Experiment No: 02

Experiment Name: Study on the observations of MNEMONIC operations with machine code in MDA-8086 microprocessor kit.

Objectives:

1. To observe the result MNEMONIC operations with machine code in MDA-8086 microprocessor kit.

Theory:

MNEMONIC: A system such as a pattern of letters, ideas or associations which assists in remembering something. In other words, A combination of letters to suggest the operation of an instruction.

Machine code: A computer programming language consisting of binary and hexadecimal instructions which a computer can respond to directly. It uses the instruction set of particular computer architecture.

Assembly language: A medium of communication with a computer in which programs are written in mnemonics.

Instructions:

MOV: MOV copies the data in the source to the destination. The data can be either a byte or a word. Sometimes this has to be explicitly stated when the assembler cannot determine from the operands whether a byte or word is being referenced.

The MOV instruction has a few limitations:

- an immediate value cannot be moved into a segment register directly (i.e. mov ds,10)
- segment registers cannot be copied directly (i.e. mov es,ds)
- a memory location cannot be copied into another memory location (i.e. mov aNumber,aDigit)
- CS cannot be copied to (i.e. mov cs,ax)

SHAF: Loads the SF, ZF, AF, PF, and CF flags of the EFLAGS register with values from th corresponding bits in the AH register (bits 7, 6, 4, 2, and 0, respectively). Bits 1, 3, and 5 of registe AH are ignored; the corresponding reserved bits (1, 3, and 5) in the EFLAGS register

ADD AX, 4789: This adds the value 4789 to the contents of AX and leaves the result in AX.

ADC: Adds the destination operand (first operand), the source operand (second operand), and th carry (CF) flag and stores the result in the destination operand. The destination operand can be register or a memory location; the source operand can be an immediate, a register, or a memor location. (However, two memory operands cannot be used in one instruction.) The state of the C flag represents a carry from a previous addition.

SUB: Subtract the destination operand (first operand), the source operand (second operand) store the result in the destination operand. Used to subtract the byte from byte/word from word.

SBB: Subtract with borrow instruction subtract the source operand and the borrow flag (CF) whic may reflect the result of the previous calculations, from the destination operand. Used to perfor subtraction with borrow.

DEC: The decrement instruction subtracts one from the contents of the specified register o memory location .Used to decrement the provided byte/word by 1.

INC: The instruction increases the contents of the specified register or memory location b 1.Immediate data cannot be operand of this instruction .Used to increment the provided byte/wor by 1.

CBW: Used to fill the upper byte of the word with the copies of sign bit of the lower byte.

MUL: Used to multiply unsigned byte by byte/word by word.

IMUL: Used to multiply signed byte by byte/word by word.

DIV: Used to divide the unsigned word by byte or unsigned double word by word.

IDIV: Used to divide the signed word by byte or signed double word by word.

ADDRESS	MACHINE CODE	MNEMONIC
1000	B8 0000	MO
1003	9E	SAHF

AX, 0

1004 4789H	05 8947	ADD AX,
1007 6488H	15 8864	ADC AX,
100A 88H	04 88	ADD AL,
100C AH, 33H	80 D4 33	ADC
100F 3567H	2D 6735	SUB AX,
1012 8000H	1D 0080	SBB AX,
1015 45H	2C 45	SUB AL,
1017 78H	80 DC 78	SBB AH,
101A AL, FFH	B0 FF	MOV
101C	FE C0	INC AL
101E	FE C8	DEC AL

KEY

LCD

RE

MDE8086 Kit V9.5

Midas 2109-5964/5

Seg.	Oset	data
0000	1000	B8

1) MOV AX,0

RE

IP=1003 FL=0100

=.....t.....

Results verify!

+

AX=0000 BX=0000

CX=0000 DX=0000

2) SHAF

IP=1004 FL=0100

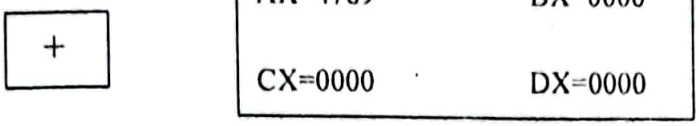
=.....t.....

3) ADD AX,4789H

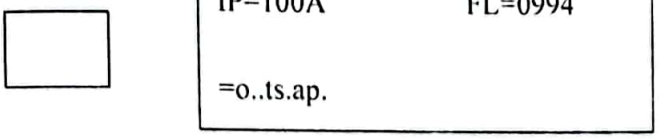
IP=1007 FL=0100

=.....t.....

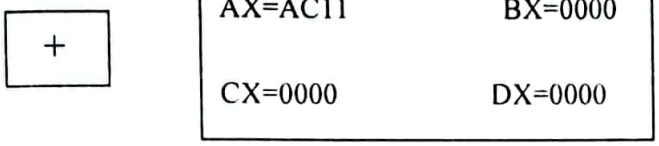
Results verify!



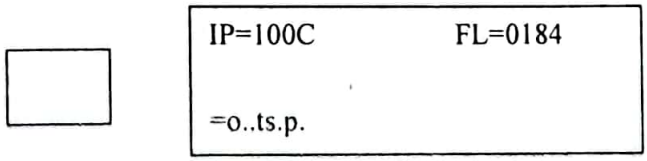
4) ADC AX,6488H



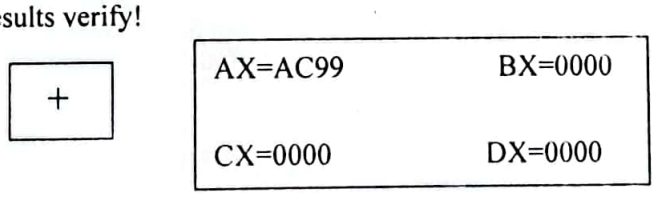
Results verify!



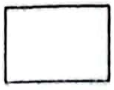
5) ADD AL,88H



Results verify!



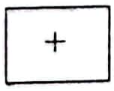
6) ADD AH,33H



IP=100F FL=0180

=..ts....

Results verify!



AX=DF99 BX=0000

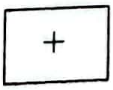
CX=0000 DX=0000

7) SUB AX,3567H

IP=10012 FL=0180

=..ts....

Results verify!



AX=AA32 BX=0000

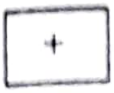
CX=0000 DX=0000

8) SBB AX,8000H

IP=10015 FL=0100

=..t....

Results verify!

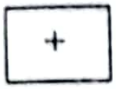


```
AX=2A32          BX=0000

CX=0000          DX=0000
```

9) SUB AL,45H

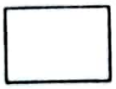
```
IP=10017         FL=0195

=..ts...apc.
```

Results verify!



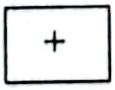
```
AX=2AED          BX=0000

CX=0000          DX=0000
```

10) SBB AH,78H



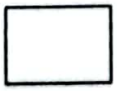
```
IP=1001A         FL=0185

=..ts...pc.
```

Results verify!



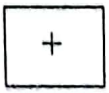
```
AX=B1ED          BX=0000

CX=0000          DX=0000
```

11) MOV AL,FFH



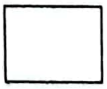
```
IP=1001C         FL=0185

=..ts...pc.
```

Results verify!



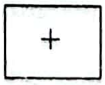
AX=B1FF	BX=0000
CX=0000	DX=0000

12) INC AL



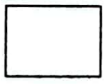
IP=101E	FL=0155
=..ts...zapc.	

Results verify!



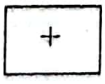
AX=B100	BX=0000
CX=0000	DX=0000

13) DEC AL



IP=1020	FL=0195
=..ts...apc.	

Results verify!



AX=B1FF	BX=0000
CX=0000	DX=0000

Experiment no: 03

Experiment no: Study on how to load the machine codes of a sample program to MDA-8086 then execute it and verify the results.

Objectives:

- To observe and verify the results of machine codes with sample programs in MDA-8086 microprocessor.

Theory:

In MDA-8086 microprocessor for various programs the respected generated machine codes will be different. Machine code is a computer program written in machine language. It uses the instruction set of a particular computer architecture. It is usually written in binary, in this case it hexadecimal. Machine code is the lowest level of software programming language. Other programming language are translated into machine code so the computer can execute it. In this particular experiment we will generate machine codes with the mnemonics using the DOSBOX software. In this software, we will give inputs as mnemonics and the respected machine codes will be generated.

Machine codes	Mnemonics
	.model small
0100	.stack 100h
0000	.data
0000	.code
0000	main proc
0000 B8 805E	mov ax, 805Eh
0003 BA 0504	mov dx, 0504h
0006 03 C2	add ax, dx
0008 BB 0050	mov bx, 0050h
000B 2B C3	sub ax, bx

```
000D 42          inc dx
000E 4B          dec bx
000F             main endp
000F             end main
```

Instruction that has been used here:

MOV: for allocating the values to the respected registers.

ADD: for adding the values of the registers.

SUB: for subtracting the values from the registers.

INC: increments the value of the registers by 1.

DEC: decrements the value of the registers by 1.

Task:

1. Generate the codes of adding two hexadecimal numbers using the DOSBOX.
   i. 67EF, 52CD
   ii. AD34, 29A7

2. Generate the codes of subtracting two hexadecimal numbers using the DOSBOX.
   i. CA45, AE49
   ii. BF98, 608A

3. Generate the codes of operation of Multiplication and Division using MUL and DI instructions.

Experiment No:04

Experiment name: Study on different types of arithmetic and logic operation in assembly language

Objectives:

a. The *objective* for this lab is to understand the fundamentals of ***logic gates*** and its use in implementing basic Boolean functions.
b. To study and understand the 3 basic gates
c. Implement basic gates
d. The study the specifications of every gate when connected it with one input constant and the other is variable.

Theory:

AND operations: The AND gate is acts in the same way as the logical "and" operator.The output is 1 only when both inputs one AND two are 1.

For example- In assembly language (8086 microprocessor kit) we operate AND operation in this following formula:

Mov Ax, 1010h	Ax=0001 0000 0001 0000
Mov Cx, 2020h	Cx=0010 0000 0010 0000
AND Ax, Cx	AND=0011 0000 0000 0000

This value of this operation will be stored in Ax register

OR operation: In OR operation, the output to be 1, at least input one OR two must be 1.

For example- we operate OR operation in assembly language by using following formula-

Mov Ax, 1010h	AX=0001 0000 0001 0000
Mov Bx, 2020h	BX=0010 0000 0010 00000
Mov Cx, 3001h	CX=0011 0000 0000 0011
OR AX,BX	OR=0011 0000 0011 0000

This value of this operation will be stored in Ax register

NOT operation:

NOT operation takes only one input. Also known as inversions or complementation.

Mov Ax, 1010h NOT Ax	Ax=0001 0000 0001 0000 NOT=1100 1111 1111 1111

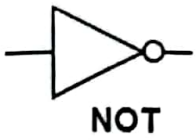
The value of this operation will be stored in Ax register.

Task:

1: Perform the XOR, XNOR TEST

2: Perfrom the NAND operation and verify it.

3: Perfrom and verify the NOR operation.

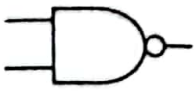


NOT

Input	Output
I	F
0	1
1	0

AND

Inputs		Output
A	B	F
0	0	0
1	0	0
0	1	0
1	1	1



NAND

Inputs		Output
A	B	F
0	0	1
1	0	1
0	1	1
1	1	0

OR

Inputs		Output
A	B	F
0	0	0
1	0	1
0	1	1
1	1	1

NOR

Inputs		Output
A	B	F
0	0	1
1	0	0
0	1	0
1	1	0

EXCLUSIVE OR

Inputs		Output
A	B	F
0	0	0
0	1	1
1	0	1
1	1	0

EXCLUSIVE NOR

Inputs		Output
A	B	F
0	0	1
0	1	0
1	0	0
1	1	1

Eqipement:

a) 8086 microprocessor
b) A Pc with DosBox software

Code:

Mov Ax, 1010h

Mov Dx , 2020h

Mov Cx, 3001h

Mov Dx,1110h

OR Ax, Bx

AND Ax, Cx

NOT Ax

Output:

Ax=0001 0000 0001 0000

Bx=0010 0000 0010 0000

OR=0011 0000 0011 0000 =3030h=Ax

Ax=0011 0000 0011 0000

Bx=0011 0000 0000 0001

AND=0011 0000 0000 0000=3000h=Ax

NOT (Ax) =1100 1111 1111 1111=0FFFh

References:

1. Internet

2. Lab manual

3. Class lecture

Experiment no: 05

Name of the experiment: Study on the different jump condition of MDA 8086 microprocessor

Objective:

- Knowledge about Jump Conditions.
- Impact of Jump Conditions in a code.

Introduction: A microprocessor is an electronic component that is used by a computer to do its work. It is a central processing unit on a single integrated circuit chip containing millions of very small components including transistors, resistors, and diodes that work together.

Theory: Jump Instructions are used for changing the flow of execution of instructions in the processor. If we want jump to any instruction in between the code, then this can be achieved by these instructions. There are two types of Jump instructions:

- Unconditional Jump Instructions
- Conditional Jump Instructions

Unconditional Jump Instructions

These instructions are used to jump on a particular location unconditionally, i.e. there is no need to satisfy any condition for the jump to take place. There are three types of procedures used for unconditional jump. They are:

- NEAR – This procedure targets within the same code segment. (Intra-segment)
- SHORT - This procedure also targets within the same code segment, but the offset is 1 byte long. (Intra-segment)
- FAR - In this procedure, the target is outside the segment and the size of the pointer is double word. (Inter-segment)

Syntax: JMP procedure name memory location

Example: JMP short target

Conditional Jumps

In these types of instructions, the processor must check for the particular condition. If it is true, then only the jump takes place else the normal flow in the execution of the statements is maintained.

The ALU operations set flags in the status word (Flag register). The conditional jump statements test the flag and jump is the flag is set.

There are following types of conditional jump instructions:

- **JC:** Stands for 'Jump if Carry'.

It checks whether the carry flag is set or not. If yes, then jump takes place, that is: If CF = 1, th jump.

- **JNC:** Stands for 'Jump if Not Carry'.

It checks whether the carry flag is reset or not. If yes, then jump takes place, that is: If CF = 0, th jump.

- **JE / JZ:** Stands for 'Jump if Equal' or 'Jump if Zero'.

It checks whether the zero flag is set or not. If yes, then jump takes place, that is: If ZF = 1, th jump.

- **JNE / JNZ:** Stands for 'Jump if Not Equal' or 'Jump if Not Zero'

It checks whether the zero flag is reset or not. If yes, then jump takes place, that is: If ZF = 0, th jump.

- **JP / JPE:** Stands for 'Jump if Parity' or 'Jump if Even Parity'

It checks whether the Parity flag is set or not. If yes, then jump takes place, that is: If PF = 1, th jump.

- **JNP / JPO:** Stands for 'Jump if Not Parity' or 'Jump if Odd Parity'

It checks whether the Parity flag is reset or not. If yes, then jump takes place, that is: If PF = then jump.

Operation Instruction:

1. **First Code:**

```
mov ax,3254h;
mov bx,1f48h;
mov cx,412ah;
add ax, cx;
jmp s7;
sub ax, bx;
s7: and ax, bx;
hlt;
```

2. **Second Code:**

```
mov ax,125bh;
mov bx,125bh;
mov cx,412ah;
sub ax, cx;
```

```
jz s5;
div bx;
s5: and ax, cx;
hlt;
```

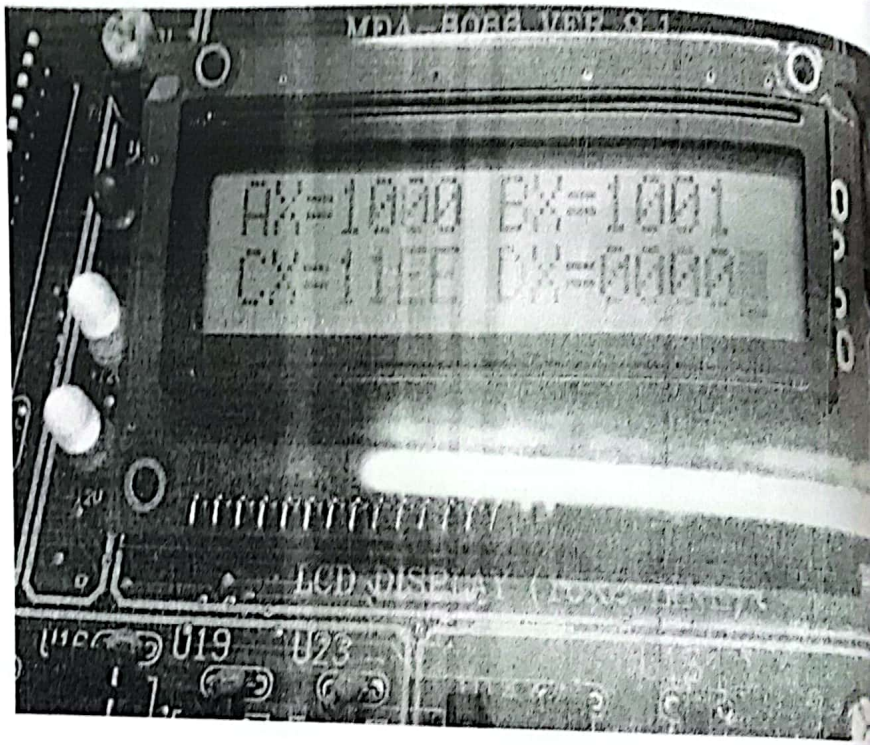
3. **Third Code:**

```
mov ax,7a24h;
mov bx,9543h;
add ax, bx;
jc BAUET;
ICE: or ax,23h;
jnz last;
BAUET: mov cx,0fc7h;
sub ax, cx;
jz ICE;
last: hlt;
```

4. **Fourth Code:**

```
mov ax, 1000h;
mov bx, 1001h;
add ax, bx;
jc polok;
polok: mov cx,11eeh;
jnc polokn;
polokn: mov dx,2233h;
jmp a5;
div cx;
a5: sub ax, cx;
jz hi;
hi: add bx, cx;
jnz hlw;
hlw: add cx, dx;
hlt;
```

Output:



Machine Code:

Microsoft (R) Macro Assembler Version 5.00 1/2/20 00:19:08

Page 1-1

```
                        .model small
0100                    .stack 100h
0000                    .data
0000                    .code
0000                    main proc
0000  B8 1000                 mov ax,1000h
0003  BB 1001                 mov bx,1001h
0006  03 C3               add ax,bx
0008  72 00               jc polok
```

```
000A  B9 11EE          polok: mov cx,11eeh
000D  73 00        jnc polokn
000F  BA 2233          polokn: mov dx,2233h
0012  EB 03 90         jmp a5
0015  F7 F1        div cx
0017  2B C1        a5: sub ax,cx
0019  74 00        jz hi
001B  03 D9        hi: add bx,cx
001D  75 00        jnz hlw
001F  03 CA        hlw: add cx,dx
0021  F4           hlt
0022               main endp
0022               end main
```