Microprocessor Based System Design

(CE-207)

Computer Engineering Department
Sir Syed University of Engineering & Technology
University Road, Karachi 75300
http://www.ssuet.edu.pk
## Content

<table>
<thead>
<tr>
<th>Lab #</th>
<th>Topic</th>
<th>Page#</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Explore debugger using different commands</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>a. To understand the basic concept and functionality of Assembly Language&lt;br&gt;b. To understand the concept of new line using carriage return, line feed and macro</td>
<td>11, 19</td>
</tr>
<tr>
<td>3</td>
<td>To understand the concept of displaying string in assembly language</td>
<td>21</td>
</tr>
<tr>
<td>4</td>
<td>To understand the concepts of loop in assembly language</td>
<td>24</td>
</tr>
<tr>
<td>5</td>
<td>Introduction to MDA – 8086 Training Kit</td>
<td>26</td>
</tr>
<tr>
<td>6</td>
<td>To understand the basic operations of MDA 80x86 trainer kit</td>
<td>28</td>
</tr>
<tr>
<td>7</td>
<td>To understand the different commands of MDA 80x86 trainer Kit</td>
<td>32</td>
</tr>
<tr>
<td>8</td>
<td>Explore kit mode functionality</td>
<td>36</td>
</tr>
<tr>
<td>9</td>
<td>Write a program to display the digits in decimal, from 0-7 into 7-segment</td>
<td>39</td>
</tr>
<tr>
<td>10</td>
<td>To initialize dot matrix display</td>
<td>42</td>
</tr>
<tr>
<td>11</td>
<td>Exploring MDE 8051 Microcontroller Trainer Boards</td>
<td>45</td>
</tr>
<tr>
<td>12</td>
<td>Programming 8051 I/O Ports as Input Ports</td>
<td>51</td>
</tr>
<tr>
<td>13</td>
<td>Programming 8051 I/O Ports as Input Ports and Output Ports</td>
<td>54</td>
</tr>
</tbody>
</table>
LAB#1

Objective:

Explore debugger using different commands.

Theory:

Programming Languages

A programming language is an artificial language that can be used to control the behavior of a machine, particularly a computer. Programming languages, like human languages, have syntactic and semantic rules to define meaning.

Types of Programming Languages

Programming languages can be classified into three basic categories on the basis of understanding level of users as well as the machine to which instructions has been given:

1. **High Level Languages**
   A programming language that enables a programmer to write programs that are more or less independent of a particular type of computer and are designed to give a better program efficiency. Such languages are considered high-level because they are closer to human languages.

2. **Low Level Languages**
   These are designed to have both: a relatively good programming efficiency and relatively good machine efficiency.

3. **Machine Language**
   Machine language is at the lowest level, because it is the actual binary code of 1s and 0s that the computer understands. These are designed to give a better machine efficiency.

Registers Classification

The registers inside the microprocessor are classified according to the function they perform.

In general, they are classified as
1. Data registers
2. Address registers
3. Segment register
4. Offset registers
5. Status register
Some General Purpose Registers:

**AX (Accumulator Register)**
- It is the preferred register to use in the arithmetic, logic and data transfer instructions because its use generates the shortest machine code.
- In multiplication and division operations, one of the numbers involved must be in AX or AL.
- Input and output operation also requires the use of AX and AL.

**BX (Base Register)**
- It is used to store the data also it serves as an address register.

**CX (Count Register)**
- Program loop instructions are facilitated by the use of CX register, serves as a loop counter.
- Also used as a counter in the string operations.
- CL is used as count in instructions that shift and rotate bits.

**DX (Data Register)**
- It is used in multiplication and division operations.
- It is used in IO operation like DL in character output and DX in string output functions.

**Register Size:**
- We have three different sizes of registers:
  - 8-bit register: AH, AL, BH, BL, CH, CL, DH, DL
  - 16-bit registers: AX, BX, CX, DX, SP, BP, SI, DI, SS, DS, CS, ES, FS, GS, IP, FLAGS
  - 32-bit registers: EAX, EXB, ECX, EDX, ESI, EDI, ESP, EBP, EIP, and EFLAGS.

**Basic MOV Instruction**
- The basic MOV instruction is used to transfer data between registers, between and memory locations, or to have a number directly to a register or memory location.

**Syntax:** MOV Destination, Source
Examples:

- MOV AH, BL ; 8-bits register to register
- MOV BX, AX ; 16-bits register to register
- MOV byte1, BL ; 8-bit register to memory
- MOV AX, word1 ;16-bit memory to register

Some Arithmetic Instructions

**ADD**: Add the contents of source operand1 to source operand 2 and result store in the source operand1.

**Syntax**: `ADD Source operand1, Source operand2`

**Examples**: `ADD AL, BL`

**SUB**: Subtract the contents of source operand1 to source operand 2 and result store in the source operand1.

**Syntax**: `SUB Source operand1, Source operand2`

**Examples**: `SUB AL, BL`

**Debug Program**

To create a program in assembler two options exist, the first one is to use the assembler program such as TASM or Turbo Assembler from Borland, and the second one is to use the debugger - on this first section we will use this last one since it is found in any PC with the MS-DOS, which makes it available to any user who has access to a machine with these characteristics.

Debug can only create files with a .COM extension, and because of the characteristics of these kinds of programs they cannot be larger than 64 kb, and they also must start with displacement, offset, or 0100H memory direction inside the specific segment.

Debug provides a set of commands that lets you perform a number of useful operations:
A Assemble symbolic instructions into machine code

D Display the contents of an area of memory

E Enter data into memory, beginning at a specific location

G Run the executable program in memory

N Name a program

P Proceed, or execute a set of related instructions

Q Quit the debug program

R Display the contents of one or more registers

T Trace the contents of one instruction

U Unassembled machine code into symbolic code

W Write a program onto disk

- It is possible to visualize the values of the internal registers of the CPU using the Debug program. To begin working with Debug, type the following prompt in your computer:

C:/>Debug [Enter]

- On the next line a dash will appear, this is the indicator of Debug, at this moment the instructions of Debug can be introduced using the following command:

-r[Enter]

AX=0000 BX=0000 CX=0000 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000

DS=0D62 ES=0D62 SS=0D62 CS=0D62 IP=0100 NV EI PL NZ NA PO NC

0D62:0100 2E CS:

0D62:0101 803ED3DF00 CMP BYTE PTR [DFD3],00 CS:DFD3=03

- All the contents of the internal registers of the CPU are displayed; an alternative of viewing them is to use the "r" command using as a parameter the name of the register whose value wants to be seen.

- For example:

-rbx
BX 0000
 :

**Assembler Structure**

In assembly language code lines have two parts, the first one is the name of the instruction which is to be executed, and the second one are the parameters of the command. For example:

**add ah,bh**

Here "add" is the command to be executed; in this case an addition, and "ah" as well as "bh" are the parameters. **For example:**

**mov al, 25**

In the above example, we are using the instruction mov, it means move the value 25 to al register. The name of the instructions in this language is made of two, three or four letters. These instructions are also called mnemonic names or operation codes, since they represent a function the processor will perform.

**Direct Addressing**

Sometimes instructions are used as follows:

**add al,[170]**

The brackets in the second parameter indicate to us that we are going to work with the content of the memory cell number 170 and not with the 170 value, this is known as direct addressing.

**Creating basic assembler program:**

a 100[enter]

mov ax,0002[enter]

mov bx,0004[enter]

add ax,bx[enter]

nop[enter][enter]

C:\>debug

-a 100

0D62:0100 mov ax,0002
Microprocessor Based System Design

0D62:0103 mov bx,0004
0D62:0106 add ax,bx
0D62:0108 nop
0D62:0109

- Type the command "t" (trace), to execute each instruction of this program, example

-t
AX=0002 BX=0000 CX=0000 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000
DS=0D62 ES=0D62 SS=0D62 CS=0D62 IP=0103 NV EI PL NZ NA PO NC
0D62:0103 BB0400 MOV BX,0004

- You see that the value 2 move to AX registers. Type the command "t" (trace), again, and you see the second instruction is executed.

-t
AX=0002 BX=0004 CX=0000 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000
DS=0D62 ES=0D62 SS=0D62 CS=0D62 IP=0106 NV EI PL NZ NA PO NC
0D62:0106 01D8 ADD AX,BX

- Type the command "t" (trace) to see the instruction add is executed, you will see the follow lines:

-t
AX=0006 BX=0004 CX=0000 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000
DS=0D62 ES=0D62 SS=0D62 CS=0D62 IP=0108 NV EI PL NZ NA PE NC
0D62:0108 90 NOP
Lab Objective:

Task#1: Display the contents of the defined memory locations 120, 133, 122 using D command.

Task #2: Edit the contents of the above memory locations 120, 133, 122 by 02, 04, 03 respectively using E command.

Task #3: Then again display the contents of the memory locations which we edit in the Task# 2.

Task #4: Add the contents of the above defined memory location using mov instruction.

Task #5: Subtract the content of 120 locations by 133 and then store the result in the 120 location and add the new 120 location contents with the content of 122 location.

Task #6: Perform the following debug activities

ACT 1.1 : Use debug command U100 to un-assemble the instructions in ACT 1.1. What is the machine code corresponding to each assembly code instruction.

<table>
<thead>
<tr>
<th>Assembly Code</th>
<th>Machine Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>mov ax,2864h</td>
<td></td>
</tr>
<tr>
<td>add ax,3749h</td>
<td></td>
</tr>
<tr>
<td>mov bx,ax</td>
<td></td>
</tr>
<tr>
<td>sub bx,2805</td>
<td></td>
</tr>
<tr>
<td>nop</td>
<td></td>
</tr>
</tbody>
</table>

ACT 1.2 : How many bytes does it need to represent each instruction in binary.

<table>
<thead>
<tr>
<th>Assembly Code</th>
<th>No. Of Byte</th>
</tr>
</thead>
<tbody>
<tr>
<td>mov ax,2864h</td>
<td></td>
</tr>
<tr>
<td>add ax,3749h</td>
<td></td>
</tr>
<tr>
<td>mov bx,ax</td>
<td></td>
</tr>
<tr>
<td>sub bx,2805</td>
<td></td>
</tr>
<tr>
<td>nop</td>
<td></td>
</tr>
</tbody>
</table>

ACT 1.3 : What are the contents of CS, IP, AX & BX ?. Use debug command R to display these information.

<table>
<thead>
<tr>
<th>Register</th>
<th>Content</th>
</tr>
</thead>
<tbody>
<tr>
<td>CS</td>
<td></td>
</tr>
<tr>
<td>IP</td>
<td></td>
</tr>
<tr>
<td>AX</td>
<td></td>
</tr>
<tr>
<td>BX</td>
<td></td>
</tr>
</tbody>
</table>
ACT 1.4: Predict the contents of the following registers after execution of each instruction
CS, IP, AX, BX

<table>
<thead>
<tr>
<th>Register</th>
<th>Mov ax,2864</th>
<th>Add ax,3794</th>
<th>Mov bx,ax</th>
</tr>
</thead>
<tbody>
<tr>
<td>CS</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IP</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>AX</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BX</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
LAB#2(a)

Objective
To understand the basic concept and functionality of Assembly Language

Theory

Assembly Language
Assembly language is a machine specific programming language with a one-to-one correspondence between its statements and the computer’s native machine language. There are many different types of assembly language, each specific to a processor or processor family. IBM-PC assembly language refers to instruction recognized by a number of different microprocessors in the Intel family: 8086, 8088, 80186, 80286, 80386, 80486, and Pentium.

Uses
- Assembly language is used most often when either communicating with the operating system or directly accessing computer hardware.
- Secondly, assembly language is used to optimize certain critical areas of application programs to speed up their runtime execution.

Assembler
An assembler is a program that converts source code programs from the assembly language into machine language. The assembler can optionally generate a source-listing file with line numbers, memory addresses, source code statements and a cross-reference listing of symbols and variables used in a program.

The most popular assemblers for the Intel family are MASM (Microsoft Assembler), TASM (Turbo Assembler).

Linker
A companion program that combines individual files created by an assembler into a single executable file

Debugger
A program that provides a way for a programmer to trace the execution of a program, and examine the contents of memory

Microprocessor Architecture
As assembly language is microprocessor specific, so before writing assembly language programs it is required to know the architecture of that microprocessor.
8086 Microprocessor

We will be studying the assembly language for the microprocessor 8086.
- 8086 microprocessor is a 16-bit microprocessor (a 16-bit microprocessor can operate on the 16-bits of data at a time) having 16-bit registers.
- 16-bit data bus and 20-bit address bus.
- It can access 1 MB of total memory.

To write the assembly language programs for 8086 microprocessor, we are required to know the internal organization at least programming model

Programming Model

The programming model of the 8086 through Pentium II is considered to be program visible because its registers are used during application programming and are specified by the instruction. Other registers are considered to be program invisible because they are not addressed directly during application programming, but may be used indirectly during system programming.

Registers Classifications

The registers inside the microprocessor are classified according to the function they perform. In general, they are classified as:
- Data registers
- Address registers
- Segment register
- Offset registers
- Status register

Some General Purpose Registers

General purpose register we already defined in the Lab #1.

Segment and Offset Register:

- Address registers store addresses of instructions and data in memory.
- These values are used by the microprocessor to access memory locations.
- Every memory location is actually identified by an address of 20-bit.
- We are have the registers of 16-bits son that over 20-bit address (Physical Address) is store into two parts in two 16-bit registers called segment number and offset.
- A memory segment is a block of $2^{16}$ (64 K) consecutive memory bytes.
- A segment number, called segment address (starting with 0) identifies each segment.
- A segment number is 16 bits so the highest segment number is FFFFh.
- Within segment, giving an offset number called the offset identifies a memory location.
- This is the number of bytes from the beginning of the segment.
- With a 64KB segment, the offset can be given as a 16-bit number.
- The first byte in a segment has offset 0.
- The last offset in a segment is FFFFh.
**Program Segment**
- A machine language program consists of instructions (code) and data.
- A data structure called Stack is used by the microprocessor to implement procedure calls.
- The program’s code, data and stack are loaded into different memory segments, called code segment, data segment and the stack segment.

**Segment Registers**
To keep track of the various program segments, the 8086 microprocessor is equipped with four segment registers to hold the segment numbers.
- The CS, DS and SS registers contain the code, data and stack segment numbers respectively.
- If a program needs to access a second data segment, it can use the ES (Extra Segment) register.

**Pointer and Segment Register**
- The registers SP, BP, SI and DI normally points to (contain the offset address of) memory location.
- These registers can be used in arithmetic and other operations.

**SP (Stack Pointer)**
- The SP register contains the offset of the top of stack.
- The SP and SS registers combine to form the complete address of the top of the stack.

**BP (base Pointer)**
- The BP register contains an assumed offset from the SS register, as does the stack pointer.
- The BP register is often used by a subroutine to locate variables that were passed on the stack by a calling program.
- The BP register can be used for other segment unlike SP register.

**SI (Source Index)**
- The SI register is used to point to memory locations in the data segment addressed by DS.
- This register takes its name from the string movement instructions, in which the source string is pointed to by the SI register.

**DI (Destination Index)**
- The DI register performs the same operation as the SI register.
- The DI register acts as the destination for string movement instructions.

**Control Register**

**IP (Instruction Pointer)**
- The IP register contains the offset of the next instruction to be executed within the current code segment.
- The IP and CS registers combine to form the complete address of the next instruction.
Flags (Flag Register)

- This is a special register with individual bit positions assigned to show the status of the CPU or the results of arithmetic operations.
- Each relevant bit is given a name and others are undefined.

Assembly Program Syntax

- Assembly language program consists of statements.
- A statement is either an instruction to be executed when the program runs or a directive for the assembler.
- A program normally consists of three parts or segments.

Data Segment

- Variables are declared in the data segment.
- Each variable is assigned space in memory and may be initialized.

Exp:

- A DW 3501H
  It sets memory for a variable called A, and initialize it to 3501H.
- DW - Define word (16 bits = 2 memory locations)
  - A DW (?) ; un-initialized variable

Code Segment

- Program instructions are placed in the code segment. Instructions are actually organized into units called procedures. Every procedure starts with a line.

Exp:

- Main Proc;
  Main is the name of procedure and PROC is the directive identify the start of the procedure
- Main Endp;
  Main is again the name of the procedure and Endp is the directive ; identifies the end of the procedure

Stack Segment

- The stack segment is used for temporary storage of addresses and data. If no stack segment is declared, an error message is generated, so there must be a stack segment even if the program doesn’t utilize the stack.
- These segments begin with the directives .stack, .code, and .data
Program Syntax
TITLE first program syntax
.Model Small ;Specifies the memory model used
.Stack 100H ;allocate 100H memory locations for stack
.Data ;start of the data segment
; Data definitions here
A DB ?
........
.Code ;start of the code segment
Main Proc ;start of the first procedure
; instructions here
......
Main Endp ;end of the first procedure
; Other procedures here
End Main ;end of the complete assembly program

Basic Directives
Following are the description of commonly used directives;

**Model:** It is used to specify the memory model used for program to identify the size of code and data segments.

**Stack:** Defines the size of stack used in program

**Data:** Defines the data segments for data used in the program. Mark the beginning of the data segment

**Code:** Identifies the code segment which contains all the statements. Also .code marks the beginning of the code segment.

**Proc:** Beginning of the procedure

**EndP:** End of the procedure

**End:** End of assembly language program

Basic MOV Instruction
We already defined in the Lab#1

**Restrictions:**

- Move between memory to memory is not allowed.
- A number directly inside a segment register is not allowed.
- Segment to segment registers, move is not allowed.
Input / Output Operations

To display a character is an output operation.

We actually want to display single character from the microprocessor to the screen.

We don’t have any instruction that perform this operation like printf in C language.

We actually use the service routines, which are already designed and assembled programs to perform the IO operations.

There are two categories of service routines

- Basic Input Output System (BIOS) routines
- DOS routines

The BIOS routines are stored in ROM and interact directly with the I/O ports. We carry out basic screen operations.

The DOS routines are the part of the operating system running into the system.

DOS routines can carry out more complex tasks, e.g. printing a character string.

DOS routines actually use the BIOS routines to perform direct I/O operations.

INT Instruction

INT instruction is used to invoke a DOS or BIOS routine.

Syntax INT 10H (BIOS routine call)

INT 21H (DOS routine call)

Function Number & AH Register

A routine is actually a collection of different functions.

A function is a set of basic instructions, used to actually perform the operation.

Q: When we call a routine, it means we are invoking a large number of DOS functions but which function?

A: A particular function is requested by placing a function number in the AH register and invoking INT 21H.
INT 21h:

This can be used to invoke large number of DOS functions. A particular function is requested by placing a function number in the AH register and invoking INT 21h.

For E.g. following functions can be used in our program:

<table>
<thead>
<tr>
<th>Function No.</th>
<th>Routine</th>
</tr>
</thead>
<tbody>
<tr>
<td>02h</td>
<td>Single character output</td>
</tr>
</tbody>
</table>

Single Character Output

MOV AH,02H ; display character function
MOV DL,‘?’ ; character is ‘?’
INT 21h ; character will be displayed

Sample Program

Object:

Title a program that displays single character on screen.

Source Code

.model small ;specify memory model
.stack 100h ; specify size of stack
.code ; start of the code segment
main proc ; start of the first procedure
mov ah,02h ; display a single character
mov dl,’A’ ; transfer A to register dl
int 21h ; DOS routine interrupt
mov ah,4ch ; exit DOS function
int 21h
main endp
end main
Steps to Follow:

ASSEMBLING:

1-Go to preferences → Assembly → write your file name with .asm extension.

Output:

2-Go to preferences → DOS tool → Write your file name.
LAB#2(b)

Objective
To understand the concept of new line using carriage return, line feed and macros.

Theory

INT 21h:
This can be used to invoke large number of DOS functions. A particular function is requested by placing a function number in the AH register and invoking INT 21h.

For E.g. following functions can be used in our program:

<table>
<thead>
<tr>
<th>Function No.</th>
<th>Routine</th>
</tr>
</thead>
<tbody>
<tr>
<td>1h</td>
<td>Single key input</td>
</tr>
</tbody>
</table>

Examples:

Single Key Input

MOV AH,01H ; input key function
INT 21h

Macros

A macro is a block of code that has been given a name and when the compiler encounters such name, it expands the macro, copies the block of code at the place of invocation into the program. The text may contain instructions or references to other macros.

The basic advantage of using a macro is to create new instructions. These macros can be a part of the program or they can be written in a separate file and that file can be included in the program. There is a list of useful macros which are helpful to be kept in the macro library of the assembly language compiler and these can be used in other user programs.

These can also be written in the lines of code of the program without any name given to them.
Example:

Carriage Return

This macro can be used for the purpose of carriage return.

CAR_RTN  MACRO
    MOV AH, 4CH
    INT 21H
    ENDM

Once declared and defined, this macro can be invoked anywhere in the program by the name CAR_RTN. The name of the macro can be any string as defined by the user.

Line Feed

This macro can be used to feed a new line in the output and is very useful as a new line is needed many times during a formatted output.

NEW_LINE  MACRO
    MOV AH, 02H
    MOV DL, 0DH
    INT 21H
    MOV DL, 0AH
    INT 21H
    ENDM

Once declared and defined, this macro can be invoked anywhere in the program to insert a new line. The name of the macro can be any string as defined by the user.

Lab Objective

Task#1: Write a program that takes a single character input and displays it in a new line.
LAB#3

Objective
To understand the concept of displaying string in assembly language

Theory:

<table>
<thead>
<tr>
<th>Function No.</th>
<th>Routine</th>
</tr>
</thead>
<tbody>
<tr>
<td>9h</td>
<td>Character String Output</td>
</tr>
</tbody>
</table>

Example:

Character String Output

```assembly
MOV AH, 09H                  ; input key function
MOV DX,OFFSET MSG           ; moves the offset of the msg to the register
INT 21h
```

String:

A string is a list of characters treated as a unit. In programming languages we denote a string constant by using quotation marks, e.g. “Enter first number”.

In 8086 assembly language, single or double quotes may be used.

Defining String Variables

The following 3 definitions are equivalent ways of defining a string ‘abc’:

version1 db “abc” ; string constant
version2 db ‘a’, ‘b’, ‘c’ ; character constants
version3 db 97, 98, 99 ; ASCII codes

The first version uses the method of high level languages and simply encloses the string in quotes. This is the preferred method.

The second version defines a string by specifying a list of the character constants that make up the string.

We may also combine the above methods to define a string as in the following example:message db “Hello world”, 13, 10, ‘$’
String Output

MS-DOS provides subprogram number 9h to display strings which are terminated by the ‘$’ character. In order to use it we must:

1. Ensure the string is terminated with the ‘$’ character.
2. Specify the string to be displayed by storing its address in the dx register.
3. Specify the string output subprogram by storing 9h in ah.
4. Use int 21h to call MS-DOS to execute subprogram 9h.
5. Keyword : MSG

The message to be displayed in the output can also be declared in the data segment using the keyword MSG, moving the string in the register DB and can be used afterwards.

MSG DB ‘HELLO!$’

The “$” marks the end of the string and is not displayed. If the string contains the ASCII code of a control character, the control function is performed.

Sample Program

SOURCE CODE:

Object: Write a program to display a string.

.model small ; specify memory model
.stack 100h ; specify size of stack
.data ; start of the data segment

msg1 db 'Type Character : $'

.code ; start of the code segment

main proc ; start of the first program

mov ax,@data
mov ds,Ax ; mov content ax to ds
mov dx,offset msg1
mov Ah,09h ; reads from offset and display on output
int 21h ; interrupt call
mov Ah,01h ; input a single character
int 21h ; interrupt call
mov bl,Al ; mov content al to bl
mov dl,0Dh ; function of carriage return
mov Ah,02h ; display a character
int 21h ; interrupt call
mov dl,0Ah ; function of line feed
mov Al,02h ; display a character
int 21h ; interrupt call
mov Ah,4ch ; exit dos function
int 21h ; interrupt call
end main

Lab Objectives

**TASK#1:** Write a program to display your bio data using carriage return and line feed macro

**TASK#1:** Write a program to make your 3rd semester marksheet.

---

**LAB#4**

**Objective:**

To understand the concepts of loop in assembly language
Theory

Loop

A loop is a sequence of instructions that is repeated. The number of times to repeat may be known in advance, or it may depend on conditions i.e. it’s a count controlled loop.

FOR Loop

This is a loop structure in which the loop statements are repeated a known number of times.

Keyword: LOOP

A FOR loop is implemented using the LOOP instruction. The counter for the loop is the CX register, which is initialized to loop_count, which is the number of times the loop is executed. Execution of the LOOP instruction causes CX to be decremented automatically. If CX becomes 0, the next instruction after loop is done.

Sample Program:

SOURCE CODE:
Object: Title a program that prints a character 100 times.

.model small
.stack 100h
.code
    main proc
        mov ah, 02h ;display a character
        mov cx, 100 ;number of times loop will execute
        mov dl, ‘*’ ;ASCII code of character 0
        print:
            int 21h ;loop starts from here
        loop print
            ;executes the FOR loop
            mov ah, 4ch ;terminates the current process
        int 21h
        main endp
        end main

Figure. FOR LOOP
Lab Objectives:

Task #01: Write a program to print ‘*’ 100 times using linefeed and carriage return.

Task #02: Write a program to print ASCII characters.

Task #03: Write a program to print your name 10 times using linefeed and carriage return.

LAB#5

Objective

Introduction to MDA – 8086 Training Kit

Theory
MDA-8086 has high performance 64K-byte monitor program. It is designed for easy function. After power is on, the monitor begins to work. In addition to all the key function the monitor has a memory checking routine.

The following is a simple description of the key functions.

<table>
<thead>
<tr>
<th>FUNCTION KEY</th>
<th>DATA KEY</th>
</tr>
</thead>
<tbody>
<tr>
<td>GO</td>
<td>MON</td>
</tr>
<tr>
<td>STP</td>
<td>REG</td>
</tr>
<tr>
<td>+</td>
<td>8</td>
</tr>
<tr>
<td>-</td>
<td>DA</td>
</tr>
<tr>
<td>:</td>
<td>AD</td>
</tr>
<tr>
<td></td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>3</td>
</tr>
</tbody>
</table>

### Function of Keys

- **RES** - system reset
- **STP** - execute user’s program, a single step
- **AD** - set memory address
- **GO** - go to user’s program or execute monitor functions
- **DA** - Update segment & Offset, and input data to memory
- **MON** - Immediately break user’s program and Non makable interrupt.
- **REG** - Register Display.
- **Offset set.**
- **Segment & Offset + 1 increment.**
- **Register display increment.**
- **Segment & Offset - 1 increment.**
- **Register display decrement.**

### Technical Specification:

- **CPU** 8086
- **Main RAM** 64 KB (62256 x 2)
- **Monitor ROM** 64 KB (27C256 x 2)
### Microprocessor Based System Design

<table>
<thead>
<tr>
<th>Component</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>Display Unit</td>
<td>LCD (16 x 2 Line)</td>
</tr>
<tr>
<td>I/O Port</td>
<td>8255A</td>
</tr>
<tr>
<td>Serial Port</td>
<td>RS-232C (8251A x 1)</td>
</tr>
<tr>
<td>System Clock</td>
<td>4.9152 MHz</td>
</tr>
<tr>
<td>Clock Generator</td>
<td>8284</td>
</tr>
<tr>
<td>Interrupt Controller</td>
<td>8259</td>
</tr>
<tr>
<td>Level Meter</td>
<td>10 step</td>
</tr>
<tr>
<td>DOT Matrix</td>
<td>8 x 8 (3 Color)</td>
</tr>
<tr>
<td>Operation System Software</td>
<td>8086 Assembler</td>
</tr>
<tr>
<td>Keyboard</td>
<td>16 Key of data, 10 Key of function</td>
</tr>
<tr>
<td>System bus indicator</td>
<td>LED (3Ø) x 12</td>
</tr>
<tr>
<td>Expansion Connector</td>
<td>System bus 62pin x 1</td>
</tr>
<tr>
<td>Step Motor Interface</td>
<td>External Interface 20pin x 1</td>
</tr>
<tr>
<td>A/D , D/A Converter</td>
<td>Driver T.R x 4</td>
</tr>
<tr>
<td>ADC : ADC 0804</td>
<td></td>
</tr>
<tr>
<td>DAC : DAC 0800</td>
<td></td>
</tr>
<tr>
<td>Power</td>
<td>110V/220V</td>
</tr>
<tr>
<td>Board size(mm)</td>
<td>310 x 265</td>
</tr>
<tr>
<td>Wood case(mm)</td>
<td>100(H) x 300(D) x 430(W) ± 10(mm)</td>
</tr>
</tbody>
</table>

**About MDA – 8086:**

- MDA is a company name “Midas Engineering”.
- It is a trainer board of 8086 microprocessor having an IC on 8086 inside.
- It consists of an LCD screen, 16 data keys and 10 function keys.
It has 2 RAM (2x32Kb) and 2 ROM (2x32Kb) included.

It can be operated in two modes
  o  KIT MODE
  o  PC MODE

It also includes a 7 segment, dot matrix, D/A, A/D, Level meter and stepping motor projects.

LAB#6

Objective
To understand the basic operations of MDA 80x86 trainer kit

Theory
On a power up, following message will be displayed on LCD.

![MDA-8086 Kit!!](Midas 2109-5964)

Or

Serial monitor!

Midas 2109-5964

Figure 1-1.

Figure 1-2.

So as to use serial monitor, move jumper P1 which located on the PCB like this.

![F1](KEYBOARD)

![F1](KEYBOARD)

Machine code

Serial monitor

Whenever RES is pressed, the display becomes FIGURE 1-1 and user can operate keyboard only in this situation.

![RES](System Reset Key)

**EXAMPLE 1.** SET THE CONTENTS IN MEMORY.

<table>
<thead>
<tr>
<th>KEY</th>
<th>LCD</th>
</tr>
</thead>
<tbody>
<tr>
<td>AD</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

[Seg. Oset data] 0000 1000 FF

Input data offset
[The contents of memory 0000:1000 (It may be different)]

[Seg. Oset data] 000F 1000 FF

Input data offset
[The contents of memory 000F:1000 (It may be different)]
0

Segment 0 offset data
00F0 1000 FF

Input data offset
[The contents of memory 00F0:1000
(If may be different)]

Segment 0 offset data
0F00 1000 FF

Input data offset
[The contents of memory 0F00:1000
(If may be different)]

Segment 0 offset data
F000 1000 FF

Input data offset
[The contents of memory F000:1000
(If may be different)]
Microprocessor Based System Design

![Diagram showing memory segmentation and offset]

**KEY**: Increment and decrement to segment & offset address.

If on a power-up or pressing RES key, following message will be displayed on LCD.

**MDA-8086 Kit ! !
Midas 2109-5964**

If on an AD key,

<table>
<thead>
<tr>
<th><strong>KEY</strong></th>
<th><strong>LCD</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>AD</td>
<td>![Diagram showing increment to segment and offset]</td>
</tr>
<tr>
<td>+</td>
<td>![Diagram showing increment to segment and offset]</td>
</tr>
</tbody>
</table>
Lab Objectives:

Task#1: Perform the following operation on MDA – 8086.

1. AD 0000
2. : 1000
3. DA B0 → MOV AL , 3E
4. + 3E
5. + 24
6. + 0F → AND AL , 0F
7. + 90 → NOP
8. STP
9. : Show AX = Result

Calculations:

\[
\begin{align*}
0011 & \quad 1110 & \quad 3E \\
\& 0000 & \quad 1111 & \quad 0F \\
\hline
0000 & \quad 1110 & \quad 0E
\end{align*}
\]

Results:

After entering the above commands in the MDA – 8086, the following result was obtained:

AX = 0E

Which is the verification of the calculated answer?
LAB#7

Objective:
To understand the different commands of MDA 80x86 trainer Kit

Theory

Serial Monitor
Serial monitor is the basic monitor program to do data communicate between MDA-8086 and computer.

Operation serial monitor command
User can only use command which stored at serial monitor. Serial monitor can execute command when user type command and then CR(carriage return) key.

If there is no any command at serial monitor, error message will be Displayed with bell sound and serial monitor prompt will be displayed again.

** 8086 Monitor 1.0 **
** Midas 335-0964/5 **

8086 >?

HELP Command
E segment : offset.................: Enter Data To Memory
D segment : offset length..........: Dump Memory Contents
R [register name]..................: Register Display & Change
M address1, length, address2.......: Move Memory From 1 to 2
F address, length, data............: Fill Memory With Any Data
L Return key......................: Program Down Load
G segment : offset...............: Execute Program
T.................................: Program 1 step execute
Commands:

1. Memory modify command

Syntax: E segment: offset

Purpose: This command is used to enter data to memory.

Example:
8086 > E 0000:1000
  0000:1000 FF ? 11
  0000:1001 FF ? 22
  0000:1002 FF ? 33
  0000:1003 FF ? 44
  0000:1004 FF ? 55
  0000:1005 FF ? (Offset decrement)
  0000:1004 55 ?

2. Memory display command

Syntax: D segment: offset

Purpose: This command is used to display the data stored in memory.

Example:
8086> D 0000:1000
  0000:1000 11 22 33 44 55 FF FF FF - FF FF FF FF FF FF FF FF
  0000:1020 FF FF FF FF FF FF FF FF - FF FF FF FF FF FF FF FF

3. Display Registers Command.

Syntax: R

Purpose: The R command is used to display the 8086 processor registers

Example:
8086 > R

AX=0000 BX=0000 CX=0000 DX=0000
SP=0540 BP=0000 SI=0000 DI=0000
DS=0000 ES=0000 SS=0000 CS=0000
IP=1000 FL=0000 = . . . . . . .
To change individual register:

8086 > R  AX  
AX = 0000  1234

8086 > R  BX  
BX = 0000  4567

8086 > R  CX  
CX = 0000  7788

8086 > R  DX  
DX = 0000  1111

To see the result:

8086 > R

AX=1234  BX=4567  CX=7788  DX=1111  
SP=0540  BP=0000  SI=0000  DI=0000  
DS=0000  ES=0000  SS=0000  CS=0000  
IP=1000  FL=0000  = . . . . . . . .

Lab Objectives:

Task#1:

1-To store the following data to the given addresss

2-Display all registers with its content.

Let’s store the following like to 01000H ~ 01003H

<table>
<thead>
<tr>
<th>ADDRESS</th>
<th>DATA</th>
</tr>
</thead>
<tbody>
<tr>
<td>01000</td>
<td>AB</td>
</tr>
<tr>
<td>01001</td>
<td>CD</td>
</tr>
<tr>
<td>01002</td>
<td>EF</td>
</tr>
</tbody>
</table>
Task#2:
1- To enter data from 0000:0000 memory address to 0000:0020 memory address using command.
2- Display the data you entered in task#3 using command.
3- Change all register content using command then display all registers.
LAB # 8

Objective:
Explore kit mode functionality.

Theory:

**SHL: Shift Logical Left:**
Intel 80x86; shifts the contents of a data register or memory location (8, 16, or 32 bits) to the left (towards most significant bit) by a specified amount (by 1, by 0 to 31 bits specified by an immediate operand.

**SHR: Shift Logical Right:**
Intel 80x86; shifts the contents of a data register or memory location (8, 16, or 32 bits) to the right (towards least significant bit) by a specified amount (by 1, by 0 to 31 bits specified by an immediate operand.

**ROL: Rotate Left:**
Intel 80x86; rotates the contents of a general purpose register or a memory location (8, 16, or 32 bits) to the left (towards the most significant bit) by a specified amount (by 1 bit or by 0 to 31 bits specified by an immediate operand.

**ROR: Rotate Right:**
Intel 80x86; rotates the contents of a general purpose register or a memory location (8, 16, or 32 bits) to the right (towards the least significant bit) by a specified amount (by 1 bit or by 0 to 31 bits specified by an immediate operand.

**Example:**
Try the following code on Dos debugger as well as on 80 x 86 Trainers (Kit mode)

<table>
<thead>
<tr>
<th>Assembly code</th>
<th>Equivalent Machine code</th>
</tr>
</thead>
<tbody>
<tr>
<td>MOV AL,20h</td>
<td>BO 20</td>
</tr>
<tr>
<td>MOV BL,30h</td>
<td>B3 30</td>
</tr>
<tr>
<td>ADD AL,BL</td>
<td>00 D8</td>
</tr>
<tr>
<td>INT 3</td>
<td>CC</td>
</tr>
</tbody>
</table>
Debug View

Program:

Following lab will be performing on kit mode.

Write a program to transfer following values in the specified registers

\[ AL=0E, BL=0C, DL=0D, CL=0A \]

Using MiDas 8086 trainer, find the value of AL at the end of the following instructions:

i) ADD AL, BL
ii) SUB AL, BL
iii) SHL AL, 1
iv) SHR AL, CL
v) ROR AL, 1
vi) ROL AL, 1
vii) ADD AL, DL
viii) AND AL, CL
ix) OR AL, DL
x) XOR AL, AL
xi) NOT AL

Final output will be look like this
Execution Steps:

**Step#1:** Find the machine language code of the given program.

**Step#2:** Type the Machine code in the MDA 80x86 trainer kit mode

**Step#3:** Press STP key and then GO key, and verify the calculated value of AL.

Lab Objectives:

**Task#1:** Addition of Any Five Hexadecimal numbers. e.g (Ax=30,BX=40…….)

**Task#2:** Move data from Ax to Bx,Cx and Dx of task #1.
LAB # 9

Objective:
Write a program to display the digits in decimal, from 0-7 into 7-segment.

Theory:

7 Segment Display

- The 7 segment inside the MDA – 8086 trainer kit can be used to display numbers.
- This requires PIO 8255 ports which are already connected to the 7 segment internally.
- Through the code we can access these ports and provide binary or hex value to switch the required segment on and off.
- In order to turn a segment ON, a logical 0 is required as shown below.

![7 Segment Display Diagram]

- Any number from 0 – 9 can be display on the 7 segment by providing the actual hex or binary value which turns those segments ON to display the digit.

<table>
<thead>
<tr>
<th>8</th>
<th>4</th>
<th>2</th>
<th>1</th>
<th>8</th>
<th>4</th>
<th>2</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>dp</td>
<td>g</td>
<td>f</td>
<td>e</td>
<td>d</td>
<td>c</td>
<td>b</td>
<td>a</td>
</tr>
<tr>
<td>0 0 0 0</td>
<td>0 0 0 0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1 1 1 1</td>
<td>1 1 0 1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1 0 1 0</td>
<td>0 1 0 0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1 0 1 1</td>
<td>1 0 0 1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1 0 0 1</td>
<td>1 0 0 1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1 0 0 1</td>
<td>1 0 0 1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1 0 0 1</td>
<td>1 0 0 1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1 0 0 1</td>
<td>1 0 0 1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1 0 0 1</td>
<td>1 0 0 1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

DECIMAL VALUE | HEX VALUE
00 | 00
F9 | 2A4
B0 | 99
92 | 92
82 | 82
F8 | 80
90 | 90
Source Code

<table>
<thead>
<tr>
<th>Assembly code</th>
<th>Equivalent machine language code</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mov  Al, 80H</td>
<td>BO 80</td>
</tr>
<tr>
<td>; to make all port out</td>
<td></td>
</tr>
<tr>
<td>use 80h</td>
<td>in command byte</td>
</tr>
<tr>
<td>Out 1F, AL</td>
<td>E6 1F</td>
</tr>
<tr>
<td>; mov 80h to port C</td>
<td></td>
</tr>
<tr>
<td>Mov AL,BOH</td>
<td>B0 B0</td>
</tr>
<tr>
<td>; mov B0=3 to AL</td>
<td></td>
</tr>
<tr>
<td>Out 19H, AL</td>
<td>E6 19</td>
</tr>
<tr>
<td>; mov B0=3 to port A</td>
<td></td>
</tr>
<tr>
<td>Int 3</td>
<td>CC</td>
</tr>
</tbody>
</table>

Steps to Follow

1. **STEP#1**: Put the Kit in PC mode, move jumper P1 which located on the kit like this.

2. **STEP#2**: Connect RS232 cable to PC.
3. **STEP#3**: After moving JP1 to serial monitor status and if on a power-up or pressing RES key, following message will be displayed on LCD and data communication is possible with computer.

```
Serial Monitor !
Midas 2109-5964
```

You are now ready to work in PC mode.
4. **STEP#4**: Open wincomm and set segment and offset(e.g:E 0000:1000)
5. **STEP#5**: Write above machine language code in wincomm.
6. **STEP#6**: Press G for execute.
**Screen Shots**

![Image of output](http://program-plc.blogspot.com/)

**Output:** 3” (B0) is displayed on the 7 segment in MDA – 8086

Lab Objectives:

**Task#1:** Write a program to display all digits from 0-9 as shown in the following diagram.

![Image of digital clock](http://program-plc.blogspot.com/)
LAB # 10

Objective:

To initialize DOT MATRIX DISPLAY.

Theory:

Dot Matrix Display

1. The Dot Matrix inside the MDA – 8086 trainer kit can be used to display any pattern of LEDs in the dot matrix display.
2. This requires PIO 8255 ports which are already connected to the Dot Matrix internally.
3. Through the code we can access these ports and provide binary or hex value to switch the required LEDs on and off.
4. In order to turn an LED ON, a logical 0 should be provided to the row and a logical 1 should be provided to the column because of the following arrangement,

5. Any particular shape or design can be formed by turn on the required LEDs on the Dot Matrix Display.

SOURCE CODE:

<table>
<thead>
<tr>
<th>Assembly code</th>
<th>Equivalent machine language code</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mov  AL, 80H</td>
<td>B0 80</td>
</tr>
<tr>
<td>Out  1E, AL</td>
<td>E6 1E</td>
</tr>
<tr>
<td>Mov  AL, 01H</td>
<td>B0 01</td>
</tr>
<tr>
<td></td>
<td>; column to provide logical 1</td>
</tr>
<tr>
<td>Out  1CH, AL</td>
<td>E6 1C</td>
</tr>
<tr>
<td>Mov  AL, 00H</td>
<td>B0 00</td>
</tr>
<tr>
<td></td>
<td>; row to provide logical 0</td>
</tr>
<tr>
<td>Out  1AH, AL</td>
<td>E6 1A</td>
</tr>
</tbody>
</table>
Microprocessor Based System Design

### Screen Shots:

** Mov AL, FF  |  B0 FF  
** Out 18H, AL  |  E6 18  
** Int 3    |  CC

### Steps to Follow

1. **STEP#1**: Put the Kit in PC mode, move jumper P1 which located on the kit like this.

   ![Diagram showing the position of jumper P1 for PC mode]

2. **STEP#2**: Connect RS232 cable to PC.

3. **STEP#3**: After moving JP1 to serial monitor status and if on a power-up or pressing RES key, following message will be displayed on LCD and data communication is possible with computer.

   ![Serial Monitor message on LCD]

   You are now ready to work in PC mode.

4. **STEP#4**: Open wincomm and set segment and offset(e.g:E 0000:1000)

5. **STEP#5**: Write above machine language code in wincomm.

6. **STEP#6**: Press G for execute.
**Output:** First Column LEDs are activated

![Image of LED array](image)

**Lab Objectives:**

*TASK#1:* Write a program to display 2\(^{nd}\) column with alternate colors led’s.

*TASK#2:* Write a program to display a pattern ‘KARACHI’ on DOT MATRIX.
LAB # 11

Objective

Exploring MDE 8051 Microcontroller Trainer Boards

Theory

Introduction to Microcontrollers

A microcontroller (sometimes abbreviated μC, uC or MCU) is a small computer on a single integrated circuit containing a processor core, memory, and programmable input/output peripherals. Microcontrollers are designed for embedded applications, in contrast to the microprocessors used in personal computers or other general purpose applications.

Microcontrollers are used in automatically controlled products and devices, such as automobile engine control systems, implantable medical devices, remote controls, office machines, appliances, power tools, toys and other embedded systems. By reducing the size and cost compared to a design that uses a separate microprocessor, memory, and input/output devices, microcontrollers make it economical to digitally control even more devices and processes.

MDE 8051 Trainer Boards

The 8051 board is a useful tool for embedded control and robotics projects for both students and hobbyists. Its versatile design and programmable microcontroller lets you access numerous peripheral devices and program the board for multiple uses. The board has many I/O connectors and supports a number of programming options including 8051 assembly and C.

The 8051 trainer board has 8 switches and 8 buffered LEDs for connection to the microcontroller, bread board or peripheral devices. It provides access to pins of the 8051 through sip male and female connectors for wiring to bread board or attaching Diligent Pmod™ peripheral modules. Diligent peripheral modules include H-bridges, analog-to-digital and digital-to analog converters, speaker amplifier, switches, buttons, LEDs, as well as converters for easy connection to screw terminals, BNC jacks, servo motors, and more.

Features include:

- A Maxim Semiconductor DS89C450 microcontroller (an 8051/52) with 64K bytes of on-chip Flash memory.
- Eight on-board switches accessible via both male and female connector
- Eight on-board LEDs accessible via both male and female connector
- An on-board voltage regulator (in some versions)
- Two 20-pin male and female connectors allowing access to all 8051 ports of P0, P1, P2, and P3 for connection to external devices such as bread board or Diligent peripheral module boards.
- Support for the Maxim on-chip serial programmer
- Two RS232 compatible Serial ports with DB9 connectors
- An small bread board can be screwed on the board to insert any external IC and connect it to the board

MDE 8051 Trainer Board
The 8051 is designed for embedded control and robotic applications as well as microprocessor experimentation. The 8051 has an on-chip loader/programmer: The loader / programmer are accessed via Serial COM Port #0 DB9 Connector. The 8051 Trainer features a flexible power supply routing system with VCC and GND pin available on 20 pin male and female connector for powering the ICs on the bread board as well as Diligent Pmod peripheral modules connected to the board. Diligent Pmod peripheral modules can be connected to the connectors on the 8051 Trainer board via cables. Diligent has a variety of Pmod interconnect cables available.

**Functional Description**

**Programming the 8051 Trainer**

The 8051 Trainer programming can be accomplished using Serial#0. Programming via Serial#0 requires use of the HyperTerminal program which comes with the Microsoft Windows Operating System.

**Lab Objectives:**

**TASK#1:** How to Test, Download, and Run a Program on the MDE 8051 Trainer Board
Steps to follow

Connecting the 8051 Trainer to PC with HyperTerminal:

1. Connect the Serial 0 connector of the 8051 Trainer to COM port of the PC using DB-9 cable, as shown below.

2. On your PC go to Start, Accessories, Communications, select HyperTerminal.
3. Enter a name for the terminal session.

4. Select the PC COM Port the MDE 8051 is connected to.
Configure HyperTerminal for 9600 8n1 with no hardware flow control

5. Connect the power and put the Switch on PRG, as shown below. (The PRG LED is turned on now)

6. Go to your HyperTerminal. Press Enter couple of times and examine the Hyper Terminal screen. You should now see the following message on the screen.
7. Clear (Erase) the Flash by entering letter K followed by Enter.
8. Ready to download by entering letter L followed by Enter.
9. Using HyperTerminal send a text file to the trainer, like shown below.
10. When the dialogue appears, select all files and navigate to find your HEX file to be sent to the trainer.
11. You should see a series of Gs appear on your screen if the download is successful. Otherwise repeat the process.
12. If the Download is successful, you can move the Switch to Run position and press RESET button to execute the code.
LAB #12

Objective:
Programming 8051 I/O Ports as Input Ports

Theory
The four 8-bit I/O ports P0, P1, P2 and P3 each use 8 pins. All the ports upon RESET are configured as input, ready to be used as input ports. When the first 0 is written to a port, it becomes an output. To reconfigure it as an input, a 1 must be sent to the port. To use any of these ports as an input port, it must be programmed.

PORT 0
It can be used for input or output; each pin must be connected externally to a 10K ohm pull-up resistor. This is due to the fact that P0 is an open drain, unlike P1, P2, and P3. Open drain is a term used for MOS chips in the same way that open collector is used for TTL chips.
In order to make port 0 an input port, the port must be programmed by writing 1 to all the bits. Port 0 is also designated as AD0-AD7, allowing it to be used for both address and data. When connecting an 8051/31 to an external memory, port 0 provides both address and data.

**Port 0**

Port 0 is configured first as an input port by writing 1s to it, and then data is received from that port and sent to P1.

```
MOV A, #0FFH ; A=FF hex
MOV P0, A ; make P0 an i/p port

BACK: MOV A, P0 ; get data from P0
      MOV P1, A ; send it to port 1
      SJMP BACK ; keep doing it
```

**PORT 1**

Port 1 can be used as input or output. In contrast to port 0, this port does not need any pull-up resistors since it already has pull-up resistors internally. Upon reset, port 1 is configured as an input port. To make port 1 an input port, it must be programmed as such by writing 1 to all its bits.

```
PORT 1

Port 1 is configured first as an input port by writing 1s to it, then data is received from that port and saved in R7 and R5

MOV A, #0FFH ; A=FF hex
MOV P1, A ; make P1 an input port

MOV A, P1 ; get data from P1
MOV R7, A ; save it to in reg R7
ACALL DELAY ; wait
MOV A, P1 ; another data from P1
MOV R5, A ; save it to in reg R5
```
PORT 2

Port 2 can be used as input or output. Just like P1, port 2 does not need any pull-up resistors since it already has pull-up resistors internally. Upon reset, port 2 is configured as an input port. To make port 2 an input port, it must be programmed as such by writing 1 to all its bits. In many 8051-based systems, P2 is used as simple I/O. To make port 2 an input port, it must be programmed as such by writing 1 to all its bits. In many 8051-based systems, P2 is used as simple I/O.

PORT 3

Port 3 can be used as input or output. It also does not need any pull-up resistors. Port 3 is configured as an input port upon reset; this is not the way it is most commonly used.

Lab Objectives:

Task#1 Write a program for the DS89C420 to toggle all the bits of P0, P1, and P2 every 1/4 of a second

Source Code in Assembly Language

```assembly
ORG 0
BACK: MOV A,#55H
       MOV P0,A
       MOV P1,A
       MOV P2,A
       ACALL QSDELAY
       MOV A,#0AAH
       MOV P0,A
       MOV P1,A
       MOV P2,A
       ACALL QSDELAY
       SJMP BACK
QSDELAY:
       MOV R5, #11
H3: MOV R4, #248
H2: MOV R3, #255
H1: DJNZ R3, H1
    DJNZ R4, H2
    DJNZ R5, H3
    RET
END
```
LAB# 13

Objective:
Programming 8051 I/O Ports as Input Ports and Output Ports

Theory:
Refer Lab #12

Lab Objectives:
Task#1 Write a program to that takes input from switch to port 1 and display output at port 2

```
ORG 0

MOV A,#0
MOV P2,A ;P2 as output
MOV A,#0FFH
MOV P1,A ;P1 AS INPUT

BACK: MOV A,P1 ;get data from SWs on P1
MOV P2,A ;send it to LEDs on P2
SJMP BACK ;keep doing it
END
```