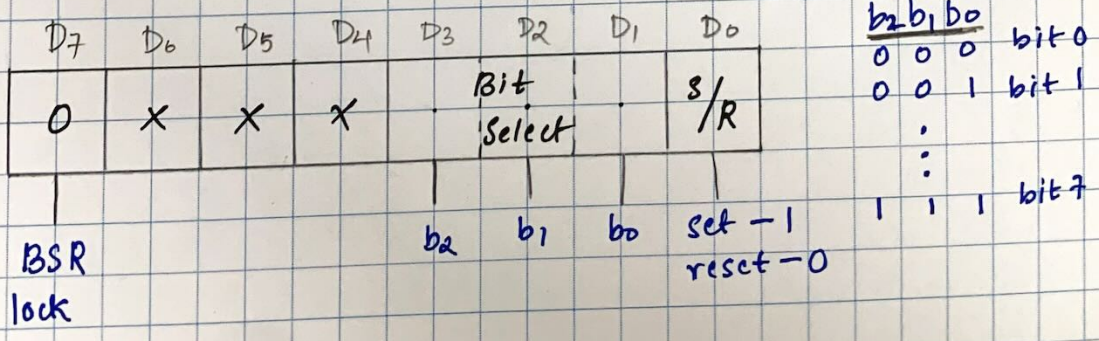
8255 PARALLEL INTERFACE

Aim: To interface the microprocessor with parallel interface of 8255 chip

Control word of 8255.



BSR mode control word.



I.

MNEMONIC	COMMENT
MOV SI, 1500	SI ← 1500
MOV AL, 90	AL ← 90
OUT C6, AL	C6 ← AL
IN AL, C0	input through C0 to AL
MOV [SI], AL	[SI] ← AL
HLT	Halt the program.

II

MNEMONICS	COMMENT
MOV AL, 90H	AL ← 90
OUT C6, AL	C6 ← AL
IN AL, C0	input through C0 to AL
OUT C2, AL	C2 ← AL (output)
HLT	Halt the program.

I. Aim: To initialise Port A as input port in mode 0

Algorithm.
1. Move 1500 to SI
2. Move control word 90H to AL
3. Output contents of AL to CG
4. Input a character through C0 and store it in memory

Control word : (1 0 0 1 0 0 0 0) — 90H

II Aim: To initialise port A as input port and port B as output port in mode 0

Algorithm:
1. Move control word 90H to AL
2. Output contents of AL to CG
3. Input values through port A
4. Output value through port B, C2 register

Control word : (1 0 0 1 0 0 0 0) - 90H

III

Mnemonics	Comments.
MOV AL, 90	AL ← 90
OUT C6, AL	C6 ← AL
MOV AL, 80	AL ← 80
OUT C4, AL	C4 ← AL (80)
HLT	Halt the program.

IV

Mnemonics	Comments.
MOV AL, 80	AL ← 80
OUT C6, AL	C6 ← AL
MOV AL, 01	AL ← 01
OUT C4, AL	C4 ← AL (01)
MOV AL, 07	AL ← 07
OUT C6, AL	C6 ← AL
HLT	Halt the program.

III Aim: To initialise Port C as output port in mode-0

Algorithm:
1. Move control word 90H to control register CG.
2. Move some value say 80 to port C
3. Halt the program.

Control word : (1 0 0 1 0 0 0 0)-90

IV Aim: To initialise port C as an output port in mode 0 and to explain the bit-set and reset (BSR) feature of Port C.

Algorithm:
1. Move control word 80 into AL
2. Move the control word in AL to control register CG.
3. place a data into AL
4. output the contents of AL into CH register.
5. Calculate control word of BSR mode and store it in control register
6. Halt the program.

Control word for mode 0 : (1 0 0 0 0 0 0 0)- 80
Control word for BSR : (0 0 0 0 0 1 1 1)- 07

bit 3 set

V

Mnemonics	Comments
MOV SI, 1200	SI ← 1200
MOV AL, 99	AL ← 99
OUT C6, AL	C6 ← AL
IN AL, C4	AL ← C4
MOV [SI], AL	[SI] ← AL
HLT	Halt the program.

VI

	Mnemonics	Comments.
	MOV AL, BH	AL ← BH
	OUT C6, AL	C6 ← AL
read:	IN AL, C4	AL ← C4
	AND AL, 20	AL and 20
	JZ read	
	IN AL, C0	AL ← C0
	OUT C2, AL	C2 ← AL
	HLT	Halt the program.

V Aim: To initialise Port C as an input port in Mode 0

Algorithm:
1. Make SI point to 1200
2. Move control word 99H into AL
3. Store control word in AL to control register CG
4. Input a charecter from CH port
5. Move the value to 1200.

Control word: (1 0 0 1 1 0 0 1) - 99.

VI Aim: To verify the working of 8255 in mode 1

Algorithm:
1. Move control word BH to AL
2. Output the word in AL to control register CG
3. Input data through port C
4. If D5 is not 0, repeat Step 3
5. If D5 is zero then get input from port A - C0
6. Output the data to port B - C2

Control Word: (1 0 1 1 0 1 0 0) - BH
(Port A input, Port B output)

D7	D6	D5	D4	D3	D2	D1	D0
I/O	I/O	IBF_A	$INTE_A$	$INTR_A$	$INTE_B$	IBF_B	$INTR_B$

→ is set when strobe is given.

Result: 8255 chip was interfaced with 8086 microprocessor.

Port A

Bits	A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0
LEDs	D_8	D_7	D_6	D_5	D_4	D_3	D_2	D_1

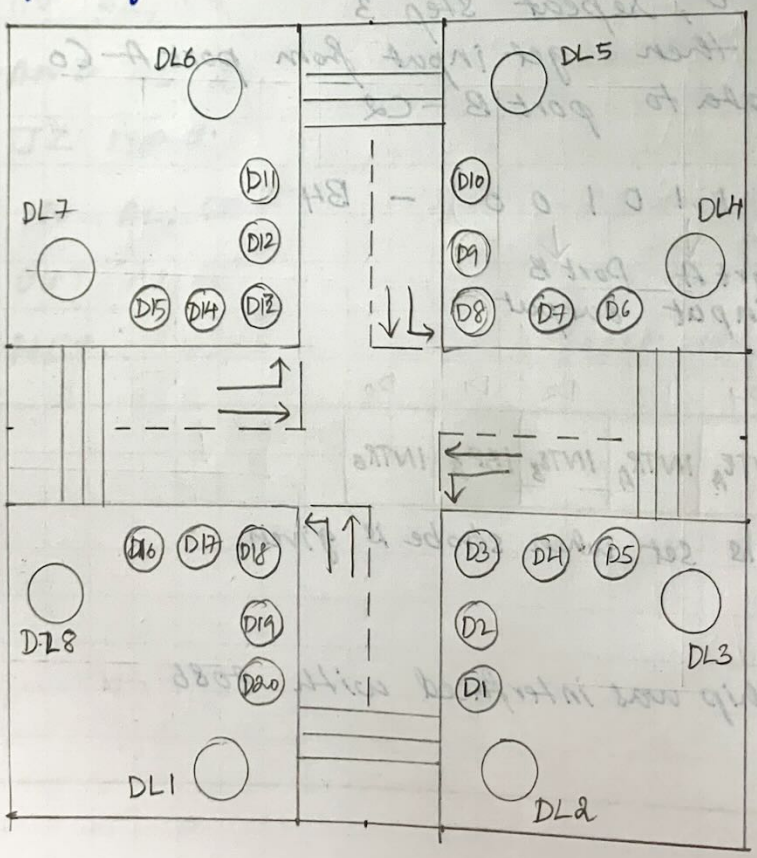
Port B

Bits	B_7	B_6	B_5	B_4	B_3	B_2	B_1	B_0
LEDs	D_{20}	D_{19}	D_{18}	D_{17}	DL7 DL8	DL5 DL6	DL3 DL4	DL1 DL2

Port C

Bits	C_7	C_6	C_5	C_4	C_3	C_2	C_1	C_0
LEDs	D_{16}	D_{15}	D_{14}	D_{13}	D_{12}	D_{11}	D_{10}	D_9

traffic light board:



Assignment - 13

TRAFFIC LIGHT CONTROLLER

Aim: To implement traffic light controller by interfacing with 8086 microprocessor

Control words Lookup:

Repeat the following

12 > Port C 27 > Port B 44 > Port A Call Delay.	Time slot I
48 > Port A Call Delay1	Time slot II
10 > PortC 2B > PortB 92 > Port A Call delay.	Time slot III
48 > Port A Call delay1	Time slot IV
10 > Port C 9D > Port B 84 > Port A Call Delay	Time slot V
20 > Port C Call delay1	Time slot VI
48 > Port C 2E > Port B 84 > Port A Call Delay	Time slot VII
49 > Port C 04 > Port A Call Delay1	Time slot VIII

Label	Mnemonics	Comments.
	ORG 1000	
	CNTRL EQU 26H	control register address
	PORTA EQU 20H	port A address
	PORTB EQU 22H	port B address
	PORTC EQU 24H	port C address
Start	MOV AL, 80H	AL ← 80
	OUT CNTRL, AL	CNTRL ← AL
		BX ← LOOKUP
Repeat	MOV BX, LOOKUP	SI ← Label
	MOV SI, Label	
	CALL out	
		AL ← [SI]
	MOV AL, [SI]	PORTA ← AL
	OUT PORTA, AL	
	CALL DELAY	SI ← SI + 1
	INC SI	BX ← BX + 1
	INC BX	
	CALL OUT	
		AL ← [SI]
	MOV AL, [SI]	PORTB ← AL
	OUT PORTB, AL	SI ← SI + 1
	INC SI	AL ← [SI]
	MOV AL, [SI]	PORTA ← A
	OUT PORTA, AL	
	CALL DELAY1	
	JMP REPEAT	
OUT	MOV AL, [BX]	AL ← [BX]
	OUT PORTC, AL	PORTC ← AL
	INC BX	BX ← BX + 1
	MOV AL, [BX]	AL ← [BX]
	OUT PORTA, AL	PORTA ← AL
	CALL DELAY	
	RET	

Algorithm:

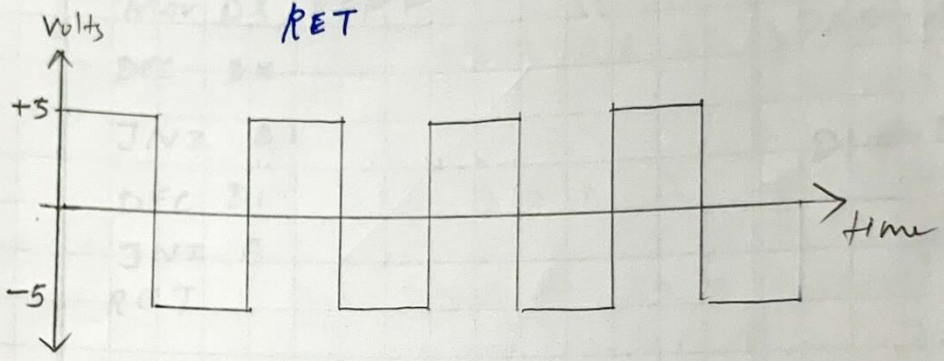
1. Start
2. Use 'EQU' directive to assign values to CNTRL, PORTS A, B, and C
3. Move control word 80 into AL and push it to the control register.
4. Load lookup table with values, addressed using BX
5. Move the values into SI (of label)
6. Jump to step 15
7. Move the contents of SI into AL
8. Output contents of AL into Port A
9. Call delay / routine
10. Increment SI and BX
11. Repeat above 3 steps for Port B and C
12. Jump to step 4
13. Output the contents of AL into Port C and increment BX
14. Move the contents of BX into AL
15. Output contents of AL into port B register and increment BX
16. Return to statement after calling routine
17. The delay and delay1 are defined
18. Move 40H to D1
19. Move FFFF to DX
20. Decrement DX / D1 until DX = 0 / D1 = 0
21. Return to statement after calling routine
22. Load lookup and label tables with required values

Label	Mnemonics	Comments.
DELAY	Mov DI, 0040H	DI ← 0040
A	Mov DX, FFFF	DX ← FFFF
A1	DEC DX	DX ← DX - 1
	JNZ A1	
	DEC DI	DI ← DI - 1
	JNZ A	
	RET	
DELAY1	Mov DI, 0015H	DI ← 0015
B	Mov DX, FFFF	DX ← FFFF
B1	DEC DX	DX ← DX - 1
	JNZ B1	
	DEC DI	DI ← DI - 1
	JNZ B	
	RET	
LOOKUP	DB 12H, 87H, 44H, 10H, 2BH, 92H,	
	10H, 9DH, 84H, 48H, 2EH, 84H	
LABEL	DB 48H, 4BH, 20H, 49H, 04H	
	END	

Result: Traffic light controller was unimplemented by interfacing with 8086 microprocessor

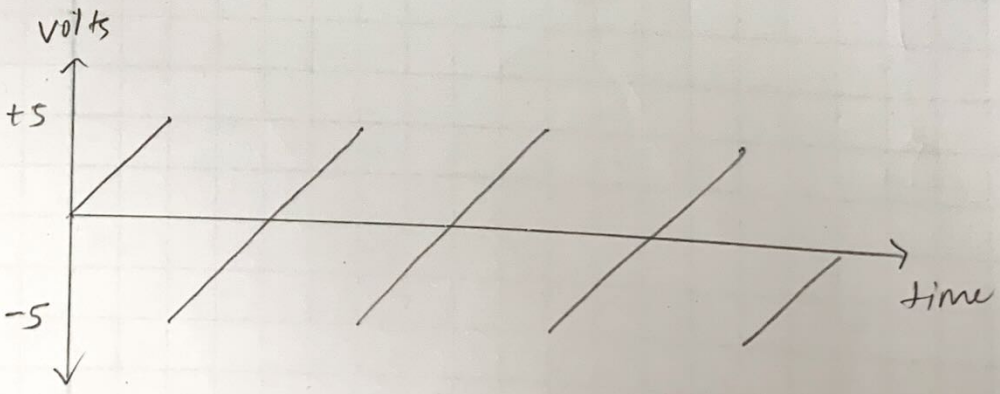
Square Wave Generation

```
START :   MOV  AL, 00
          OUT  C8, AL
          CALL 1010
          MOV  AL, FF
          CALL 1010
          JMP  START
1010 :    MOV  CX, 05FF
1013 :    LOOP 1013
          RET
```



Sawtooth Wave Generation

```
START :   MOV  AL, 00
L     :   OUT  C8, AL
          INC  AL
          JNZ  L
          JMP  START
```



Assignment - 14

WAVEFORM GENERATION USING DAC INTERFACE 30

Aim: To generate waveform using DAC interface

DAC:

Input data is in hexadecimal and corresponding output voltage are tested.

Input	Output Voltage
00	-5
01	-4.96
:	:
FF	+5

I Aim: Generate a square wave at output of DAC

Algorithm:

1. Move value of 00 to AL
2. Output contents of AL into C8 register
3. Call delay routine
4. Move FF to AL
5. Output AL into C8 register
6. Call delay routine.
7. Repeat

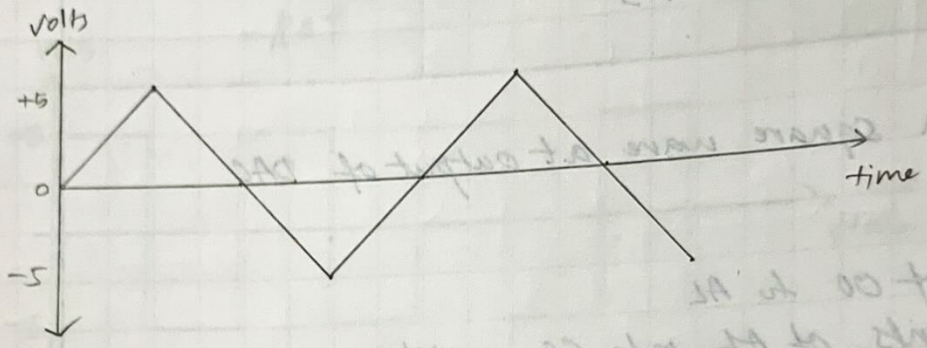
II Aim: Generate a saw tooth wave at output of DAC

Algorithm:

1. Move 00 to AL
2. Output contents of AL into C8
3. Increment AL
4. Repeat incrementing until AL == FF
5. Repeat

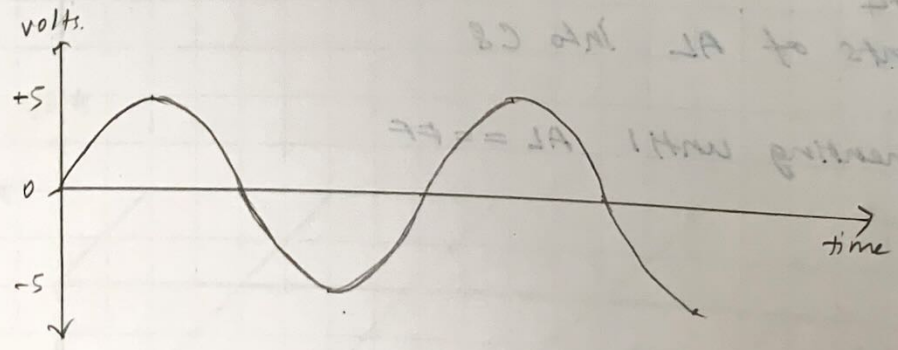
Triangular wave generator.

```
START:  MOV AL,00
L:      OUT C8,AL
        INC AL
        JNZ L
        MOV AL, FF
L:      OUT C8, AL
        DEC AL
        JNZ L1
        JMP START
```



Sine wave generator

```
1000  :  MOV CX, 42
         MOV SI, 1500
LABEL :  MOV AL, [SI]
         OUT CS, AL
         INC SI
         LOOP LABEL
         JMP 1000
```



LOOKUP TABLE:

```
1500: 7F 8A 95 A0
      AA B5 BF C8
      D1 D9 E0 E7
      ED F2 F7 FA
      FC FA F7 F2
      ED E7 E0 D9
      D1 C8 BF B5
      AA A0 95 8A
      7F 74 69 5F
      53 49 3F 36
      2D 25 1B 17
      10 0B 07 04
      01 00 01 04
      07 0B 10 17
      1D 25 2D 36
      3F 49 53 5F
      69 74
```

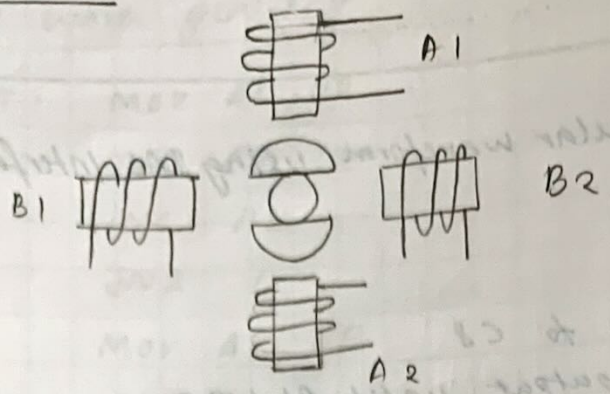
III **Aim:** To generate triangular waveform using DAC interface.

Algorithm:

1. Move 00 to AL
2. Output contents of AL to C8
3. Increment AL and output until AL != 0
4. Move FF to AL
5. Output contents of AL to C8
6. Decrement AL until AL != 0
7. Repeat.

Results: waveforms grenerated from DAC using code is generated.

Stepper Motor:



Clockwise.

Step	A1	A2	B1	B2	
1	1	0	0	0	8
2	0	0	0	1	1
3	0	1	0	0	4
4	0	0	1	0	2

Anticlockwise

Step	A1	A2	B1	B2	
1	1	0	0	0	8
2	0	0	1	0	2
3	0	1	0	0	4
4	0	0	0	1	1

Label	Mnemonics	comments.
START	MOV DI, 1018	
	MOV CL, 04	CL ← 04
next	MOV AL, [DI]	AL ← [DI]
	OUT C0, AL	C0 ← AL
	MOV DX, 1010	
delay	DEC DX	
	JNZ delay	delay routine
	INC DI	
	LOOP next	
	JMP START	

1018 — 08 01 04 02 — clockwise

1018 — 08 02 04 01 — anticlockwise.

Assignment 15.

STEPPER MOTOR INTERFACE

Aim: To interface the stepper motor with 8086 and run it in clockwise and anticlockwise direction

Data register address = C0

lookup table:

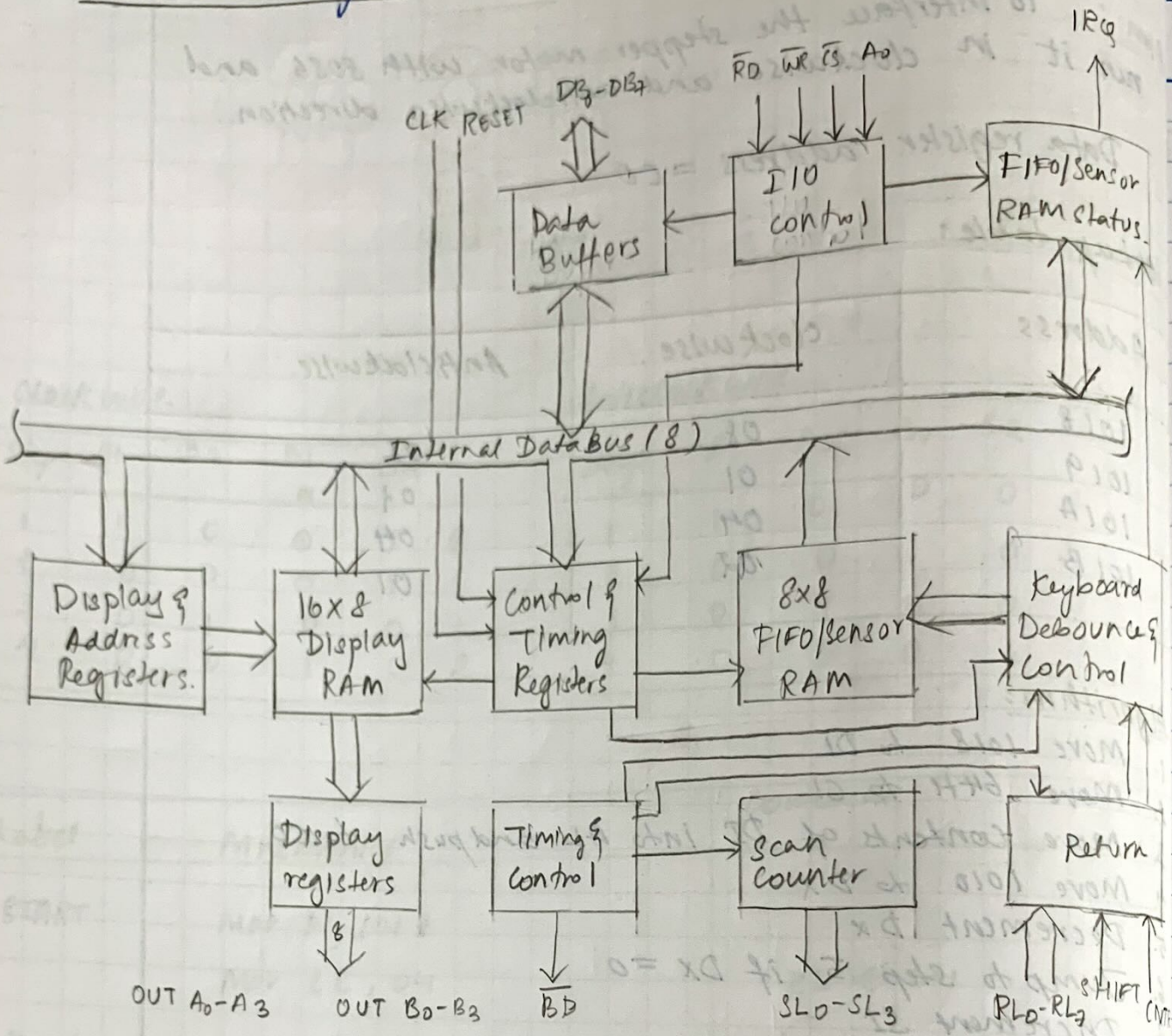
Address	clockwise	Anticlockwise.
1018	08	08
1019	01	02
101A	04	04
101B	02	01

Algorithm:

1. Move 1018 to DI
2. Move 64H to CL
3. Move contents of DI into AL and push to C0
4. Move 1010 to DX
5. Decrement DX
6. Jump to step 5 if DX = 0
7. Increment SI
8. Repeat Step 4 if CL ≠ 0
9. Load values into lookup table at 1018 for clockwise and anticlockwise
10. Repeat.

Result: Stepper motor was interfaced with the microprocessor.

8279 Block diagram:



Keyboard Data Format:

CNTL	SHIFT	ROW	COLUMN

control register - C2

data register - C0

Keyboard Condition in Board.

	C0	C1	C2	C3	C4	C5	C6	C7
R0	0	1	2	3	4	5	6	7
R1	8	9	10	11	12	13	14	15

Assignment – 16

KEYBOARD AND DISPLAY INTERFACE

Aim: To interface keyboard and display chip (8279) to 8086 microprocessor

Command word format:

0	0	0	D	D	K	K	K

K	K	K	Keyboard Moves
0	0	0	Encoded scan keyboard - 2 key lockout
0	0	1	Decoded scan keyboard - 2key lockout
0	1	0	Encoded scan keyboard - N key rollover
0	1	1	Decoded scan keyboard - N key rollover
1	0	0	Encoded scan sensor matrix
1	0	1	Decoded scan sensor matrix
1	1	0	Strobed input encoded display scan
1	1	1	Strobed input decoded display scan.

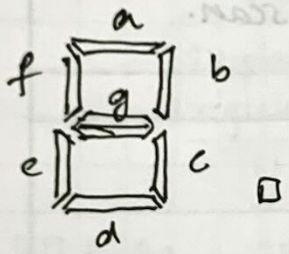
D	D	Display Modes
0	0	8-8 bit character display left entry
0	1	16-8 bit character display ~~right~~ left entry
1	0	8-8 bit character display right entry
1	1	16-8 bit character display right entry.

Read FIFO/Sensor RAM command.

0	1	0	AI	X	A	A	A

I	Mnemonics	Comments.
	MOV BX, 1100	BX ← 1100
	IN AL, C2	AL ← C2
LOOP:	TEST AL, 07	test for D2 - D0 bits
	JZ LOOP	
	MOV AL, 40	AL ← 40
	OUT C2, AL	C2 ← AL
	IN AL, C0	AL ← C0
	MOV [BX], AL	[BX] ← AL
	HLT	Halt the program

Segment Definition



If key 1 is pressed.

data word:

1 1 0 0 0 0 0 1

Display format: (Data)

D7	D6	D5	D4	D3	D2	D1	D0
d	c	b	a	dp	e	f	g

I Aim: To read a key from the keyboard and store it in one memory location

Algorithm:

1. Move 1100 to BX
2. Read the control word from the interface board and store it in AL
3. If the D_2 to D_0 bits are 000, jump to above step.
4. Move the control word (40) to AL and write it to C2
5. Input keystroke from C0 and store it in AL
6. Move the contents of AL to memory pointed by BX

II

Label	Mnemonics	comments.
	ORG 1000H	
START:	MOV SI, 1200	SI ← 1200
	MOV CX, 000F	CX ← 000F
	MOV AL, 10	AL ← 10
	OUT C2, AL	C2 ← AL
	MOV AL, CC	AL ← CC
	OUT C2, AL	C2 ← AC
	MOV AL, 90	AL ← 90
	OUT C2, AL	C2 ← AL
NEXT	MOV AL, [SI]	AL ← [SI]
	OUT C0, AL	~~OUT~~ C0 ← AL
	CALL DELAY	
	INC SI	SI ← SI + 1
	LOOP NEXT	
	JMP START	
DELAY	MOV D1, 0003	
B	MOV DX, FFFF	
B1	DEC DX	
	JNZ B1	
	DEC D1	
	JNZ B	
	RET	

LOOKUP

1200				
	FF	FF	FF	FF
	FF	FF	FF	FF
	98	68	7C	C8
	FF	1C	29	FF

II Aim: To roll message in display "HELP US"

Algorithm:

1. Move 1200 to SI
2. Move 000F to CX
3. Move control word (10 to AL)
4. Output contents of AL to C2
5. Output control word CC and 90 to C2
6. Move contents of AL into C0
7. Call delay.
8. Increment SI and repeat steps from 6 till CX ≠ 0
9. Decrement FFFF to 0000 thrice inside the delay routine
10. Load lookup table with required values.

Result: The 8279 was interfaced with 8086 microprocessor.

Addition

I Label	Mnemonics	Label Comments
	MOV R0, #00	carry
	MOV A, #06	input 1
	ADD A, #01	input 2
	JNC LABEL	
	INC R0	R0 ← R0+1
LABEL	MOV DPTR, #4150	DPTR ← 4150
	MOVX @DPTR, A	[DPTR] ← A
	MOV A, R0	A ← R0
	INC DPTR	
	MOVX @DPTR, A	[DPTR] ← A
HERE	SJMP HERE	

II. Subtraction

LABEL	Mnemonics	comments.
	MOV R0, #00	R0 ← 00
	CLR C	clear carry bit
	MOV A, #FF	A ← FF
	SUBB A, #01	A ← A − 01
	JNC label	
	INC R0	R0 ← R0+1
	CPL A	complement A
	INC A	A ← A+1
label	MOV DPTR, #4150	DPTR ← 4150
	MOVX @DPTR, A	[DPTR] ← A
	MOV A, R0	A ← R0
	INC DPTR	DPTR ← DPTR+1
	MOVX @DPTR, A	[DPTR] ← A
here	SJMP here.	

Assignment 17

8-BIT ARITHMETIC OPERATIONS USING 8051

Aim: To perform 8-bit operations using 8051 microcontroller.

Algorithm:

I **Addition:**

1. Maintain carry in Register R0
2. Move the values to be added to A and then add with a the other number.
3. If carry is not set then halt.
4. Else update carry in R0.
5. Move the contents of R0 back to memory.

Sample I/O:

Output: [4150] - 07 (06;01)
[4151] - 00

II **Subtraction:**

1. Maintain carry in R0
2. Clear the carry
3. Move FF into A
4. Subtract 01 from A
5. If carry is not set jump to 8
6. Increment R0
7. Obtain its complement and add 1 to it
8. Move 4150 to DPTR
9. Move the value in A to 4150
10. Increment DPTR and move carry to address

Sample II:

Output: [4150] - FE (FF, 01)
[4151] - 00

III Multiplication

Label	Mnemonics	comments.
	MOV A, #03	A ← 03
	MOV B, #05	B ← 05
	MUL AB	BA ← A × B
	MOV DPTR, #4150	DPTR ← 4150
	MOVX @DPTR, A	[4150] ← A
	INC DPTR	DPTR ← DPTR + 1
	MOV A, B	A ← B
	MOVX @DPTR, A	[4151] ← A
here	SJMP here.	

IV Division:

Label	Mnemonics	Comments.
	MOV A, #0F	A ← 0F
	MOV B, #03	B ← 03
	DIV AB	BA ← A ÷ B
	MOV DPTR, #4150	DPTR ← 4150
	MOVX @DPTR, A	[DPTR] ← A
	INC DPTR	DPTR ← DPTR + 1
	MOV A, B	A ← B
	MOVX @DPTR, A	[DPTR] ← A
here	SJMP here.	

III Multiplication:

1. Mov 03H to A and 05H to B
2. Multiply A x B and store in BA
3. Move 4150 to DPTR
4. Move the contents in A to memory pointed by DPTR. and increment DPTR
5. Move the contents of B to DPTR

Sample I/O:
output: [4150] - 0F
[4151] - 00

IV Division:

1. Move 0FH to A and 03H to B
2. Divide A by B
3. Quotient is in A and remainder at B.
4. point DPTR to 4150
5. Move the contents of A to DPTR and increment DPTR
6. Move contents of B to DPTR

Sample I/O:
output: [4150] - 05
[4151] - 00

Result: Various 8 bit arithmetic operations were performed on 8051 microcontroller

Label	Mnemonics	Comments.
	MOV A, #03	~~BA ← A x B~~ A ← 03
	MOV B, #03	B ← 03
	MOV R0, #03	R0 ← 03
	MUL AB	BA ← A x B
	MOV B, R0	B ← R0
	MUL AB	BA ← A x B
	MOV DPTR, #4150	DPTR ← 4150
	MOVX @DPTR, A	[DPTR] ← A
	INC DPTR	DPTR ← DPTR + 1
	MOV A, B	A ← B
	MOVX @DPTR, A	[DPTR] ← A
here	SJMP here.	

Assignment - 18

CUBE OF A NUMBER

Aim: To calculate the cube of a number using 8051 Microcontroller.

Algorithm:

1. Move 03H to A, B and R0
2. Multiply A x B and store in BA
3. Move R0 to B
4. Multiply A x B and store in BA
5. Move A to [4150] and B to [4151]

Sample I/O:

output: [4150] - 1B
[4151] - 00

Result: Cube of a number is calculated using 8051.

Label	Mnemonics	comments:
	MOV A, #23	A ← 23
	MOV R0, A	R0 ← 23
	ANL A, #F0	bitwise and 23 and F0
	SWAP A	swap nibbles
	MOV R1, A	R1 ← A (higher nibble)
	MOV A, R0	A ← R0
	MOV R0, #00	R0 ← 00
	ANL A, #0F	A ← 0F
	MOV R0, A.	R0 ← A (lower nibble)
	MOV A, R0	A ← R0
	ADD A, #30	A ← A + 30
	MOV DPTR, #4150	DPTR ← 4150
	MOVX @DPTR, A	[DPTR] ← A
	MOV A, R1	A ← R1
	ADD A, #30	~~A ← 30~~
	INC DPTR	A ← A + 30
	MOVX @DPTR, A	DPTR ← DPTR + 1
		[DPTR] ← A
here	SJMP here	

Assignment – 19

BCD to ASCII CONVERSION OSING ASCII

Aim: To convert BCD to ASCII using 8051

Algorithm:

1. Move input BCD value to A
2. Copy A to R0
3. Swap higher and lower nibble of A
4. Move A to R1
5. Move R0 to A
6. clear R0
7. perform ANL instruction
8. Now BCD values (2) are unpacked.
9. add 30H to get ASCII values.
10. Move R1 to memory
11. Move A to memory

Sample I/O:

output: [4150] -32
[4151] -33

Result: BCD – ASCII conversion has been implemented using 8051.