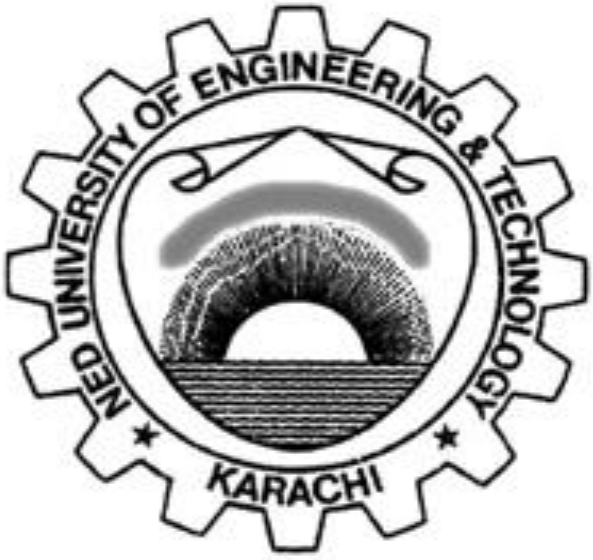
Practical Workbook

CS-353
Microprocessor and Their Applications (TCIT)



Name : _____

Year : _____

Batch : _____

Roll No : _____

Department: _____

Department of Computer & Information Systems Engineering
NED University of Engineering & Technology,

INTRODUCTION

Microprocessors play a vital role in the design of digital systems. They are found in a wide range of applications such as process control, communication systems, digital instruments and consumer products. Before embedding microprocessor in any system, profound knowledge and full understanding of the architecture and the instruction set of that microprocessor is imperative.

First two lab sessions provide an in depth coverage of the instruction set of 8088 microprocessor. In next two lab sessions an Introduction to Assembly Language programming is provided so that the students have a good knowledge of programming as well as the environments like MASM (Microsoft Macro Assembler) and TASM etc.

Further laboratory exercises enable the students to enhance their assembly language programming skills. Interfacing techniques are introduced, which gives students an opportunity to interface various I/O devices with the trainer board.

After studying the architecture and instruction set of 8088 microprocessor, students are encouraged to undertake a mini project. This project enables the students to design their own microprocessor-based system. Also students are encouraged to take project other than the one mentioned in the table of contents.

Programmable Logic Controllers (PLCs) are microprocessor-based devices used to control industrial processes or machines. They provide advanced functions, including analog monitoring, control and high speed motion control as well as share data over communication networks. Programmable Logic controllers are introduced in the last lab session. Programming PLCs and ladder design are discussed in detail.

CONTENTS

Lab Session 01

OBJECT

Introduction to Assembly Language Programming

THEORY

ASSEMBLY LANGUAGE SYNTAX

name operation operand (s) comment

Assembly language statement is classified in two types

Instruction

Assembler translates into machine code.

Example:

START: MOV CX, 5 ; initialize counter

Comparing with the syntax of the Assembly statement, name field consists of the label START:. The operation is MOV, operands are CX and 5 and the comment is ;initialize counter.

Assembler Directive

Instructs the assembler to perform some specific task, and are not converted into machine code.

Example:

MAIN PROC

MAIN is the name, and operation field contains PROC. This particular directive creates a procedure called MAIN.

Name field

Assembler translate name into memory addresses. It can be 31 characters long.

Operation field

It contains symbolic operation code (opcode). The assembler translates symbolic opcode into machine language opcode. In assembler directive, the operation field contains a pseudo-operation code (pseudo-op). Pseudo-op are not translated into machine code, rather they simply tell the assembler to do something.

Operand field

It specifies the data that are to be acted on by the operation. An instruction may have a zero, one or two operands.

Comment field

A semicolon marks the beginning of a comment. Good programming practice dictates comment on every line

Examples: MOVCX, 0 ;move 0 to CX
Do not say something obvious; so:
MOV CX, 0 ;CX counts terms, initially 0

Put instruction in context of program
; initialize registers

DATA REPRESENTATION

Numbers

11011	decimal
11011B	binary
64223	decimal
-21843D	decimal
1,234	illegal, contains a non-digit character
1B4DH	hexadecimal number
1B4D	illegal hex number, does not end with
FFFFH	illegal hex number, does not begin with digit
OFFFFH	hexadecimal number

Signed numbers represented using 2's complement.

Characters

- Must be enclosed in single or double quotes, e.g. “Hello”, ‘Hello’, “A”, ‘B’
- encoded by ASCII code
  - 'A' has ASCII code 41H
  - 'a' has ASCII code 61H
  - '0' has ASCII code 30H
  - Line feed has ASCII code OAH
  - Carriage Return has ASCII code
  - Back Space has ASCII code 08H
  - Horizontal tab has ASCII code 09H

VARIABLE DECLARATION

Each variable has a type and assigned a memory address.
Data-defining pseudo-ops

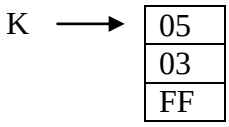
DB	define byte
DW	define word
DD	define double word (two consecutive words)
DQ	define quad word (four consecutive words)
DT	define ten bytes (five consecutive words)

Each pseudo-op can be used to define one or more data items of given type.

Byte Variables

Assembler directive format assigning a byte variable
Name DB initial value
A question mark ("?") place in initial value leaves variable uninitialized

```
I       DB   4          ;define variable I with initial value 4
J       DB   ?          ;Define variable J with uninitialized value
Name    DB   "Course"   ;allocate 6 bytes for name
K       DB   5, 3,-1    ;allocate 3 bytes
```



Other data type variables have the same format for defining the variables.
Like:
Name DW initial value

NAMED CONSTANTS

- EQU pseudo-op used to assign a name to constant.
- Makes assembly language easier to understand.
- No memory allocated for EQU names.

```
LF      EQU     0AH
        o MOV   DL, 0AH
        o MOV   DL, LF

PROMPT  EQU     "Type your name"
        o MSG   DB   "Type your name"
        o MDC   DB   PROMPT
```

INPUT AND OUTPUT USING DOS ROUTINES

CPU communicates with peripherals through I/O registers called I/O ports. Two instructions access I/O ports directly: IN and OUT. These are used when fast I/O is essential, e.g. games.

Most programs do not use IN/OUT instructions. Since port addresses vary among computer models and it is much easier to program I/O with service routines provided by manufacturer.

Two categories of I/O service routines are Basic input & output system (BIOS) routines and Disk operating system (DOS) routines. Both DOS and BIOS routines are invoked by INT (interrupt) instruction.

Disk operating system (DOS) routines

INT 21 H is used to invoke a large number of DOS function. The type of called function is specified by pulling a number in AH register.

For example

AH=1	input with echo
AH=2	single-character output
AH=9	character string output
AH=8	single-key input without echo
AH=0Ah	character string input

Single-Key Input

Input: AH=1
Output: AL= ASCII code if character key is pressed, otherwise 0.

To input character with echo:

```
MOV     AH, 1
INT     21H          read character will be in AL register
```

To input a character without echo:

```
MOV     AH, 8
INT     21H          read character will be in AL register
```

Single-Character Output

Input: AH=2,
DL= ASCII code of character to be output
Output: AL=ASCII code of character

To display a character

```
MOV     AH, 2
MOV     DL, '?'
INT     21H          displaying character'?'
```

Combining it together:

```
MOV     AH, 1
INT     21H
MOV     AH, 2
MOV     DL, AL
INT     21H          read a character and display it
```

To Display a String

Input: AH=9,
DX= offset address of a string.
String must end with a '$' character.

To display the message Hello!

```
MSG     DB    "Hello!"
MOV     AH, 9
MOV     DX, offset MSG
INT     2IH
```

OFFSET operator returns the address of a variable The instruction LEA (load effective address) loads destination with address of source
LEA DX, MSG

PROGRAM STRUCTURE

Machine language programs consist of code, data and stack. Each part occupies a memory segment. Each program segment is translated into a memory segment by the assembler.

Memory models

The size of code and data a program can have is determined by specifying a memory model using the .MODEL directive. The format is:

.MODEL memory-model

Unless there is lot of code or data, the appropriate model is SMALL

memory-model	description
SMALL	One code-segment. One data-segment.
MEDIUM	More than one code-segment. One data-segment. Thus code may be greater than 64K
COMPACT	One code-segment. More than one data-segment.
LARGE	More than one code-segment. More than one data-segment. No array larger than 64K.
HUGE	More than one code-segment. More than one data-segment. Arrays may be larger than 64K.

Data segment

A program's DATA SEGMENT contains all the variable definitions.
To declare a data segment, we use the directive .DATA, followed by variable and constants declarations.

```
.DATA
WORD1       DW      2
MASK        EQU     10010010B
```

Stack segment

It sets aside a block of memory for storing the stack contents.

```
.STACK      100H        ;this reserves 256 bytes for the stack
```

If size is omitted then by-default size is 1KB.

Code segment
Contain program's instructions.

```
.CODE       name
```

Where name is the optional name of the segment
There is no need for a name in a SMALL program, because the assembler will generate an error). Inside a code segment, instructions are organised as procedures. The simplest procedure definition is

```
name    PROC
;body of message
name    ENDP
```

An example

```
MAIN PROC
;main procedure instructions
MAIN ENDP
;other procedures go here
```

Putting it together

```
.MODEL          SMALL
.STACK          100H
.DATA
;data definition go here
.CODE
MAIN    PROC
;instructions go here
MAIN    ENDP
;other procedures go here
END             MAIN
```

The last line in the program should be the END directive followed by name of the main procedure.

A Case Conversion Program

Prompt the user to enter a lowercase letter, and on next line displays another message with letter in uppercase, as:
Enter a lowercase letter: a
In upper case it is: A

```
TITLE PGM4_1:  CASE CONVERSION PROGRAM
.MODEL  SMALL
.STACK   100H
.DATA
        CR      EQU     0DH
        LF      EQU     0AH
        MSG1    DB      'ENTER A LOWER CASE LETTER:  $'
        MSG2    DB      CR, LF, 'IN UPPER CASE IT IS:  '
        CHAR    DB      ?,'$'

.CODE
MAIN PROC
;initialize DS
        MOV     AX,@DATA  ; get data segment
        MOV     DS,AX     ; initialize DS
;print user prompt
        LEA     DX,MSG1   ; get first message
        MOV     AH,9      ; display string function
        INT     21H       ; display first message
;input a character and convert to upper case
        MOV     AH,1      ; read character function
        INT     21H       ; read a small letter into AL
        SUB     AL,20H    ; convert it to upper case
```

```
        MOV     CHAR,AL     ; and store it
;display on the next line
        LEA     DX,MSG2     ; get second message
        MOV     AH,9        ; display string function
        INT     21H         ; display message and upper case letter in front
;DOS exit
        MOV     AH,4CH      ; DOS exit
        INT     21H
MAIN    ENDP
        END     MAIN
```

Save your program with (.asm) extension.
If "**first**" is the name of program then save it as "**first.asm**"

EXERCISE:

- Explain, the term Assembly Language Statement:

  _____
  _____
  _____
  _____
  _____

- In what manner, the data will be stored in data segment in response of following statements: Let starting offset is 0000h within data segment.

  STR1 DB 'A','B','C' : _____

  STR2 DB "LAB SESSION#1 $" : _____

  DT1 DW 40, 50, 60, 70, 80 : _____

  DT2 DB 10110100b, 45o, 45 : _____

  THR DB 30h : _____

  TAB EQU 09 :_____

  NUMB1 EQU 8435h : _____

Lab Session 02

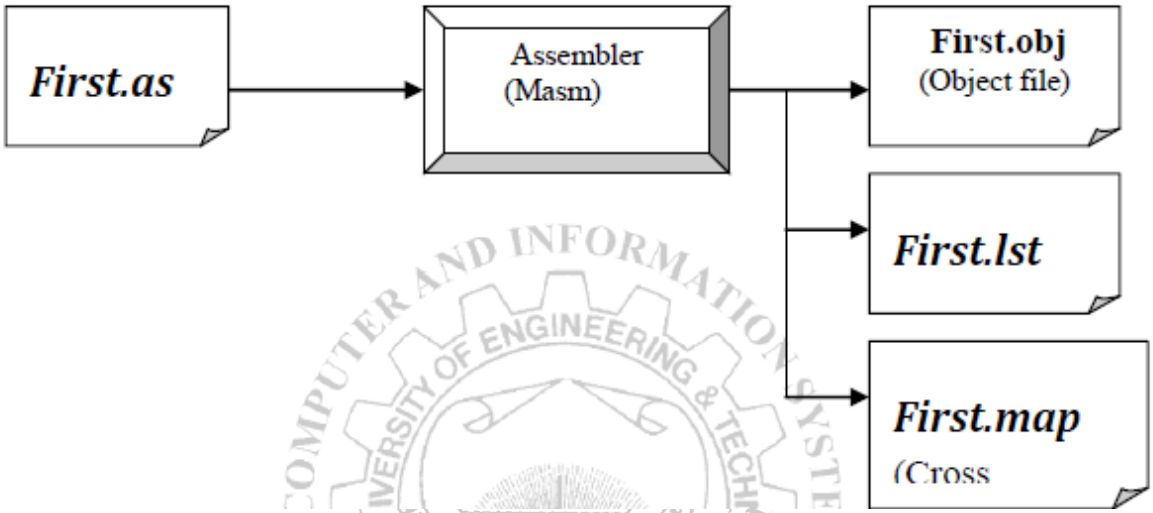
OBJECT

Running and Debugging the Assembly Program

THEORY

ASSEMBLING THE PROGRAM

Assembling is the process of converting the assembly language source program into machine language object file. The program "ASSEMBLER" does this.



Assemble the program

```
C:\>masm first.asm
Microsoft (R) Macro Assembler Version 5.10
Copyright (C) Microsoft Corp 1981, 1988. All rights reserved.
Object filename [first.OBJ]: first
Source listing [NUL.LST]: first
Cross-reference [NUL.CRF]: first
47338 + 430081 Bytes symbol space free
0 Warning Errors
0 Severe Errors
```

After assembling the program as shown above you will find two additional files with the **object file**, automatically generated by the assembler, in your directory i.e. the **list file** and the **cross-reference file**. Name must be provided for .LST else NUL (nothing) will be generated.

1. OBJECT FILE

A non-executable file contains the machine code translation of assembly code, plus other information needed to produce the executable.

2. LIST FILE

The list file is a text file that gives you assembly language code and the corresponding machine language code, a list of names used in the program, error messages and other statistics as shown below for the assembly file first.asm:

PGM4_1: CASE CONVERSION PROGRAM	Page 1-1

```
                                TITLE   PGM4_1: CASE CONVERSION PROGRAM
                               .MODEL   SMALL
                               .STACK   100H
                               .DATA
 = 000D                                        CR      EQU
                                                0DH
 = 000A                                        LF      EQU
                                                0AH
0000  45 4E 54 45 52 20   MSG1          DB      'ENTER A LOWER
                                CASE LETTER:  $'
       41 20 4C 4F 57 45
       52 20 43 41 53 45
      20 4C 45 54 54 45
      52 3A 20 20 24
001D  0D 0A 49 4E 20 55  MSG2           DB      0DH, 0AH, 'IN U
                                PPER CASE IT IS:   '
       50 50 45 52 20 43
       41 53 45 20 49 54
      20 49 53 3A 20 20
0035  00 24                     CHAR    DB      ? ,'$'
                               .CODE
0000                            MAIN    PROC
                               ; initialize DS
0000  B8 ---- R                 MOV     AX,@DATA        ; get data segment
0003  8E D8                     MOV     DS, AX          ; initialize DS
                                                        ;print user prompt
0005  8D 16 0000 R              LEA     DX,MSG1         ; get first message
0009  B4 09                     MOV     AH,9            ;       display       string
function
000B  CD 21                     INT     21H             ;       display       first
message
                                                        ;input a character and
                                                        ;convert to uppercase

000D  B4 01                     MOV     AH,1            ;       read       character
function
000F  CD 21                     INT     21H             ;read a small letter
into AL
0011  2C 20                     SUB     AL,20H          ; convert it to upper case
0013  A2 0035 R                 MOV     CHAR,AL         ; and store it
                                                        ;display on the next line
0016  8D 16 001D R              LEA     DX,MSG2         ;get second message
001A  B4 09                     MOV     AH,9            ;       display       string
function
001C  CD 21                     INT     21H             ; display message and
                                                        ;upper case letter in front
                               ;DOS   exit
```



```
001E  B4 4C                     MOV     AH,4CH          ; DOS e
```

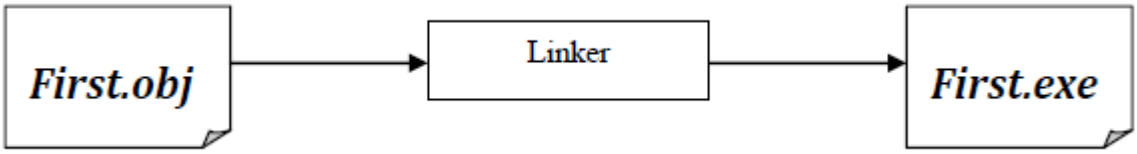
```
                                xit
   37 0020  CD 21                       INT     21H

   38 0022                      MAIN     ENDP
   39                                END     MAIN


PGM4_1: CASE CONVERSION PROGRAM                 Symbols-1


Segments and Groups:

            N a m e                 Length     Align      Combine     Class

DGROUP . . . . . . . . . . . .      GROUP
  _DATA  . . . . . . . . . . . . 0037    WORD       PUBLIC     'DATA'
  STACK  . . . . . . . . . . . . 0100    PARA       STACK      'STACK'
_TEXT  . . . . . . . . . . . . . 0022    WORD       PUBLIC     'CODE'

Symbols:

            N a m e                 Type          Value          Attr

CHAR . . . . . . . . . . . . . . . .    L BYTE      0035        _DATA
CR . . . . . . . . . . . . . . . . . .  NUMBER      000D

LF . . . . . . . . . . . . . . . . . .  NUMBER      000A

MAIN . . . . . . . . . . . . . . . .    N PROC      0000        _TEXT       Length = 0022
MSG1 . . . . . . . . . . . . . . . .    L BYTE      0000        _DATA
MSG2 . . . . . . . . . . . . . . . .    L BYTE      001D        _DATA

@CODE . . . . . . . . . . . . . . .     TEXT        _TEXT
@CODESIZE . . . . . . . . . . . .       TEXT        0
@CPU . . . . . . . . . . . . . . . .    TEXT        0101h
@DATASIZE . . . . . . . . . . . .       TEXT        0
@FILENAME . . . . . . . . . . . .       TEXT        cc
@VERSION . . . . . . . . . . . .        TEXT        510

    32 Source  Lines
    32 Total   Lines
    23 Symbols

 46146 + 447082 Bytes symbol space free

    0 Warning Errors
    0 Severe  Errors
```

3. CROSS-REFERENCE FILE

List names used in the program and the line number.

LINKING THE PROGRAM

Linking is the process of converting the one or more object files into a single executable file. The program "LINKER" does this.




```
C:\>link first.obj;
Microsoft (R) Overlay Linker Version 3.64
Copyright (C) Microsoft Corp 1983-1988. All rights reserved.
```

RUNNING THE PRORAM

On the command line type the name of the program to run.

```
C:\>first.exe
ENTER A LOWER CASE LETTER: a
IN UPPER CASE IT IS: A
```

DEBUGGING

DEBUG is a primitive but utilitarian program, supplied with MS-DOS, with a small easy to learn command set. After assembling and linking the program in previous practical, (**first.asm**) we take the **first.exe** into DEBUG.

On the MS-DOS prompt type the following command,

C:\>**DEBUG first.exe**

-

DEBUG comes back with its "-"command prompt.

Useful Commands

Commands	Description
R	to display registers
R IP	to display/change IP register
T	to execute single instruction
T 4	to execute 4 instructions
G	execute till completion

G 4	execute till address 0004
D	dump bytes in hex format
D 100	dump 128bytes starting from DS:100
D 100 104	dump from 100 to 104
E DS:0 A B C	enter Ah, Bh, Ch in bytes DS:0, DS:1, DS:2
E 25	Enter bytes interactively starting at DS: 25. Space bar moves to next byte
Q	quit from debug

To view registers and FLAGS, type "R"

```
C:\>debug first.exe
-R
AX=0000 BX=0000 CX=0030 DX=0000 SP=0100 BP=0000 SI=0000 DI=0000
DS=1189 ES=1189 SS=119C CS=1199 IP=0000
```

```
NV UP EI PL NZ NA PO NC
1199:0000 B89A11 MOV AX,119A
-
```

As we know 8086 has 14 registers, all of these registers are shown by DEBUG with different values stored in these registers.

FLAG REGISTER

The letters pairs on the fourth line are the current setting of some of the status and control FLAGS. The FLAGS displayed and the symbols DEBUG uses are the following:

Unused Flag Register Bits

SYMBOL	*FLAGS*	CLEAR (0)	SET (1)
O	Overflow Flag	NV	OV
D	Direction Flag	UP	DN
I	Interrupt Flag	DI	EI
S	Sign Flag	PL	NG
Z	Zero Flag	NZ	ZR
A	Auxiliary Flag	NA	AC
P	Parity Flag	PO	PE
C	Carry Flag	NC	CY

To change the contents of a register-for example, AX to 1245h

```
-RDX
DX 0000
:1245
```

Note:- DEBUG assumes that all numbers are expressed in **hex**. Now let us verify the change, through "R" command.

```
-RDX
DX 0000
:1245
-r
AX=0000  BX=0000  CX=0059  DX=1245  SP=0100  BP=0000  SI=0000  DI=0000
DS=1453  ES=1453  SS=1469  CS=1463  IP=0000   NV UP EI PL NZ NA PO NC
1463:0000 B86514        MOV     AX,1465
-
```

DX now contain 1245h.

The next instruction to be executed by the CPU is written on the last line with its address in the memory. Let us execute each instruction one by one using „T" trace command. But before that, just check whether the ".exe" file is representing the same assembly language program or not, using the U (unassembled) command.

```
-u
1463:0000 B86514        MOV     AX,1465
1463:0003 8ED8          MOV     DS,AX
1463:0005 8D160200      LEA     DX,[0002]
1463:0009 B409          MOV     AH,09
1463:000B CD21          INT     21
1463:000D B401          MOV     AH,01
1463:000F CD21          INT     21
1463:0011 2C20          SUB     AL,20
1463:0013 A23700        MOV     [0037],AL
1463:0016 8D161F00      LEA     DX,[001F]
1463:001A B409          MOV     AH,09
1463:001C CD21          INT     21
1463:001E B44C          MOV     AH,4C
-u 20 22
1463:0020 CD21          INT     21
1463:0022 45            INC     BP
-
```

The U command by default shows 32 bytes of program coding. The last instruction shown above is not our last program's instruction. To see the remaining instructions, specify directly some address ranges ahead. Now execute instructions one be one using T command.

```
-t

AX=1465  BX=0000  CX=0059  DX=0000  SP=0100  BP=0000  SI=0000  DI=0000
DS=1453  ES=1453  SS=1469  CS=1463  IP=0003   NV UP EI PL NZ NA PO NC
1463:0003 8ED8          MOV     DS,AX
```

AX now have the segment number of the data segment. Again press T for one more time will execute the instruction MOV DS, AX as shown on the last line above. This will initialize the data segment register with the data segment address of the program.

```
-t

AX=1465  BX=0000  CX=0059  DX=0000  SP=0100  BP=0000  SI=0000  DI=0000
DS=1465  ES=1453  SS=1469  CS=1463  IP=0005   NV UP EI PL NZ NA PO NC
```

The next command LEA DX, [0002] will load the offset address of MSG1 in DX which is 0002.

```
-t

AX=1465  BX=0000  CX=0059  DX=0002  SP=0100  BP=0000  SI=0000  DI=0000
DS=1465  ES=1453  SS=1469  CS=1463  IP=0005   NV UP EI PL NZ NA PO NC
```

Check the contents of the data segment using the D command:

```
-d
1463:0000  B8 65 14 8E D8 8D 16 02-00 B4 09 CD 21 B4 01 CD   .e..........!...
1463:0010  21 2C 20 A2 37 00 8D 16-1F 00 B4 09 CD 21 B4 4C   !, .7........!.L
1463:0020  CD 21 45 4E 54 45 52 20-41 20 4C 4F 57 45 52 20   .!ENTER A LOWER
1463:0030  43 41 53 45 20 4C 45 54-54 45 52 3A 20 20 24 0D   CASE LETTER:  $.
1463:0040  0A 49 4E 20 55 50 50 45-52 20 43 41 53 45 20 49   .IN UPPER CASE I
1463:0050  54 20 49 53 3A 20 20 00-24 00 00 00 00 00 00 00   T IS:  .$.......
1463:0060  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
1463:0070  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
-
```

We can see that the string variables initialized in the Data Segment has been successfully loaded into the memory locations as above.

Now through MOV AH, 09 and interrupt command -g 000d, MSG1will be displayed as shown below:

```
-t

AX=0965  BX=0000  CX=0059  DX=0002  SP=0100  BP=0000  SI=0000  DI=0000
DS=1465  ES=1453  SS=1469  CS=1463  IP=0005   NV UP EI PL NZ NA PO NC
```

```
-g 000d
ENTER A LOWER CASE LETTER:
AX=0924  BX=0000  CX=0059  DX=0002  SP=0100  BP=0000  SI=0000  DI=0000
DS=1465  ES=144F  SS=1465  CS=145F  IP=000D   NV UP EI PL NZ NA PO NC
```

Pressing T one more time will move 01 in AH so that we can take input.

```
-t

AX=0124  BX=0000  CX=0059  DX=0002  SP=0100  BP=0000  SI=0000  DI=0000
DS=1465  ES=144F  SS=1465  CS=145F  IP=000F   NV UP EI PL NZ NA PO NC
```

Now through interrupt command -g 0011, user will be prompted to enter a lower case letter As you can see, 'a' is entered as input, so AX will now contain 0161 where 61 is the ASCII code of 'a'.

```
-g 0011
a
AX=0161  BX=0000  CX=0059  DX=0002  SP=0100  BP=0000  SI=0000  DI=0000
DS=1465  ES=144F  SS=1465  CS=145F  IP=0011   NV UP EI PL NZ NA PO NC
```

Now the SUB command will subtract 20 out of the contents of AL to perform case conversion.

```
-t

AX=0141  BX=0000  CX=0059  DX=0002  SP=0100  BP=0000  SI=0000  DI=0000
DS=1465  ES=1453  SS=1469  CS=1463  IP=0013   NV UP EI PL NZ NA PE NC
```

Again pressing 't' will store the case conversion output i.e. 'A' in memory.

Now to display MSG2, its offset address will be loaded in DX:

```
-t

AX=0141  BX=0000  CX=0059  DX=0002  SP=0100  BP=0000  SI=0000  DI=0000
DS=1465  ES=1453  SS=1469  CS=1463  IP=0016   NV UP EI PL NZ NA PE NC
```

MOV AH, 09 and interrupt command are used to print the string on screen as done before. The result will be displayed as follows:

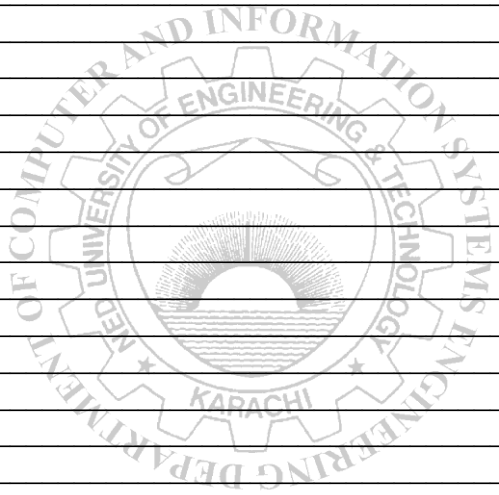
```
-t

AX=0141  BX=0000  CX=0059  DX=001F  SP=0100  BP=0000  SI=0000  DI=0000
DS=1465  ES=1453  SS=1469  CS=1463  IP=001A   NV UP EI PL NZ NA PE NC
1463:001A B409          MOV     AH,09
-t

AX=0941  BX=0000  CX=0059  DX=001F  SP=0100  BP=0000  SI=0000  DI=0000
DS=1465  ES=1453  SS=1469  CS=1463  IP=001C   NV UP EI PL NZ NA PE NC
1463:001C CD21          INT     21
-g

IN UPPER CASE IT IS:  A
Program terminated normally
```

This message indicates that the program has run to completion. The program must be reloaded to execute again. Now leave the DEBUG using "Q",

```
-q

C:\Users\admin>
```

EXEERCISE:

Write a program that asks user to enter two numbers to be added and then display the result with appropriate message on the monitor screen.


COMPUTER AND INFORMATION SYSTEMS ENGINEERING DEPARTMENT OF
NED UNIVERSITY OF ENGINEERING & TECHNOLOGY
KARACHI

Lab Session 03

OBJECT

Calling a subroutine from another assembly file as a near procedure

THEORY

Near call—A call to a procedure within the current code segment (the segment currently pointed to by the CS register), sometimes referred to as an intrasegment call.

Procedure Declaration

- The syntax of procedure declaration is the following:

  PROC *name NEAR*
  ; body of procedure
  ret
  ENDP *name*

The CALL Instruction

- CALL invokes a procedure

  call *name*

where ***name*** is the name of a procedure.

Executing a CALL

- The return address to the calling program (the current value of the IP) is saved on the stack
- IP get the offset address of the first instruction of the procedure (this transfers control to the procedure)

The RET instruction

- To return from a procedure, the instruction

  ret *pop_value*

is executed.

- The integer argument ***pop_value*** is optional.
- **ret** causes the stack to be popped into IP.

A Case Conversion Program

Prompt the user to enter a lowercase letter, and on next line displays another message with letter in uppercase, as:
Enter a lowercase letter: a
In upper case it is: A
We will create two different assembly files to implement case conversion. First file contains the code that will prompt user to enter a lower case letter. This file contains a call to a near procedure named CONVERT, which is used to perform case conversion. The second file contains the code of the procedure CONVERT. So, when the procedure CONVERT is invoked, the given lower case letter will be converted to upper case. The control will then be returned back to the calling procedure in the first file which will display the output.
Assembly code for both of the files is given below:

```
TITLE          PGM4_2: CASE CONVERSION
EXTRN  CONVERT: NEAR
.MODEL         SMALL
```

```
.STACK  100H
.DATA
MSG     DB      'ENTER A LOWER CASE LETTER:  $'
.CODE
MAIN    PROC
        MOV     AX,@DATA        ; get data segment
        MOV     DS,AX           ; initialize DS
;print user prompt
        LEA     DX,MSG                  ; get first message
        MOV     AH,9            ; display string function
        INT     21H             ; display first message
;input a character and convert to upper case
        MOV     AH,1            ; read character function
        INT     21H             ; read a small letter into AL
        CALL    CONVERT                 ; convert to uppercase
        MOV     AH,4CH
        INT     21H             ;DOS exit
MAIN    ENDP
        END     MAIN
```

Save your program with (.asm) extension. If "**first**" is the name of program then save it as "**first.asm**".

```
TITLE           PGM4_2A : CASE CONVERSION
PUBLIC CONVERT
.MODEL          SMALL
.DATA
MSG     DB      0DH, 0AH, 'IN UPPER CASE IT IS:  '
CHAR    DB      -20H,'$'
.CODE
CONVERT         PROC    NEAR
;converts char in AL to uppercase
        PUSH    BX
        PUSH    DX
        ADD     CHAR,AL
        MOV     AH,9
        LEA     DX,MSG
        INT     21H
        POP     DX
        POP     BX
        RET
CONVERT         ENDP
        END
```

Save the above program as well with (.asm) extension. If "**second**" is the name of program then save it as "**second.asm**".

Now follow the steps as mentioned in the previous lab session to assemble the two files. First perform all the steps to assemble and create .obj file for the first program, list file and cross reference file will also be

generated automatically by the assembler for the first program. Now, do the same for the second program. Observe the list files for both the programs yourself.

Now we have to link the two files. For this, write the following line on the command prompt:

>link first + second

Then give any name to the resultant file (e.g.: first). Now we have a single .exe file to perform case conversion. Write following line on the command prompt:

>debug first.exe

Check whether the .exe file is representing the same assembly language program or not, using the U (unassembled) command.

```
0BA7:0000 B8A90B        MOV     AX,0BA9
0BA7:0003 8ED8          MOV     DS,AX
0BA7:0005 8D160A00      LEA     DX,[000A]
0BA7:0009 B409          MOV     AH,09
0BA7:000B CD21          INT     21
0BA7:000D B401          MOV     AH,01
0BA7:000F CD21          INT     21
0BA7:0011 E80400        CALL    0018
0BA7:0014 B44C          MOV     AH,4C
0BA7:0016 CD21          INT     21
0BA7:0018 53            PUSH    BX
0BA7:0019 52            PUSH    DX
0BA7:001A 00064000      ADD     [0040],AL
0BA7:001E B409          MOV     AH,09
```

The U command by default shows 32 bytes of program coding. To see the remaining instructions, specify directly some address ranges ahead.

To see initial condition of registers, type R command.

```
-r
AX=0000  BX=0000  CX=0062  DX=0000  SP=0100  BP=0000  SI=0000  DI=0000
DS=0B97  ES=0B97  SS=0BAE  CS=0BA7  IP=0000   NV UP EI PL NZ NA PO NC
0BA7:0000 B8A90B        MOV     AX,0BA9
```

Now execute instructions one be one using T command.

```
-t

AX=0BA9  BX=0000  CX=0062  DX=0000  SP=0100  BP=0000  SI=0000  DI=0000
DS=0B97  ES=0B97  SS=0BAE  CS=0BA7  IP=0003   NV UP EI PL NZ NA PO NC
0BA7:0003 8ED8          MOV     DS,AX
-t

AX=0BA9  BX=0000  CX=0062  DX=0000  SP=0100  BP=0000  SI=0000  DI=0000
DS=0BA9  ES=0B97  SS=0BAE  CS=0BA7  IP=0005   NV UP EI PL NZ NA PO NC
0BA7:0005 8D160A00      LEA     DX,[000A]                       DS:000A=4E45
```

Through above commands, we have initialized the data segment, verify by using D command.

```
-d ds: 0 41
0BA9:0000  8D 16 28 00 CD 21 5A 5B-C3 00 45 4E 54 45 52 20   ..(..!Z[..ENTER
0BA9:0010  41 20 4C 4F 57 45 52 20-43 41 53 45 20 4C 45 54   A LOWER CASE LET
0BA9:0020  54 45 52 3A 20 20 24 00-0D 0A 49 4E 20 55 50 50   TER:  $...IN UPP
0BA9:0030  45 52 20 43 41 53 45 20-49 54 20 49 53 3A 20 20   ER CASE IT IS:
0BA9:0040  E0 24                                             .$
-
```

You can see in the above figure that the data segment is initialized with the messages. Now execute the assembly and interrupt commands and note down the observations stepwise.

EXERCISE 1

Write a program that takes two numbers as input and performs addition or subtraction (asks user to select any one operation). The code for addition/subtraction of the numbers should be present in another assembly file that should be called as a near procedure in the first file.

Lab Session 04

OBJECT

Introduction to the trainer.

THEORY

The MC 8088/EV microcomputer trainer is a microprocessor controlled educational system, based on 8088, conceived to face any problem concerning the study and use of microprocessor systems.

The 8088 is one of the most common microprocessors and so it can be of help for studying the structure and general function of PCs. Consequently a fundamental step in the evolution of PCs is the introduction, by IBM of this kind of microprocessor into the PC "IBM PC" in 1981.

The basic MC8088/EV contains all the necessary components for the study of this kind of systems (8088 microprocessor, RAM and EPROM memory, liquid crystal display and keyboard, serial and parallel interface, analog inputs and outputs, troubleshooting section).

Technical characteristics of the trainer are:

- 8088/4.77 MHz microprocessor;
- 16 Kbytes system EPROM;
- 16*2 Kbyte user EPROM;
- 2 Kbyte RAM memory expandable to 6 Kbyte;
- Keyboard (USA type PC keyboard);
- Liquid crystal display (max 40 characters : 2 lines with 20 characters each);
- Buzzer;
- Cassette recorder interface;
- CENTRONICS parallel interface;
- 8 bit IN/OUT parallel ports;
- serial interface (standard RS-232);
- BUS expansion interface;
- Analog output with 8-bit D/A converter;
- Analog input with 8-bit A/D converter;
- Device for troubleshooting (Num.max.=8);
- 8+2 logic probes for fault insertion;
- Power supplies: 5V/3A, +/-12V/0.3A;
- EPROM monitor with:
    - Register display and edit
    - Memory display and edit
    - Continuous, step-by-step, break-points program run

- Load and save on cassette recorder.

General operation:

All the system's operations are controlled by microprocessor 8088 (IC1). The clock is generated by an oscillator composed by inverters TTL-7404 (IC15) and by the system quartz (14.318 MHz). With the two J-K flip flops included in IC 74107 the original frequency is divided to obtain the microprocessor clock.
The general RESET line, used by UART also, is short circuited to ground by a condenser switching on the system (logic level "0") while this line returns to logic level "1" after few m-seconds.

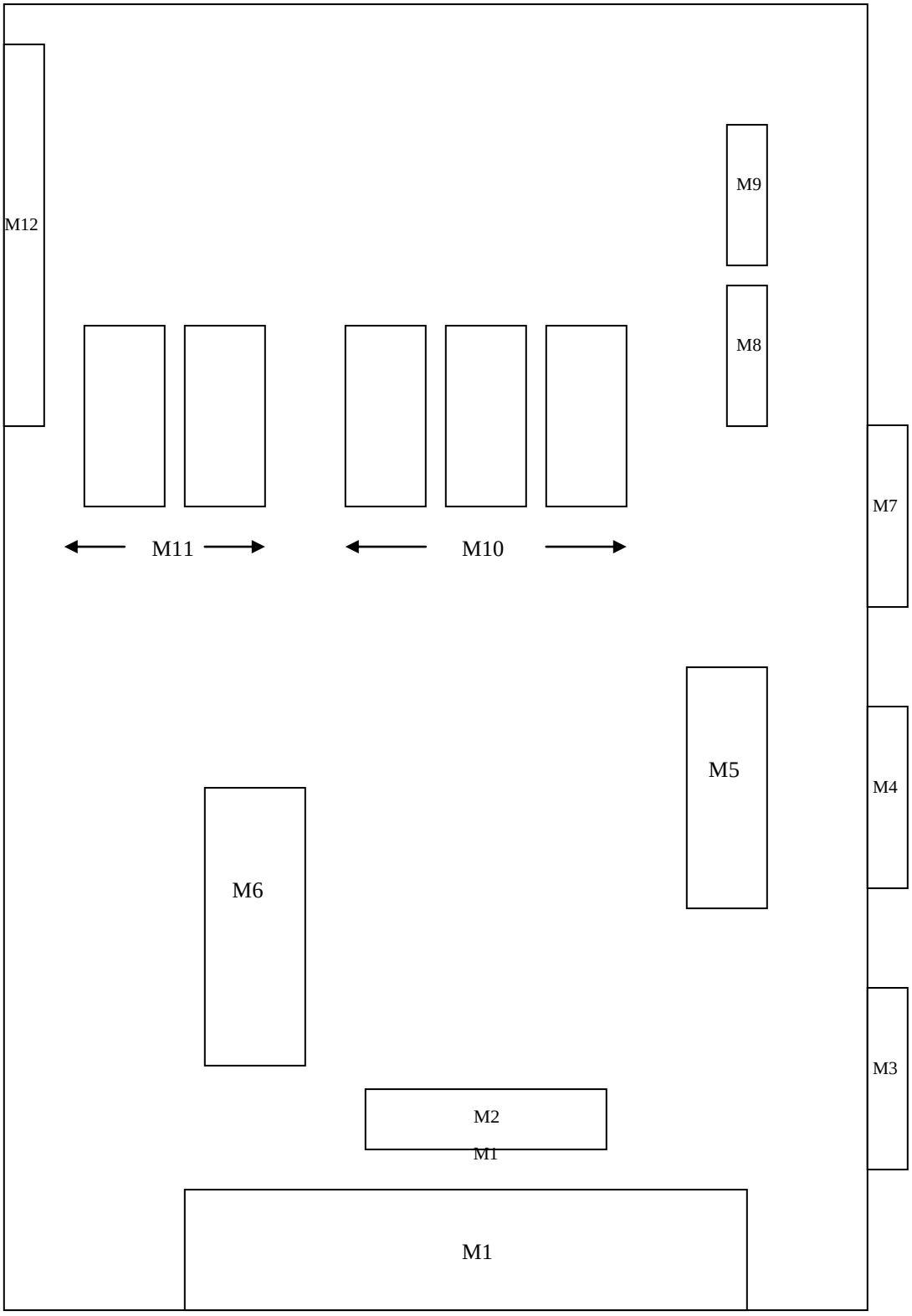
The data, addresses and control lines bus are buffered with ICs type 74244, 74245 and 74373 (IC3, IC2, IC4, IC8, IC16).

The selection among the devices concerned with the processor (EPROM memory, RAM, I/O ports…) is made by ICs type IC17, IC19, IC21, IC22, IC23 and IC24.

These components type 74139 and 74138 are line decoders, 21N – 4OUT and 3IN – 8OUT respectively. The logic combination of the two or three input lines selects one of the four or eight possible outputs and the selected device because these lines are connected to the devices enable ones.

EXERCISES

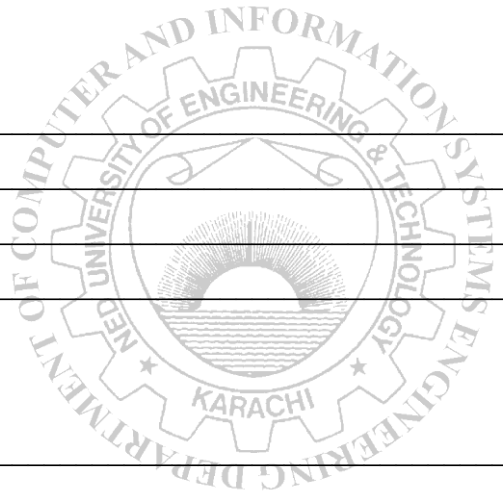
Identify the modules M1 to M12 by writing their names on the figure below. Describe each module in the space provided for this purpose.



Module M1:

Module M2:

Module M3:



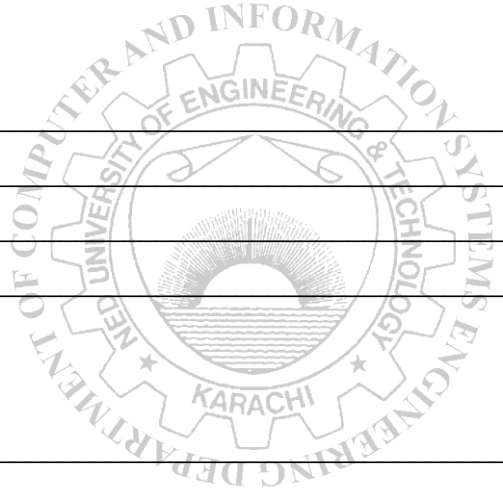
Module M4:

Module M5:

Module M6:

Module M7:

Module M8:



Module M9:

Module M10:

Module M11:

Module M12:

Lab Session 05

OBJECT

Using the trainer.

THEORY

The monitor commands are given below:

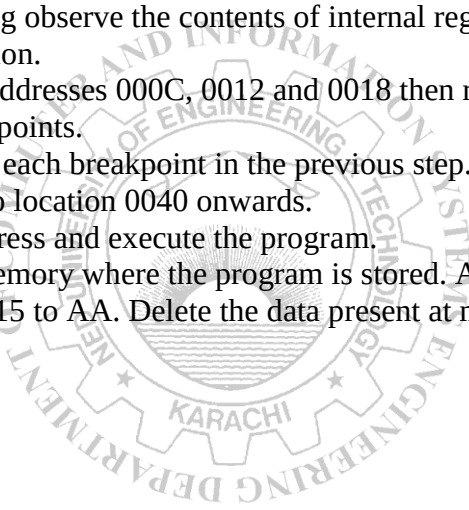
Command	*Name*	*Purpose*	*Syntax*
A	Assembler	To let the user to type 8088 assembly language programs into memory and assemble them into machine code line by line.	A A <addr>
L	Disassembler	To translate (disassemble) a block of memory into 8088 assembly instructions.	L L <addr1> L <addr1> / <n> L <addr1> <addr2>
G	Go	To execute a program in memory.	G G <addr>
S	Step	To single-step a program or execute a specified number of instructions and then stop with a display of register contents on the screen; execution starts from the address pointed to by the CS regirter and the IP register.	S S n
B	Breakpoint	To set up to three breakpoints or display their current settings. When a program is on execution and runs into a breakpoint address, the program execution will be halted.	B B <n> B <n> <addr>

C	Cancel Breakpoint	To cancel one or all of the breakpoints set previously.	C C <n>
X	Register	To display or change the contents of any of the registers.	X X <register name>
M	Memory	To display or change the contents of a memory location or a range of memory location.	M M <addr1> M <addr1> <addr2> M <addr1> <addr2> / <data1> /
I	Insert	To insert data into a memory location or a portion of memory locations.	I I /<data1> [data2] …/ I <addr1>
D	Delete	To delete a byte of data or a segment of data in memory.	D D / <n> D <addr1> / <n>
F	Find	To search for a specified value or set of values in memory.	F / <datastring> F <addr1> / <datastring> F <addr1> <addr2> / <datastring>
J	Jump	To directly jump to the particular address from which program execution must start.	(1) J <addr>.
T	Transfer	To copy a range of memory contents to another area in memory.	T <addr1> <addr2> <addr3> T <addr4> <addr5> / <n>
P	Pause	To adjust the speed of displaying on the screen.	(1) P <n>
N	Input	To input and display in hexadecimal one byte of data from the specified port.	(1) N <port_address>
O	Output	To send one or more	(1) O <port_address> / <data>

		bytes of data to a specified output port.	
W	Write	To record the contents of a range of memory on tape.	(1) W <addr1> <addr2> / <file_name>
R	Read	To read the contents from tape and copy in the memory.	R / <file_name> R <addr> / <file_name> R R <addr>

EXERCISE

1. Write down the machine code for the program after passing through Assembler and also write the output of Disassembler.
2. By using single stepping observe the contents of internal registers of microprocessor during program execution.
3. Set breakpoints at the addresses 000C, 0012 and 0018 then run the program to the end by Canceling the breakpoints.
4. Display the registers at each breakpoint in the previous step.
5. Transfer the program to location 0040 onwards.
6. Now jump to 0040 address and execute the program.
7. Note the contents of memory where the program is stored. Also change the contents of memory location 0015 to AA. Delete the data present at memory location 0008.

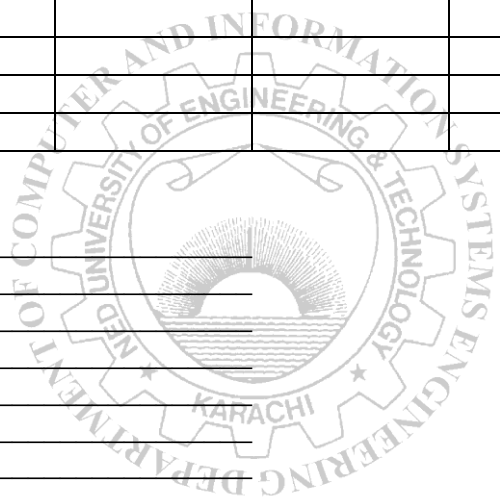



```
MOV   AX , 1111
MOV   BX , 0200
MOV   CX , 3333
MOV   DX , 4444
MOV  WORD [0200] , 6A9E
MOV  DX , [0200]
MOV  CX , DX
MOV  AL , [0200]
MOV  [0100] , AL
INT  7
```

OBSERVATIONS

Observe the contents of registers by using single stepping and record them. (Task 2)

Register	After 1st instruction	After 2nd instruction	After 3rd instruction	After 4th instruction	After 5th instruction
AX					
BX					
CX					
DX					
DS:[0200]					
DS:[0100]					
Register	After 6th instruction	After 7th instruction	After 8th instruction	After 9th instruction	
AX					
BX					
CX					
DX					
DS:[0200]					
DS:]0100]					



Lab Session 06

OBJECT

Learning Data transfer and Stack operation instructions.

THEORY

Opcode of following MOV instructions: 100010dw oorrrmmm disp

MOV reg1 , reg2 ; copy the contents of 8-bit register "reg2" in the 8-bit register "reg1".
MOV mem , reg ; copy the contents of 8-bit register "reg" in memory location "mem".
MOV reg , mem ; copy the contents of memory location "mem" into the register "reg".

Opcode of following MOV instruction: 100010dw oorrrmmm disp data

MOV mem , imm ; copy the immediate data "imm" into memory location "mem".

Opcode of following MOV instruction: 1011wrrr data

MOV reg , imm ; copy the immediate data "imm" into the register "reg".

Opcode of following MOV instructions: 101000dw disp

MOV mem , acc ; copy the contents of accumulator into memory location "mem".
MOV acc , mem ; copy the contents of memory location "mem" into accumulator.

Instruction	opcode	Description
PUSH reg	01010rrr	pushes the contents of register "reg" onto the stack.
PUSH mem	11111111 oo110mmm disp	pushes the contents of memory location "mem" onto the stack.
PUSH seg	00sss110	pushes the contents of segment register "seg" onto the stack.
PUSH imm	011010s0 data	pushes the immediate data "imm" onto the stack.
PUSHA/PUSHAD	01100000	pushes all the registers onto the stack
PUSHF/PUSHFD	10011100	pushes the flags onto the stack.
POP reg	01011rrr	pops the contents of register "reg" from top of the stack.

POP mem	10001111 oo000mmm disp	pops the contents of memory location "mem" from top of the stack.
POP seg	00sss111	pops the contents of segment register "seg" from top of the stack
POPA/POPAD	01100001	pops all the registers from the stack.
POPF/POPFD	10010000	pops the flags from the stack.

PUSHA and POPA instructions are not available in 8008 microprocessor.

ASSEMBLY PROGRAM

1. MOV AX , B386
2. MOV BX , 0200
3. MOV CX , 0A5C
4. MOV DX , D659
5. MOV BP , 0300
6. MOV ES , CX
7. MOV WORD[0200], 95D8
8. ADD AX , BX
9. PUSH AX
10. PUSH [BX]
11. PUSH DS
12. PUSHF
13. PUSH DX
14. POP CX
15. POP DI
16. POP ES
17. POP [BP]
18. POPF
19. INT 7

OBSERVATIONS

By using single stepping observe the working of the program.

Inst#	AX	BX	CX	DX	Flag	BP	SP	ES	DS	DI	[0200]	[0300]
7th												
8th												
13th												
14th												
15th												
16th												
17th												
18th												

Note the contents of the SS: SP register after 13th instruction and **then** examine the contents of the corresponding memory locations pointed out to by SS:SP.

EXERCISE 1

Write a program, which

1. Loads AX, BX, CX and DX registers with A154, 7812, 9067, BFD3.
2. Exchange lower byte of CX and higher byte of DX registers by using memory location 0150 in between the transfer. Then stores CX and DX registers onto memory location 0170 onward.
3. Exchange higher byte of AX and higher byte of BX registers by using memory location 0160 in between the transfer. Then stores AX and BX registers onto memory location 0174 onward.
4. Also draw the flow chart of the program.

Program Flowchart

OBSERVATIONS 1

- Observe the contents of memory location from 0170 to 0177 and record them below in a table.
- Observe the contents of registers by using single stepping and record the final contents below.

Contents of memory location

Contents of Registers

AX	
BX	
CX	
DX	

EXERCISE 2

Write a program that produces certain delay and then increment the Accumulator register. When accumulator produces a carry then the buzzer should generate tone for a certain time. Implement this program using subroutine. The length of delay is passed to the delay subroutine as a parameter, using stack. Also draw the flowchart. You can also use any assembler for this exercise.

Program

Flowchart

Lab Session 07

OBJECT

Learning Logic group of instructions (AND, OR and XOR).

THEORY

Opcode	Inst.	Operand1, Operand2	Description
001000dw oorrrmmm disp	AND	reg/ mem**,** reg/ mem	Perform logical operation on register/memory with the memory or the second register. Both the two operands cannot be the memory location.
000010dw oorrrmmm disp	OR		
001100dw oorrrmmm disp	XOR		
100000sw oo100mmm disp data	AND	reg/mem/acc**,** imm	Perform logical operation on the "immediate value" with the contents of the register / memory location or specifically the accumulator.
100000sw oo001mmm disp data	OR		
100000sw oo100mmm disp data	XOR		

ASSEMBLER PROGRAM

1. MOV AX, 8A53
2. MOV BX, 0200
3. MOV CX, 692D
4. MOV DX, E6CB
5. MOV WORD [BX], 7B8A
6. AND AX, BX
7. AND CX, [BX]
8. OR [BX], CX
9. OR WORD [BX], 6F0C
10. XOR AX, 94D7
11. XOR DX, C4D1
12. INT 7

OBSERVATIONS

By using single stepping record the contents of following registers:

Register	After 5th instruction	After 6th instruction	After 7th instruction	After 8th instruction	After 9th instruction	After 10th instruction	After 11th instruction
AX							
BX							
CX							
DX							
Flag							
Word[0200]							

EXERCISE 1

Write a program which mask the bits of AX register, by setting left-most 4 bits ,resetting right most 4 bits and complement bit position number 9 and 10.(Hint: Use AND,OR and XOR instructions for masking).

Program Flowchart

EXERCISE 2

An ASCII coded number can be converted to BCD by masking. Write a program ,which converts ASCII 30H - 39H to BCD 0-9. Use any assembler for this exercise.

Program Flowchart

Lab Session 08

OBJECT

To study the shift and rotate instructions present in 8088 instruction set.

THEORY

Description	Instruction Op-code			TTT value
	1101000w ooTTTmmm disp	1101001w ooTTTmmm disp	1101001w ooTTTmmm disp	
	Shift/otate one time	Shift/Rotate according to the contents of the CL register	Shift/Rotate according to the immediate memory location "mem"	
Rotate left without carry	*ROL reg/mem , 1*	*ROL reg/mem , CL*	*ROL reg/mem , imm*	000
Rotate right without carry	*ROR reg/mem , 1*	*ROR reg/mem , CL*	*ROR reg/mem , imm*	001
Rotate left with carry	*RCL reg/mem , 1*	*RCL reg/mem , CL*	*RCL reg/mem , imm*	010
Rotate right with carry	*RCR reg/mem , 1*	*RCR reg/mem , CL*	*RCR reg/mem , imm*	011
Shift logic left	*SAL reg/mem , 1*	*SAL reg/mem , CL*	*SAL reg/mem , imm*	100
Shift Arithmetic left	*SHL reg/mem , 1*	*SHL reg/mem , CL*	*SHL reg/mem , imm*	″
Shift logic right	*SHR reg/mem , 1*	*SHR reg/mem , CL*	*SHR reg/mem , imm*	101
Shift arithmetic right	*SAR reg/mem , 1*	*SAR reg/mem , CL*	*SAR reg/mem , imm*	111

ASSEMBLER PROGRAM

```
1.  0000   MOV   AX , 1111
2.  0003   MOV   BX , 2222
3.  0006   MOV   CX , 3303
4.  000C   MOV   SI , 9254
5.  000F   MOV WORD [100] , 6655
6.  0015   MOV BYTE[123] , 77
7.  001A   MOV WORD [126] , 9988
8.  0020   ROL    AX , 1
9.  0022   ROL   BYTE [100] , 1
10. 0026   ROL   AX , CL
11. 0028   ROL   BYTE [100] , CL
12. 002C   RCL   BX , 1
13. 002E   RCL   WORD [100] , 1
```

14. 0032 RCL AX , CL
15. 0034 RCL WORD [100] , CL
16. 0038 ROR AX , 1
17. 003A ROR AX , CL
18. 003C ROR BYTE [126] , CL
19. 0040 RCR BX , 1
20. 0042 RCR BYTE [127] , CL
21. 0046 SHL BX , 1
22. 0048 SHL BYTE [126] , CL
23. 004C SAR SI , 1
24. 004E SAR SI ,CL
25. 0050 SHR BYTE [123] , 1
26. 0054 SHR BYTE [123] , CL
27. 0058 INT 7

OBSERVATIONS

By using single stepping observe the contents of the registers and memory locations that are used to store data in the program.

	AX	BX	SI	CF	Memory Locations				
					100	101	123	126	127
7.	_____	_____	_____	_____	_____	_____	_____	_____	_____
8.	_____	_____	_____	_____	_____	_____	_____	_____	_____
9.	_____	_____	_____	_____	_____	_____	_____	_____	_____
10.	_____	_____	_____	_____	_____	_____	_____	_____	_____
11.	_____	_____	_____	_____	_____	_____	_____	_____	_____
12.	_____	_____	_____	_____	_____	_____	_____	_____	_____
13.	_____	_____	_____	_____	_____	_____	_____	_____	_____
14.	_____	_____	_____	_____	_____	_____	_____	_____	_____
15.	_____	_____	_____	_____	_____	_____	_____	_____	_____
16.	_____	_____	_____	_____	_____	_____	_____	_____	_____
17.	_____	_____	_____	_____	_____	_____	_____	_____	_____
18.	_____	_____	_____	_____	_____	_____	_____	_____	_____
19.	_____	_____	_____	_____	_____	_____	_____	_____	_____
20.	_____	_____	_____	_____	_____	_____	_____	_____	_____
21.	_____	_____	_____	_____	_____	_____	_____	_____	_____
22.	_____	_____	_____	_____	_____	_____	_____	_____	_____
23.	_____	_____	_____	_____	_____	_____	_____	_____	_____
24.	_____	_____	_____	_____	_____	_____	_____	_____	_____
25.	_____	_____	_____	_____	_____	_____	_____	_____	_____
26.	_____	_____	_____	_____	_____	_____	_____	_____	_____

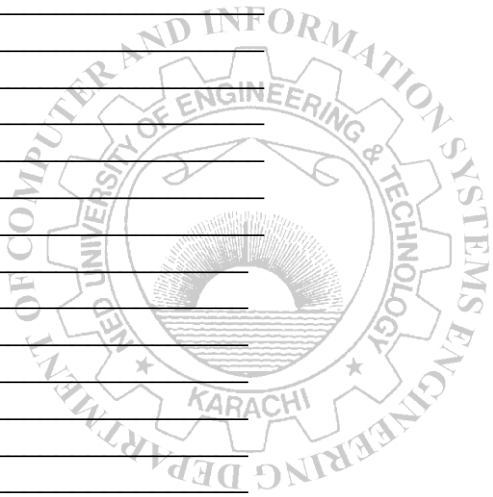
EXERCISE

Write a program, which multiply two 8-bit numbers using add and shift logic. Check the program by

(i) loads accumulator with 20H and then multiply it by 10H.
(ii) loads BL with 10H and multiply it by 12H.

Use any assembler of your choice for this purpose.
Also draw the flow chart of the program.

Program Flowchart



OBSERVATIONS 1

Value of the Multiplicand = ----------.
Value of the Multiplier = ----------.
Result of Multiplication = ----------.

OBSERVATIONS 2

Value of the Multiplicand = ----------.
Value of the Multiplier = ----------.
Result of Multiplication = ----------.

Lab Session 09

OBJECT

Studying Transfer of control instructions (Conditional & Un-Conditional jumps).

THEORY

Jump Instructions transfers the control of program to the location addressed by the specified location (as listed in description column)

Instruction	Opcode	Description
JMP label (short)	11101011 disp	IP+disp
JMP label (near)	11101001 disp	
JMP label (far)	11101010 IPnew CSnew	Label
JMP reg (near)	11111111 oo100mmm	contents of register "reg"
JMP mem (near)		memory location "mem"
JMP mem (far)	11111111 oo101mmm	
Jcnd label (8-bit disp)	0111cccc disp	IP+disp; when condition "cnd" becomes true
Jcnd label (16-bit disp)	00001111 1000cccc disp	

Condition Codes	Mnemonic	Flag	Description
0000	JO	O = 1	Jump if overflow
0001	JNO	O = 0	Jump if no overflow
0010	JB/JNAE	C = 1	Jump if below
0011	JAE/JNB	C = 0	Jump if above or equal
0100	JE/JZ	Z = 1	Jump if equal/zero
0101	JNE/JNZ	Z = 0	Jump if not equal/zero
0110	JBE/JNA	C = 1 + Z = 1	Jump if below or equal
0111	JA/JNBE	O = 0 . Z = 0	Jump if above
1000	JS	S = 1	Jump if sign
1001	JNS	S = 0	Jump if no sign
1010	JP/JPE	P = 1	Jump if parity
1011	JNP/JPO	P = 0	Jump if no parity
1100	JL/JNGE	S . O	Jump if less than
1101	JGE/JNL	S = 0	Jump if greater than or equal
1110	JLE/JNG	Z = 1 + S . O	Jump if less than or equal
1111	JG/JNLE	Z = 0 + S = O	Jump if greater than

ASSEMBLER PROGRAM 1

```
INT 8       Console In (Input a character from the keyboard and store it into the AL reg.
INT B       Console Out (Output a character contained in AL to the LCD.
JMP 0000    Jump to the first instruction.
```

OBSERVATIONS 1

By using single stepping observe the working of the program. Record the content of the AX registers.

	Character	***AX***
1		
2		
3		
4		
5		

ASSEMBLER PROGRAM 2

```
MOV   AX, 0000
MOV   BX, 0000
INT 8               ;Input from Keyboard
INT B               ;Output the character
MOV   BL, AL
INT 8               ;Input from Keyboard
INT B               ;Output the character
CMP AX, BX          ;Compare the values in AX and BX
JNZ     0000        ;if not equal jump to start of program.
INT 7
```

OBSERVATIONS 2

By using single stepping observe the contents of registers AX, BX after execution of each instruction.

	(Different Key input)		(Same Key Input)	
	AX	**BX**	**AX**	**BX**
After 1st instruction	_____	_____	_____	_____
After 2nd instruction	_____	_____	_____	_____
After 3rd instruction	_____	_____	_____	_____
After 4th instruction	_____	_____	_____	_____
After 5th instruction	_____	_____	_____	_____
After 6th instruction	_____	_____	_____	_____
After 7th instruction	_____	_____	_____	_____
After 8th instruction	_____	_____	_____	_____
After 9th instruction	_____	_____	_____	_____
Flag register after 8th instruction	_____	_____	_____	_____

EXERCISE

Write a program, which prints your name on the LCD display when 'space' key is pressed from the keyboard. Implement using conditional jump instruction. Also draw the flow chart of the program.

Program Flowchart

_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____

Lab Session 10

OBJECT

Learning Isolated I/O instructions.

THEORY

IN acc , pt opcode = 1110010w port# ; Takes an 8-bit binary number as input from input port "port#" and stores that in Accumulator.

IN acc , DX opcode = 1110110w ; Takes an 8-bit binary number as input from input port addressed by DX and stores that in Accumulator.

OUT pt , acc opcode = 1110010w port# ; Outputs an 8-bit number from Accumulator to output port number "port#".

OUT DX , acc opcode = 1110111w ; Outputs an 8-bit number from Accumulator to output port addressed by DX.

ASSEMBLER PROGRAM

INPUT PORT

```
MOV  AX , 0
MOV  DX , 1A3
IN  AL , DX
INT  7
```

OUTPUT PORT

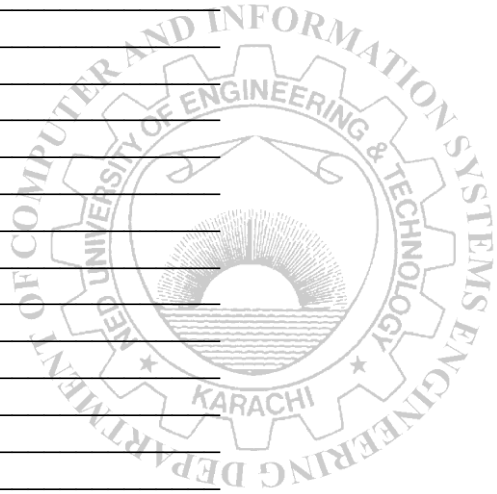
```
MOV  AL , 41
MOV  DX , 1A1
OUT  DX , AL
INT  7
```

S. No.	AL	Character
1		
2		
3		
4		
5		

EXERCISE

Write a program, which output the first ten numbers of Fibonacci series. You can also use any assembler for this exercise. (Hint: Use looping technique, to output numbers one by one in each iteration of loop)

Program Flowchart



Lab Session 11

OBJECT

Learning Arithmetic group of instructions (Add, Subtract, Multiply and Divide).

THEORY

Opcode	Inst.	Operand1, Operand2	Description
000000/000101dw oorrrmmm disp	ADD/SUB	reg1, reg2 OR mem, reg OR reg, mem	add / subtract (with carry/borrow) the contents of the register "reg" or "mem" with / from the register "reg" or "mem"
000100/000110dw oorrrmmm disp	ADC/SBB		
100000sw oo000/101mmm disp data	ADD/SUB	reg, imm OR mem, imm OR acc, imm	add / subtract (with carry/borrow) the immediate data "imm" with / from register / memory location or specifically the accumulator.
100000sw oo010/011mmm disp data	ADC/SBB		

Opcode of following MUL instructions: 1111011w oo100mmm disp

MUL reg ; multiply the contents of register "reg" with the accumulator register and return the result in "AH and AL" or "DX and AX".
MUL mem ; multiply the contents of memory "mem" with the accumulator register and return the result in "AH and AL" or "DX and AX".

Opcode of following DIV instructions: 1111011w oo110mmm disp

DIV reg ; divide the contents of the accumulator register by the contents of register "reg" and return the remainder in AH and the quotient in AL or the remainder in DX and the quotient in AX.
DIV mem ; divide the contents of the accumulator register by the contents of memory location "mem" and return the remainder in AH and the quotient in AL or the remainder in DX and the quotient in AX.

ASSEMBLER PROGRAM 1 (Add & Subtract)

ADDITION:

```
MOV  AX , 4000
MOV  BX , 0006
MOV  CX , 8
ADC  AX , BX
LOOP   0009
INT   7
```

SUBTRACTION

```
MOV  AX , 4000
MOV  BX , 0006
MOV  CX , 8
SBB  AX , BX
LOOP   0009
INT   7
```

OBSERVATIONS 1

- Using single stepping record the contents of AX register until CX becomes zero

Addition:

CX	AX	CX	AX	CX	AX

Subtraction:

CX	AX	CX	AX	CX	AX

ASSEMBLER PROGRAM 2 (MULTIPLY AND DIVIDE)

MULTIPLICATION

(8-bit)

```
MOV  AX , FF
MOV  CL , 6
MUL  CL
INT  7
```

(16-bit)

```
MOV  AX , FFFF
MOV  CX , 0200
MUL  CX
INT  7
```

DIVISION

(8-bit)

```
MOV  AX , 0400
MOV  CL , 6
DIV  CL
INT  7
```

(16-bit)

```
MOV  DX , 23
MOV  AX , 4
MOV  CX , 300
DIV  CX
INT  7
```

OBSERVATIONS 2

Record values of AX, BX, CX & DX before & after execution of MUL/DIV instruction.

For Multiplication

8-bit:

Before Execution of MUL:

AX : _____ , BX : _____
CX : _____ , DX : _____

After Execution of MUL:

AX : _____ , BX : _____
CX : _____ , DX : _____

16-bit:

Before Execution of MUL:

AX : _____ , BX : _____
CX : _____ , DX : _____

After Execution of MUL:

AX : _____ , BX : _____
CX : _____ , DX : _____

For Division

8-bit:
Before Execution of DIV:

AX : _____ , BX : _____
CX : _____ , DX : _____

After Execution of DIV:

AX : _____ , BX : _____
CX : _____ , DX : _____

16-bit:
Before Execution of DIV:

AX : _____ , BX : _____
CX : _____ , DX : _____

After Execution of DIV:

AX : _____ , BX : _____
CX : _____ , DX : _____

EXERCISES

1) Write a program, which will add the contents of two 32 bit numbers stored in DX – AX (DX contains the high order word) and memory location WORD [0202] – WORD [0200].

Program

_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____

2) Write a program which input ten 8-bit numbers as input from user and output their sum on LCD display.

Program	Hex code

3) Write a program, which calculate the factorial of any given number (the number may be used as an immediate operand in the instruction). Use any assembler for this exercise.

Program	Flowchart

Lab Session 12

OBJECT

Studying Transfer of control instructions (Call and Return).

THEORY

Opcode of following CALL instruction: 11101000 disp

CALL label ; transfer the control of program to the location "IP+disp"
(near)

Opcode of following CALL instruction: 11101000 IPnew CSnew

CALL label ; transfer the control of program to the location "label"
(far)

Opcode of following CALL instructions: 11111111 oo010mmm

CALL reg ; transfer the control of program to the location "reg"
(near)
CALL mem ; transfer the control of program to the location of memory "mem"
(near)

Opcode of following CALL instruction: 11111111 oo011mmm

CALL mem ; transfer the control of program to the location of memory "mem"
(far)

Opcode of following RET instruction: 11000011

RET ; Return the control of program to the main routine (to the
(near) instruction next to the associated CALL instruction)

Opcode of following RET instruction: 11000010 data

RET imm ; Return the control of program to the main routine and changes SP
(near) to address "SP+imm"

Opcode of following RET instruction: 11001011

RET ; Return the control of program to the main routine (to the
(far) instruction next to the associated CALL instruction)

Opcode of following RET instruction: 11001010 data

RET imm (far) ; Return the control of program to the main routine and changes SP to address "SP+imm"

ASSEMBLER PROGRAM

```
        MOV   AX , 5AD8
        MOV   CX , 0006
        MOV   WORD[FE], 349A
        MOV   WORD[100], 9CFF
        MOV   WORD[102], A9B6
        MOV    AX , WORD[102]
        CALL   LABEL
        MOV    CX , DX
        JMP    HERE
LABEL:  PUSH    AX
        MOV    AX , WORD[FE]
        INC    WORD[100]
        ADD    AX , WORD[100]
        ROL    AX , CL
        XOR   WORD[102] , AX
        SBB   WORD[FE] , AX
        MOV    DX , AX
        POP    AX
        RET
HERE:   CMP    AX , CX
```

OBSERVATIONS

By using single stepping observe the contents of registers AX, BX, CX, DX and memory location FE, 100 and 102.

Before Execution

AX	:	_____	;	BX	:	_____
CX	:	_____	;	DX	:	_____
SP	:	_____	;	WORD[FE]	:	_____
WORD[100]	:	_____	;	WORD[102]	:	_____

After CALL instruction

AX	:	_____	;	BX	:	_____
CX	:	_____	;	DX	:	_____
SP	:	_____	;	WORD[FE]	:	_____
WORD[100]	:	_____	;	WORD[102]	:	_____

After RET instruction

AX	:	_____	;	BX	:	_____
CX	:	_____	;	DX	:	_____
SP	:	_____	;	WORD[FE]	:	_____
WORD[100]	:	_____	;	WORD[102]	:	_____

After Execution

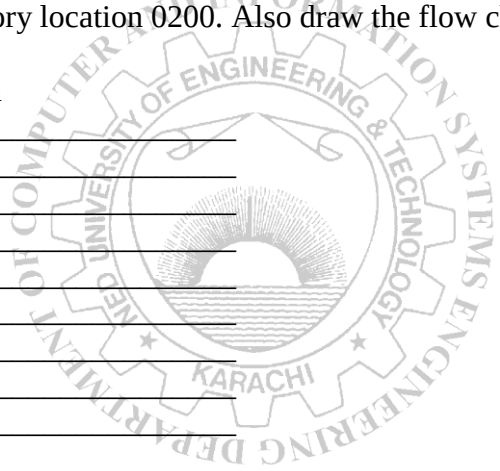
AX	:	_____	;	BX	:	_____
CX	:	_____	;	DX	:	_____
SP	:	_____	;	WORD[FE]	:	_____
WORD[100]	:	_____	;	WORD[102]	:	_____

EXERCISE

Write a program, which takes input from port address 3060h then calls a subroutine having label 'ADDNOS'. ADDNOS subroutine adds the inputted values from the specified input port. The program takes input from port 10 times. Produce a certain delay by using a subroutine between two consecutive inputs from port. Program stores the final result as a word on memory location 0200. Also draw the flow chart of the program.

Program

Flowchart



Lab Session 13

OBJECT

Using ADC/DAC

THEORY

Analog Interface

The MC8088 Analog interface provides one 8-bit ADC (0804) and one 8-bit DAC (0800).
The port address of DAC and ADC is 10C h.

DIGITAL /ANALOG CONVERTER

The digital/analog converter uses an IC34 latch (74374) directly connected to the data bus in order to give the digital information to the conversion device (DAC – 0800). The current of the digital signal to be transmitted is converted into the corresponding voltage signal with the operational IC36 (I – V converter)

ANALOG / DIGITAL CONVERTER

The analog / digital converter uses the ADC0804 for converting and a buffer (74244) for data bus communication of the system.

DAC PROGRAMMING

This program outputs a value from 00h to FFh on the DAC port. Observe the analog output of the program using a multi meter.

Program:

```
START:
          MOV  DX,10C    ; Move address of DAC in DX
          MOV  AL,0  ; reset AL
LOOP:     INT  E     ; Display AL in Hex Format
          PUSH AX    ; save AL on stack
          INT  8     ; Wait for a keyboard hit
          POP  AX
          OUT  DX, AL ; OUT to DAC at 10C h
          INC  AL
          JNZ  LOOP ;if AL is not zero then repeat
          INT  7    ; EXIT
```

ADC PROGRAMMING

Apply analogue voltage at the analogue input of ADC using variable power supply (0-10V DC) and take the digital input from the ADC port at 10C h and display it on the LCD.

Program:

```
START:
          MOV  DX, 10C    ; Move address of ADC in DX
LOOP:     INT  8          ; Wait for a keyboard hit
          IN   AL, DX     ; IN from ADC at 10C h
          INT  E      ; display AL on LCD in HEX format
          JMP  LOOP       ; repeat
          INT  7
```

OBSERVATION

DAC PROGRAMMING:

Observe the multimeter reading for the following values of AL register.

	Value of AL register	**Multimeter Reading (Volts)**
1)	00	----------
2)	15	----------
3)	DE	----------
4)	FC	----------

ADC PROGRAMMING:

Observe the value of AL register for following values of multimeter.

	Multimeter Reading (Volts)	**Value of AL register**
1)	1.5	----------
2)	3.8	----------
3)	6.3	----------
4)	9.8	----------

Lab Session 14

OBJECT

Interfacing Printer with 8088

THEORY

PARALLEL PRINTER INTERFACE

This section of the MC8088 trainer board offers 1 OUTPUT parallel port per printer.

The unit contains all the test points related to the MC8088 signals:

- Address
- Data
- Control signals

It also contains an expansion connector on which the bus signals are reported.
This powers the system with external hardware.

The 25 pin connector complete pin-out is shown below:

STROBE	1	14	AUTOFD
D0	2	15	ERROR
D1	3	16	INIT
D2	4	17	SLCT IN
D3	5	18	GROUND
D4	6	19	GROUND
D5	7	20	GROUND
D6	8	21	GROUND
D7	9	22	GROUND
AK	10	23	GROUND
BUSY	11	24	GROUND
PE	12	25	GROUND
SLCT	13		

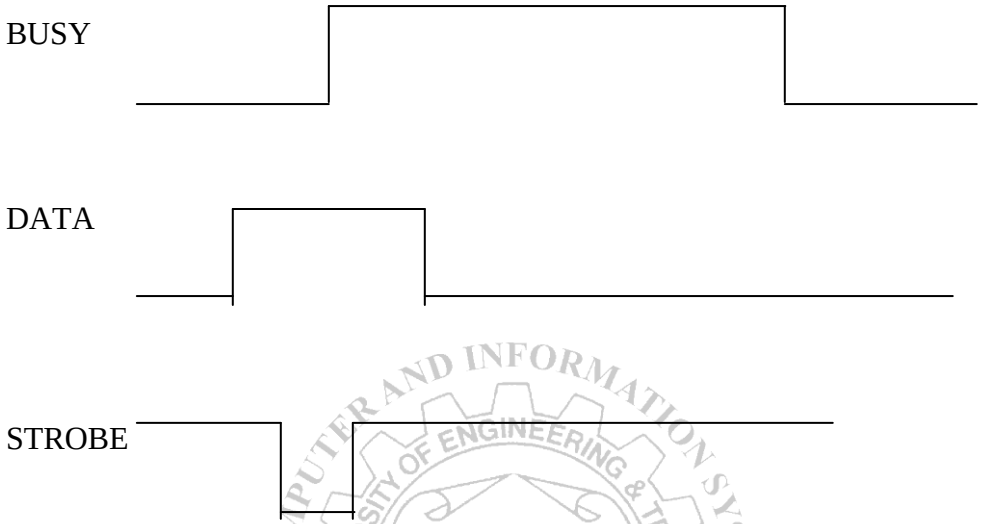
The operation involving the parallel output is controlled by 8 data lines (D0 – D7) and the two lines related to handshaking: BUSY and STROBE controls.

The data are sent to the printer with an IC47 buffer with 1E0H address.

The BUSY line is connected to line 6 of buffer IC7 (line 6 connected to relative bus data line with weight 2^6=64) and is used as input.

The STROBE line is used toward the printer and goes from the system to the printer buffer to inform the printer that the byte to be printed is available on the data lines. It is connected to line 7 of buffer IC3 and to the data bus line with weight 2^7=128.

The timing diagram for the printing operation is:



The BUSY line coming from the printer must be at low logic level for a printing operation; with high logic level the device is in printing state related to a former character.

In a second time data to be printed must be introduced and buffer IC47 must be loaded with byte related to the desired character or control.

At last the line STROBE must be put at low logic level for a moment (STROBE is ON and low) to memorize bytes to be printed in the printer buffer.

ASSEMBLY PROGRAM

PARALLEL PRINTER PROGRAMMING

PRINTER PORT ADDRESSES

DATA	1E0 H
STROBE	180 H bit 7
BUSY	1C0 H bit 6

This program prints typed characters

```
0008 :  0000        INT 8           ;load typed character in AL
```

```
0002        PUSH AX
0003        MOV DX , 1C0      ;read BUSY port
0006        IN AL , DX
0007        TEST AL , 40      ;test bit 6 of byte read
0009        JNZ 3
00OB        POP AX
00OC        MOV DX , 1E0      ;send data to port 1E0h
00OF        OUT DX , AL
0010        MOV DX , 180      ;generate strobe pulse
0013        IN AL , DX
0014        SUB AL, 80
0016        OUT DX , AL
0017        ADD AL , 80
0019        OUT DX , AL
001A        JMP 00
```

Lab Session 15(a)

OBJECT

Learning De-multiplexing of Address/Data bus of 8088 Microprocessor.

THEORY

There is 20-bit address bus and 8-bit data bus present on the chip of 8088 microprocessor. Lower 8 bits of address and data buses are time multiplexed with each other. For any machine cycle address comes out of the microprocessor and after some time the bus is used for data transfer between microprocessor and memory or I/O device. In this way the address is not present there for the whole machine cycle on the bus. For holding the address for the full machine cycle we have to design a circuit.

DESIGN OF CIRCUIT

These components will be required for design of the circuit.

1. 8088 microprocessor
2. 74LS373 latches
3. 74LS244 buffers
4. 74LS245 buffers

STEPS OF DESIGNING (Connection description)

1. Connect the lower 8 bits of the time multiplexed address/data (AD0-AD7) bus to the inputs of latch 74LS373. The only address will be available after passing through the latch.
2. The enable pin of the latch 74LS373 will be connected to the ALE pin of the 8088.
3. The only address will be available after passing through the latch.
4. Connect the lower 8 bits of the time multiplexed address/data (AD0-AD7) bus to the inputs of bi-directional buffer 74LS245.
5. The enable pin of the buffer 74LS245 will be connected to the DEN pin of the 8088.
6. The only data will be pass through the buffer in either direction.
7. The DT/R pin of the microprocessor will control the direction of data flow through the bi-directional buffer.
8. Connect the higher 8 bits of the address bus (A8-A15) to the inputs of buffer 74LS244.
9. Connect the next 4 bits (A16-A19) of address bus to the latch 74LS373.
10. Connect the same pins to the inputs of buffer 74LS244 to get the status signals S3, S4, S5 and S6 from 8088.

Lab Session 15(b)

OBJECT

Creating input / output device select pulses

THEORY

The Microprocessor 8088 has 16-bit register to address I/O devices. Here we have to create device select pulses to select input and output devices. We will use DIP switches as input device and LEDs as output device.

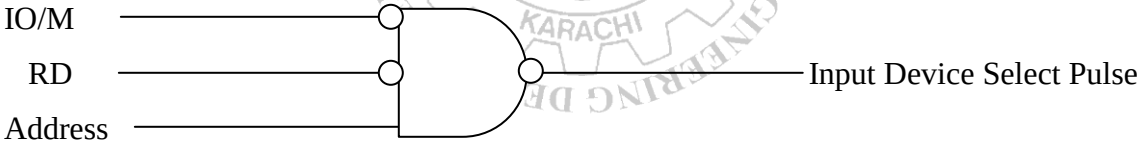
DESIGN OF CIRCUIT

These components will be required for design of the circuit.

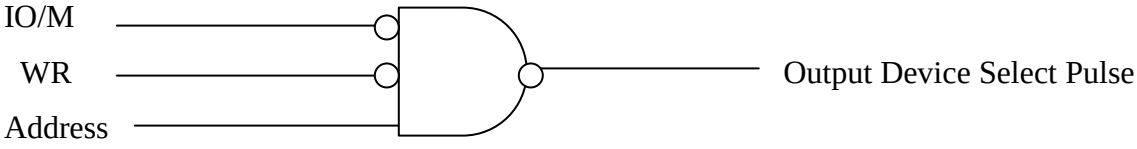
5. DIP switches.
6. LEDs.
7. 74LS08 AND gates.
8. 74LS04 hex inverter.
9. 74LS138 line decoder.

STEPS OF DESIGNING (Connection description)

- For input device selection we have to use IO/M and RD signals and address of the input device to be selected to generate the ***device select pulse***.
- For output device selection we have to use IO/M and WR signals and address of the output device to be selected to generate the ***device select pulse***.
- As IO/M, RD, WR are active low for I/O operations so we will generate the device select pulse in given below manner.



of input device



of output device

- By using these device select pulse we can select / enable the DIP switches or LEDs according to the need.

OR

- By using 74138 line decoder we can generate the device select pulses for I/O devices.

Lab Session 15(c)

OBJECT

Interfacing 8255PPI to the 8088 Microprocessor

THEORY

There are three different ports (Port A, Port B and Port C) are available to interface I/O devices to 8088 microprocessor. There is an internal register, which stores Command Word so we can call it Command register. Command Word defines the modes of working of ports of the device. There are three different types of modes present in 8255 to interface I/O devices to 8088 microprocessor.

Mode 1 : Simple I/O.
Mode 2 : Strobed I/O.
Mode 3 : Handshake I/O.

There are two pins A_0 and A_1 present on the package of 8255PPI to select the ports.

A_1	A_0	Select
0	0	Port A
0	1	Port B
1	0	Port C
1	1	Command Register

First of all the Command Register is selected and the Command Word is stored in the register. After that we can use the ports of 8255PPI according to the function that we have defined in the Command Word.

DESIGN OF CIRCUIT

These components will be required for design of the circuit.

10. 8088 microprocessor.
11. 8255 Programmable Peripheral Interface.
12. DIP switches.
13. LEDs.
14. 74LS373 latches.
15. 74LS244 buffers.
16. 74LS245 buffers.
17. 74LS04 hex inverter.
18. Small capacity RAM IC (e.g. 4016).
19. Small capacity EPROM IC (e.g. 2716).
20. 74LS138 line decoder.

STEPS OF DESIGNING (Connection description)

1. Connect the lower 8 bits of the time multiplexed address/data (AD0-AD7) bus to the inputs of latch 74LS373. The only address will be available after passing through the latch.
2. The enable pin of the latch 74LS373 will be connected to the ALE pin of the 8088.
3. The only address will be available after passing through the latch.
4. Connect the lower 8 bits of the time multiplexed address/data (AD0-AD7) bus to the inputs of bi-directional buffer 74LS245.
5. The enable pin of the buffer 74LS245 will be connected to the DEN pin of the 8088.
6. The only data will be pass through the buffer in either direction.
7. The DT/R pin of the microprocessor will control the direction of data flow through the bi-directional buffer.
8. Connect the higher 8 bits of the address bus (A8-A15) to the inputs of buffer 74LS244.
9. Connect the next 4 bits (A16-A19) of address bus to the latch 74LS373.
10. Connect the same pins to the inputs of buffer 74LS244 to get the status signals S3, S4, S5 and S6 from 8088.
11. Define the addresses for selecting 8255PPI, RAM and EPROM ICs.
12. Connect three address pins to the inputs (A, B and C) of 74138 decoder.
13. Connect the enable pins of the decoder 74138 to appropriate address lines.
14. Connect the data bus of microprocessor to the data bus of 8255PPI.
15. A_0 and A_1 pins of 8255PPI will be connected to A_0 and A_1 pins of 8088 microprocessor respectively.
16. CS (Chip Select) pin of 8255PPI will be connected to one of the outputs of 74138 decoder.
17. RESET of 8255PPI will be connected to RESET of 8088 microprocessor.
18. RD and WR pins of 8255PPI will be connected to the IORC and IOWC pins of 8088 microprocessor respectively.
19. Connect the address and data buses of EPROM and RAM to the address and data buses of 8088 microprocessor.
20. CE or CS pin of EPROM and RAM will be connected to one of the outputs of the 74138 decoder.
21. OE pin of the EPROM and RAM will be connected to the RD pin of the microprocessor.

Lab Session 16(a)

Series-Parallel Logic

OBJECT

To learn how to handle elements when connecting them in series (an AND circuit) or in parallel (an OR circuit).

THEORY

Input I1 is ORed with input I2 and this Logical combination of element is ANDed with input I3. Whether or not input I3 passes power flow to output element O1 depends on whether input 11 or input 12 passes a current flow.

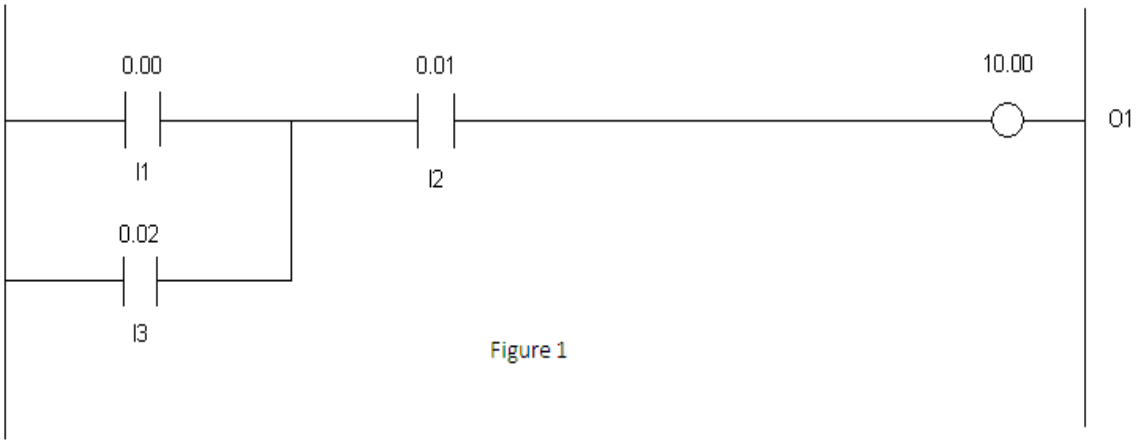


Figure 1

Suppose we try this new combination of AND and OR functions in Figur 2.

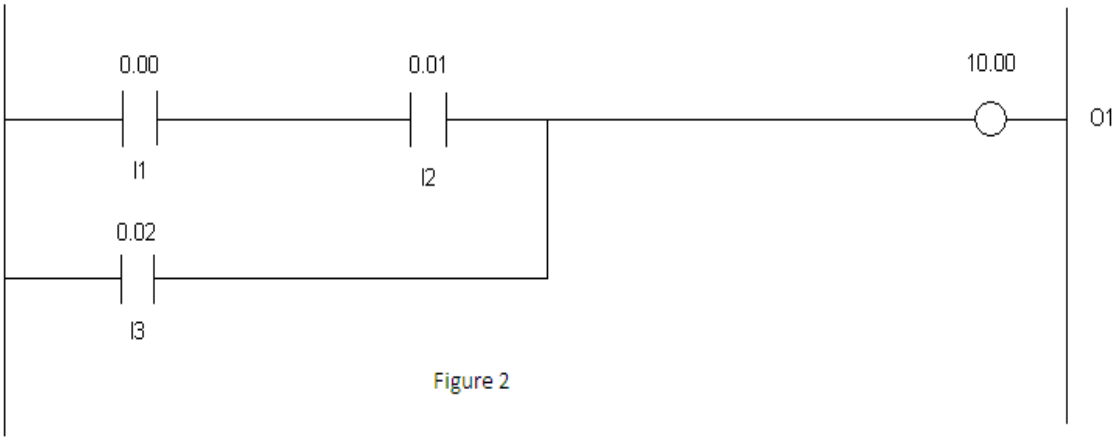


Figure 2

PROCEDURE

Program the PLC with this circuit:

1. To execute CX-programmer, do the following steps:
- Click the START and go to All Programs.
- Select the folder Omron then the folder CX-Programmer.
- Click on CX-Programmer to start.
2. To start a new project, perform the following steps:

Step 1: Create a New Project

Select <File> -- <New> OR click on the New Project icon.
A project window will appear, with a <Change PLC> window.

Step 2: Select your Settings

Assign a name for the PLC at the Device Name (Default – NewPLC1).
Select the appropriate PLC model (CPM2* for this PLC) by clicking on Device Type.
Set the driver to the COM port (in Network Type Settings) connected from PC to PLC.

After the setup has been done, the programming screens will appear. There are 4 different windows:

- Ladder design window
- Project work space
- Output Window Error on Compiling
- Watch window I/O Monitor

Ladder Design Window:

1. Develop the given ladder logic in Figure 1 in this window by following these steps:
- Place a new contact by clicking on it from toolbar and then clicking on ladder design window at the desired location.
- Write 0.00 in Edit contact and press OK then write I1 in Edit comment field and press OK.
- Place another new contact by clicking on it from toolbar and then clicking on ladder design window at the right of I1.
- Write 0.01 in Edit contact and press OK then write I2 in Edit comment field and press OK.
- Place a new coil by clicking on it from toolbar and then clicking on ladder design window at the right of I2.

- Write 10.00 in Edit contact and press OK then write O1 in Edit comment field and press OK.
- Place a vertical wire connection in between I1 and I2 for connecting I3 in parallel of I1.
- Place another new contact by clicking on it from toolbar and then clicking on ladder design window at the bottom of I1.
- Write 0.02 in Edit contact and press OK then write I3 in Edit comment field and press OK.

2. Compile this program by clicking on Program Menu and by selecting Compile option. Another window appears with number on errors and warning message.
3. Now change PLC mode to online from PLC menu and selecting Work online OR by pressing Work On-line button from toolbar. A confirmation dialogue is displayed, select the Yes pushbutton to connect.
4. Select the Download button from the toolbar. The Download Options dialogue is displayed.
5. Set the Programs field and select the OK pushbutton.
6. Deselect the Work Online option. Now you can observe the operation of PLC on the PC monitor.

TEST THE CIRCUIT

Output 10.00 should be energized when:
I-01 and I-02 are pressed.
or
I-03 and I-02 are pressed.

TASK

Develop the logic given in Figure 2 and test its operation:
Output:
_____.

EXERCISE

Program the PLC with the circuit below:

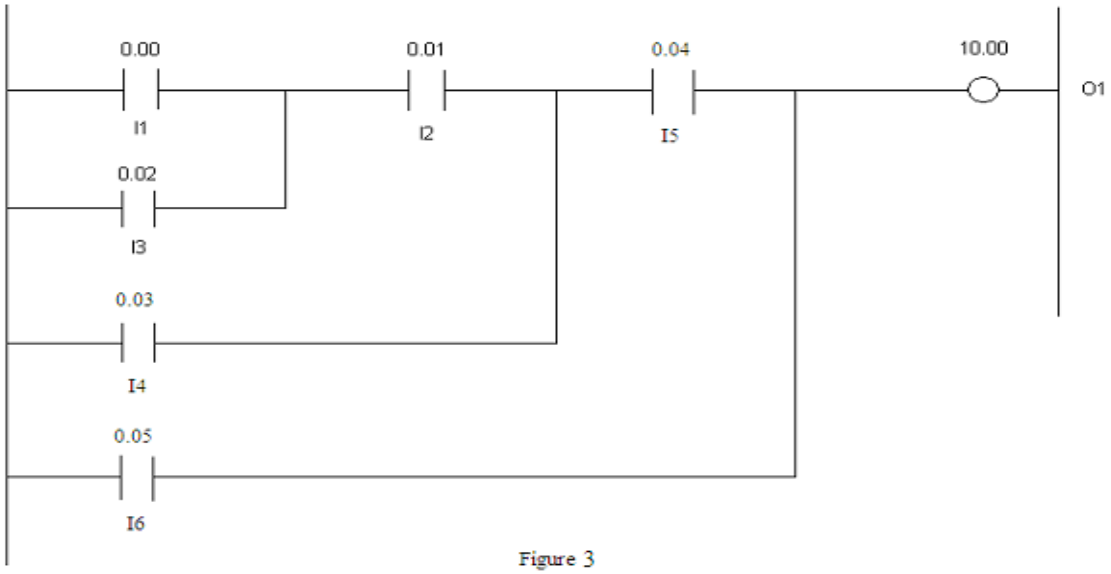


Figure 3

OBSERVATIONS

1. Output O1 when I1 and I4 are pressed ---------. (ON/OFF)
2. Output O1 when I4 and I3 are pressed ---------. (ON/OFF)
3. Output O1 when I5 and I6 are pressed ---------. (ON/OFF)
4. Output O1 when I6 is pressed ---------. (ON/OFF)

Lab Session 16(b)

Latching Circuits

OBJECT

To learn how to program a latching circuit in the PLC.

THEORY

A latching circuit provides a latched (ON) signal from a momentary pulse. When a momentary pulse is transmitted, the circuit turns on and remains on even though the pulse is momentary.

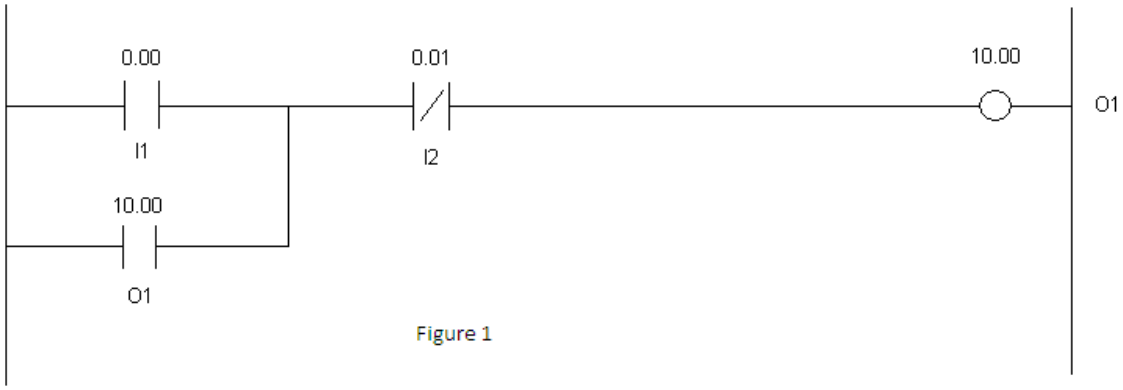


Figure 1

In Figure 1 a momentary push button, PB1, is wired to input I1. When the button is pushed and then released, I1 turns ON, then OFF, providing only momentary energizing for I1. Output O1 receives this momentary pulse, and energizes the contacts (O1) in parallel with the momentary switch. These contacts maintain the connection after the push button is released. Contact will be maintained until I2 is energized, breaking the circuit to relay coil O1. The circuit will also reset to a power off condition if the PLC loses power or the PLC is turned off.

This experiment will also introduce Nicknames and Reference Description.

Input 11 will be named as START.
Input 12 will be named as STOP.

PROCEDURE

Program the PLC with this circuit:

1. To execute CX-programmer, do the following steps:
- Click the START and go to All Programs.
- Select the folder Omron then the folder CX-Programmer.
- Click on CX-Programmer to start.
2. To start a new project, perform the following steps:

Step 1: Create a New Project

Select <File> -- <New> OR click on the New Project icon.
A project window will appear, with a <Change PLC> window.

Step 2: Select your Settings

Assign a name for the PLC at the Device Name (Default – NewPLC1).
Select the appropriate PLC model (CPM2* for this PLC) by clicking on Device Type.
Set the driver to the COM port (in Network Type Settings) connected from PC to PLC.

After the setup has been done, the programming screens will appear. There are 4 different windows:

- Ladder design window
- Project work space
- Output Window Error on Compiling
- Watch window I/O Monitor

Ladder Design Window:

1. Develop the given ladder logic in Figure 1 in this window by following these steps:
- Place a new contact by clicking on it from toolbar and then clicking on ladder design window at the desired location.
- Write 0.00 in Edit contact and press OK then write I1 in Edit comment field and press OK.
- Now Place a new closed contact by clicking on it from toolbar and then clicking on ladder design window at the right of I1.
- Write 0.01 in Edit contact and press OK then write I2 in Edit comment field and press OK.
- Place a new coil by clicking on it from toolbar and then clicking on ladder design window at the right of I2.
- Write 10.00 in Edit contact and press OK then write O1 in Edit comment field and press OK.
- Place a vertical wire connection in between I1 and I2 for connecting I3 in parallel of I1.
- Place another new contact by clicking on it from toolbar and then clicking on ladder design window at the bottom of I1.

- Write 0.02 in Edit contact and press OK then write I3 in Edit comment field and press OK.

2. Compile this program by clicking on Program Menu and by selecting Compile option. Another window appears with number on errors and warning message.
3. Now change PLC mode to online from PLC menu and selecting Work online OR by pressing Work On-line button from toolbar. A confirmation dialogue is displayed; select the Yes pushbutton to connect.
4. Select the Download button from the toolbar. The Download Options dialogue is displayed.
5. Set the Programs field and select the OK pushbutton.
6. Deselect the Work Online option. Now you can observe the operation of PLC on the PC monitor.

TEST THE CIRCUIT

- When SW1 is pressed, enabling I1, the indicator for O1 will illuminate as well. O1 will remain on even after SW1 is no longer pressed.
- When SW2 is pressed, O1 should turn off.
- If the PLC power is turned off, the circuit will RESET when power is turned back on.

EXERCISE

Program the PLC with the circuit below:

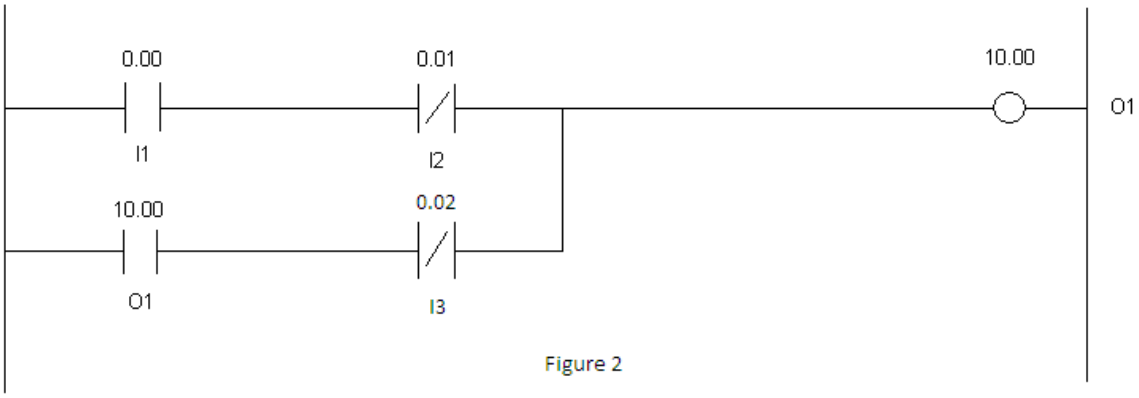


Figure 2

OBSERVATIONS

1. Output O1 when I1 is pressed. --------- (ON/OFF)
2. Output O1 when I3 is pressed and released. --------- (ON/OFF)
3. Output O1 when I2 is pressed. --------- (ON/OFF)
4. Output O1 when I2 and I3 both are pressed --------- (ON/OFF)

Lab Session 16(c)
Timer Circuits

OBJECT

To learn how to program a Timer circuit in the PLC.

THEORY

The internal PLC timer consists of an enabling input, a reset input, and a timer preset value.

The figure shows the basic timer function in a logic circuit.

Figure 1

This circuit delays power to O1 until a preset time has elapsed. The Set Value sets the length of the time delay in multiple of 0.1 seconds. In the above figure, the constant value of 40 will result in a 4 seconds time delay between the time I1 is energized and O1 output.

PROCEDURE

Program the PLC with this circuit:

1. To execute CX-programmer, do the following steps:

- Click the START and go to All Programs.
- Select the folder Omron then the folder CX-Programmer.
- Click on CX-Programmer to start.

2. To start a new project, perform the following steps:

Step 1: Create a New Project

Select <File> -- <New> OR click on the New Project icon.
A project window will appear, with a <Change PLC> window.

Step 2: Select your Settings

Assign a name for the PLC at the Device Name (Default – NewPLC1).
Select the appropriate PLC model (CPM2* for this PLC) by clicking on Device Type.
Set the driver to the COM port (in Network Type Settings) connected from PC to PLC.

After the setup has been done, the programming screens will appear. There are 4 different windows:

- Ladder design window
- Project work space
- Output Window Error on Compiling
- Watch window I/O Monitor

Ladder Design Window:

1. Develop the given ladder logic in Figure 1 in this window by following these steps:

- Place a new contact by clicking on it from toolbar and then clicking on ladder design window at the desired location.
- Write 0.00 in Edit contact and press OK then write I1 in Edit comment field and press OK.
- Now Place a new PLC instruction by clicking on it from toolbar and then clicking on ladder design window at the right of I1.
- Write TIM 001 #40 in Edit Instruction and press OK then write Timer1 in Edit comment field and press OK.
- Place another new contact by clicking on it from toolbar and then clicking on ladder design window just below I1 in the next rung.
- Write TIM001 in Edit contact and press OK then write Timer1 in Edit comment field and press OK.
- Place a new coil by clicking on it from toolbar and then clicking on ladder design window at the right of Timer1.
- Write 10.00 in Edit contact and press OK then write O1 in Edit comment field and press OK.

2. Compile this program by clicking on Program Menu and by selecting Compile option. Another window appears with number on errors and warning message.
3. Now change PLC mode to online from PLC menu and selecting Work online OR by pressing Work On-line button from toolbar. A confirmation dialogue is displayed; select the Yes pushbutton to connect.
4. Select the Download button from the toolbar. The Download Options dialogue is displayed.
5. Set the Programs field and select the OK pushbutton.
6. Deselect the Work Online option. Now you can observe the operation of PLC on the PC monitor.

TEST THE PROGRAM

- Four seconds after I1 is closed O1 will be energized.
- The timer status will be displayed in real time on the program screen in either online or monitor modes. Observe the time value shown on the program screen. If the time is running, a number will increment on the timer. This reflects the timer's internal count. After the set value has been reached, the timer stops to increment.
- Releasing I1 will reset the timer value to zero.

EXERCISE

Q: Design Ladder logic program for Priority determination design (Early Player Buzzer First).

OPERATION:
The game buzzer control requirement:

1. After the Host has finished with question.
2. The 3 players will press the switch in front of them to fight to be first to answer the question.
3. The buzzer will sound for 10 sec after any one of the players has touched the switch.
4. The light indicator in front of each player will light-up and only reset by the Host switch.

I/O ASSIGNMENT:

Input	Device
00000	PB1
00001	PB2
00002	PB3
00003	RST (reset)

Output	Device
01000	Buzzer
01001	Player1 light
01002	Player2 light
01003	Player3 light

LADDER LOGIC:

OBSERVATIONS

_____
_____
_____
_____
_____
_____
_____
_____
_____

Lab Session 16(d)
Counter Circuits

OBJECT

To learn how to program a counter circuit in the PLC.

THEORY

The basic counter within the PLC consists of the input the counter function, the counter reset input, and the counter preset. Each part of the counter performs a very specific function. Each part of the counter performs a very specific function. The counter must first be enabled before it can count events. To enable the counter, the reset element must be open initially. If the reset element is open, then closing the count element causes the accumulated value of a counter to increase by 1.For example, if the accumulated value of a counter was 7, the value would increase to 8 with the closing of the count element.

The accumulated value increases by 1each time the count element goes from open to close. If the count element remain in the close position, the accumulated value increases by only 1.The count element must then be opened and again closed in order to increases the value by a count of 1.When the accumulated value of a counter is equal to a PRESET value, the counter energizes the output relay coil.

Any time the RESET element is closed, the counter is RESET regardless of whether or not the count contact is close. If the counter is RESET, the accumulated value of the counter is RESET, to 0000 and the counter output is de-energized. This causes the output relay coil and its associated contacts to change back to their original status-the normally closed contacts close. These conditions will remain as long as the RESET element is close. When the RESET element is again opened, the counter is ready to begin counting all over again.

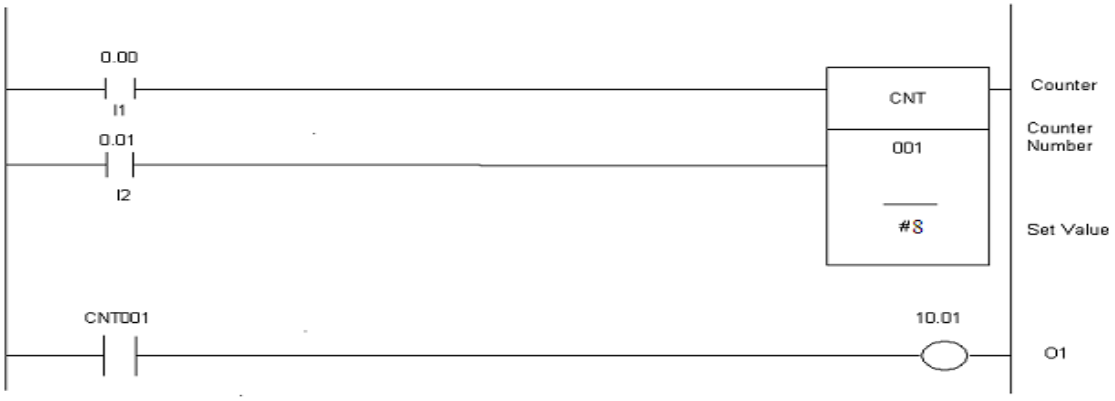


Figure 1

PROCEDURE

Program the PLC with this circuit:

1. To execute CX-programmer, do the following steps:
- Click the START and go to All Programs.
- Select the folder Omron then the folder CX-Programmer.
- Click on CX-Programmer to start.
2. To start a new project, perform the following steps:

Step 1: Create a New Project

Select <File> -- <New> OR click on the New Project icon.
A project window will appear, with a <Change PLC> window.

Step 2: Select your Settings

Assign a name for the PLC at the Device Name (Default – NewPLC1).
Select the appropriate PLC model (CPM2* for this PLC) by clicking on Device Type.
Set the driver to the COM port (in Network Type Settings) connected from PC to PLC.

After the setup has been done, the programming screens will appear. There are 4 different windows:

- Ladder design window
- Project work space
- Output Window Error on Compiling
- Watch window I/O Monitor

Ladder Design Window:

1. Develop the given ladder logic in Figure 1 in this window by following these steps:
- Place a new contact by clicking on it from toolbar and then clicking on ladder design window at the desired location.
- Write 0.00 in Edit contact and press OK then write I1 in Edit comment field and press OK.
- Now Place a new PLC instruction by clicking on it from toolbar and then clicking on ladder design window at the right of I1.
- Write CNT 010 #8 in Edit Instruction and press OK then write Counter1 in Edit comment field and press OK.
- Place another new contact by clicking on it from toolbar and then clicking on ladder design window just below I1.
- Write 0.01 in Edit contact and press OK then write I2 in Edit comment field and press OK.
- Place third new contact by clicking on it from toolbar and then clicking on ladder design window just below I2 in the next rung.

- Write CNT010 in Edit contact and press OK then write Counter1 in Edit comment field and press OK.
- Place a new coil by clicking on it from toolbar and then clicking on ladder design window at the right of Counter1.
- Write 10.00 in Edit contact and press OK then write O1 in Edit comment field and press OK.

2. Compile this program by clicking on Program Menu and by selecting Compile option. Another window appears with number on errors and warning message.
3. Now change PLC mode to online from PLC menu and selecting Work online OR by pressing Work On-line button from toolbar. A confirmation dialogue is displayed; select the Yes pushbutton to connect.
4. Select the Download button from the toolbar. The Download Options dialogue is displayed.
5. Set the Programs field and select the OK pushbutton.
6. Deselect the Work Online option. Now you can observe the operation of PLC on the PC monitor.

TEST THE PROGRAM

- Press I1 eight times. The accumulated count will appear on the counter. At the tenth switch closure O1 will energize.
- Press I2 to reset the counter and repeat the experiment.

EXERCISE

Q: Design Ladder Logic Program for Packaging Line Control:

OPERATION:
When PB1 (START Push Button) is pressed, the box conveyor moves. Upon detection of box present, the box conveyor stops and the Apple conveyor starts. Part sensor will count for 10 apples. Apple conveyor stops and box conveyor starts again. Counter will be reset and operation repeats until PB2 (STOP Push Button) is pressed.

I/O ASSIGNMENT:

Input	Devices
00000	START Push Button (PB1
00001	STOP Push Button (PB2)
00002	Part Present (SE1)
00003	Box Present (SE2)

Output	Devices
01000	Apple Conveyor
01001	Box Conveyor

LADDER LOGIC:

OBSERVATIONS

CIRCUIT DIAGRAM

(Lab session 15)