City University Panthopath Campus

Microprocessor lab Manual

Department of Computer science and engineering

Spring 2015

**Chapter: 1 Assembly Language.**

## Exp: 1 Local Environment Setup and Installing NASM

**Exp: 2** Introduction of Assembly language (Basic syntax, Register, Variable constant, String, Number, Instruction, Addressing mode, condition, Loop).

**Exp: 3** Hello world Program.

**Exp: 4** Understand Register ( display Nine \* on the Screen).

**Exp: 5** Declare variable to Display character “y”.

**Exp:6** Declare constant to Display a string.

**Exp:7** Display 1-9 on the screen using Loop.

**Chapter: 2**

**Exp: 8 Understanding Microprocessor tinnier kit 8086**

**Exp: 9** Program for 8-bit Addition Using IMMIDEATE addressing mode.

**Exp**: 10 Subtraction of two 8-bit numbers using immediate addressing mode.

**Exp**: 11 Multiplication of two 8-bit numbers using immediate addressing mode

**Exp: 12** Division of two 8-bit numbers using immediate addressing mode.

**Exp: 13Addition** and **Subtraction** of two 16-bit numbers using **direct** addressing mode.

**Exp**: 14 **Addition** and **Subtraction** of two 8-bit numbers using **indirect** addressing mode.

Exp: 15 **Addition** and **Subtraction** of two 32-bit numbers.

**Exp: 16 (Introductory 7 segment display)** Display Hexadecimal digits A on the 7 segment display

**Exp: 17 Display Hexadecimal Digit 0-9 A,B,C,D,E,F.**

**Exp: 18 (Introductory Dot matrix (8\*8) display)** Get on all the LEDs

**Exp: 19** play with row and column (moving Row and column Left to right Top to Bottom.

**Exp: 20 Stepper Motor.**

Exp: 1 Local Environment Setup and Installing NASM

Aim: Installing NASM.

**Local Environment Setup**

Assembly language is dependent upon the instruction set and the architecture of the processor. In this tutorial, we focus on Intel 32 processors like Pentium. To follow this tutorial, you will need:

* An IBM PC or any equivalent compatible computer
* A copy of Linux operating system
* A copy of NASM assembler program

There are many good assembler programs, like:

* Microsoft Assembler (MASM)
* Borland Turbo Assembler (TASM)
* The GNU assembler (GAS)

We will use the NASM assembler, as it is:

* Free. You can download it from various web sources.
* Well documented and you will get lots of information on net.
* Could be used on both Linux and Windows.

**Installing NASM**

If you select "Development Tools" while installing Linux, you may get NASM installed along with the Linux operating system and you do not need to download and install it separately. For checking whether you already have NASM installed, take the following steps:

* Open a Linux terminal.
* Type **whereis nasm** and press ENTER.
* If it is already installed, then a line like, nasm: /usr/bin/nasm appears. Otherwise, you will see just nasm:, then you need to install NASM.

To install NASM take the following steps:

* Check [The netwide assembler (NASM)](http://www.nasm.us/) website for the latest version.
* Download the Linux source archive nasm-X.XX. ta .gz, where X.XX is the NASM version number in the archive.
* Unpack the archive into a directory which creates a subdirectory nasm-X. XX.
* cd to nasm-X. XX and type **./configure** . This shell script will find the best C compiler to use and set up Makefiles accordingly.
* Type **make** to build the nasm and ndisasm binaries.
* Type **make install** to install nasm and ndisasm in /usr/local/bin and to install the man pages.

This should install NASM on your system. Alternatively, you can use an RPM distribution for the Fedora Linux. This version is simpler to install, just double-click the RPM file.

**Exp: 2** Introduction of Assembly language.

Aim: Understanding Basic syntax, Register, Variable constant, String, Number, Instruction, Addressing mode, condition, Loop.

**Basic syntax:**

An assembly program can be divided into three sections:

* The **data** section
* The **bss** section
* The **text** section

**The data Section**

The **data** section is used for declaring initialized data or constants. This data does not change at runtime. You can declare various constant values, file names or buffer size, etc., in this section.

The syntax for declaring data section is:

section .data

**The bss Section**

The **bss** section is used for declaring variables. The syntax for declaring bss section is:

section .bss

**The text section**

The **text** section is used for keeping the actual code. This section must begin with the declaration **global \_start**, which tells the kernel where the program execution begins.

The syntax for declaring text section is:

section .text

global \_start

\_start:

**Comments**

Assembly language comment begins with a semicolon (;). It may contain any printable character including blank. It can appear on a line by itself, like:

; This program displays a message on screen

Register:

To speed up the processor operations, the processor includes some internal memory storage locations, called **registers**.

The registers store data elements for processing without having to access the memory. A limited number of registers are built into the processor chip.

**Processor Registers**

There are ten 32-bit and six 16-bit processor registers in IA-32 architecture. The registers are grouped into three categories:

* General registers
* Control registers
* Segment registers

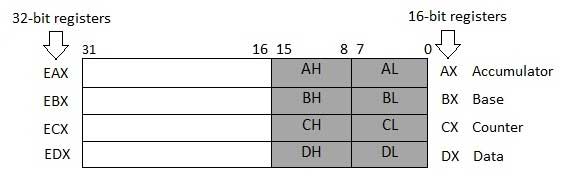
The general registers are further divided into the following groups:

* Data registers
* Pointer registers
* Index registers

**Data Registers**

Four 32-bit data registers are used for arithmetic, logical and other operations. These 32-bit registers can be used in three ways:

1. As complete 32-bit data registers: EAX, EBX, ECX, EDX.
2. Lower halves of the 32-bit registers can be used as four 16-bit data registers: AX, BX, CX and DX.
3. Lower and higher halves of the above-mentioned four 16-bit registers can be used as eight 8-bit data registers: AH, AL, BH, BL, CH, CL, DH, and DL.



Some of these data registers have specific use in arithmetical operations.

**AX is the primary accumulator**; it is used in input/output and most arithmetic instructions. For example, in multiplication operation, one operand is stored in EAX or AX or AL register according to the size of the operand.

**BX is known as the base register** as it could be used in indexed addressing.

**CX is known as the count register** as the ECX, CX registers store the loop count in iterative operations.

**DX is known as the data register**. It is also used in input/output operations. It is also used with AX register along with DX for multiply and divide operations involving large values.

Variable:

Where, variable-name is the identifier for each storage space. The assembler associates an offset value for each variable name defined in the data segment.

There are five basic forms of the define directive:

|  |  |  |
| --- | --- | --- |
| **Directive** | **Purpose** | **Storage Space** |
| DB | Define Byte | allocates 1 byte |
| DW | Define Word | allocates 2 bytes |
| DD | Define Doubleword | allocates 4 bytes |
| DQ | Define Quadword | allocates 8 bytes |
| DT | Define Ten Bytes | allocates 10 bytes |

constant:

There are several directives provided by NASM that define constants. We have already used the EQU directive in previous chapters. We will particularly discuss three directives:

* EQU
* %assign
* %define

**The EQU Directive**

The **EQU** directive is used for defining constants. The syntax of the EQU directive is as follows:

CONSTANT\_NAME EQU expression

For example,

TOTAL\_STUDENTS equ 50

**Instruction:**

## he INC Instruction

The INC instruction is used for incrementing an operand by one. It works on a single operand that can be either in a register or in memory.

### Syntax:

The INC instruction has the following syntax:

INC destination

The operand destination could be an 8-bit, 16-bit or 32-bit operand.

### Example:

INC EBX ; Increments 32-bit register

INC DL ; Increments 8-bit register

INC [count] ; Increments the count variable

## The DEC Instruction

The DEC instruction is used for decrementing an operand by one. It works on a single operand that can be either in a register or in memory.

### Syntax:

The DEC instruction has the following syntax:.

DEC destination

The operand destination could be an 8-bit, 16-bit or 32-bit operand.

## The ADD and SUB Instructions

The ADD and SUB instructions are used for performing simple addition/subtraction of binary data in byte, word and doubleword size, i.e., for adding or subtracting 8-bit, 16-bit or 32-bit operands, respectively.

### Syntax:

The ADD and SUB instructions have the following syntax:

ADD/SUB destination, source

The ADD/SUB instruction can take place between:

* Register to register
* Memory to register
* Register to memory
* Register to constant data
* Memory to constant data

Loop

The JMP instruction can be used for implementing loops. For example, the following code snippet can be used for executing the loop-body 10 times.

MOV CL, 10

L1:

<LOOP-BODY>

SUB CL, 1

JNZ L1

The processor instruction set, however, includes a group of loop instructions for implementing iteration. The basic LOOP instruction has the following syntax:

LOOP label

Where, label is the target label that identifies the target instruction as in the jump instructions. The LOOP instruction assumes that the **ECX register contains the loop count**. When the loop instruction is executed, the ECX register is decremented and the control jumps to the target label, until the ECX register value, i.e., the counter reaches the value zero.

The above code snippet could be written as:

mov ECX,10

l1:

<loop body>

loop l1

**Exp: 3** Hello world Program.

Aim: Display Hello world on the screen.

Requirements software: Assembly Compiler.

**section .text**

**global \_start ;must be declared for linker (ld)**

**\_start: ;tells linker entry point**

**mov edx,len ;message length**

**mov ecx,msg ;message to write**

**mov ebx,1 ;file descriptor (stdout)**

**mov eax,4 ;system call number (sys\_write)**

**int 0x80 ;call kernel**

**mov eax,1 ;system call number (sys\_exit)**

**int 0x80 ;call kernel**

**section .data**

**msg db 'Hello, world!', 0xa ;our dear string**

**len equ $ - msg ;length of our dear string**

output: Hello world.

**Exp: 4** Understand Register (display Nine \* on the Screen).

Aim : Understand Register

Requirements software: Assembly Compiler

**section .text**

**global \_start ;must be declared for linker (gcc)**

**\_start: ;tell linker entry point**

**mov edx,len ;message length**

**mov ecx,msg ;message to write**

**mov ebx,1 ;file descriptor (stdout)**

**mov eax,4 ;system call number (sys\_write)**

**int 0x80 ;call kernel**

**mov edx,9 ;message length**

**mov ecx,s2 ;message to write**

**mov ebx,1 ;file descriptor (stdout)**

**mov eax,4 ;system call number (sys\_write)**

**int 0x80 ;call kernel**

**mov eax,1 ;system call number (sys\_exit)**

**int 0x80 ;call kernel**

**section .data**

**msg db 'Displaying 9 stars',0xa ;a message**

**len equ $ - msg ;length of message**

**s2 times 9 db '\*'**

**When the above code is compiled and executed, it produces the following result:**

**Displaying 9 stars**

**\*\*\*\*\*\*\*\*\***

**Exp: 5** Declare variable to Display character “y”.

Aim: Understand Variable.

Requirements software: Assembly Compiler

**section .text**

**global \_start ;must be declared for linker (gcc)**

**\_start: ;tell linker entry point**

**mov edx,1 ;message length**

**mov ecx,choice ;message to write**

**mov ebx,1 ;file descriptor (stdout)**

**mov eax,4 ;system call number (sys\_write)**

**int 0x80 ;call kernel**

**mov eax,1 ;system call number (sys\_exit)**

**int 0x80 ;call kernel**

**section .data**

**choice DB 'y'**

When the above code is compiled and executed, it produces the following result:

y

**Exp:6** Declare constant to Display a string.

Aim: Understand constant.

Requirements software: Assembly Compiler

**SYS\_EXIT equ 1**

**SYS\_WRITE equ 4**

**STDIN equ 0**

**STDOUT equ 1**

**section .text**

**global \_start ;must be declared for using gcc**

**\_start: ;tell linker entry point**

**mov eax, SYS\_WRITE**

**mov ebx, STDOUT**

**mov ecx, msg1**

**mov edx, len1**

**int 0x80**

**mov eax, SYS\_WRITE**

**mov ebx, STDOUT**

**mov ecx, msg2**

**mov edx, len2**

**int 0x80**

**mov eax, SYS\_WRITE**

**mov ebx, STDOUT**

**mov ecx, msg3**

**mov edx, len3**

**int 0x80**

**mov eax,SYS\_EXIT ;system call number (sys\_exit)**

**int 0x80 ;call kernel**

**section .data**

**msg1 db 'Hello, programmers!',0xA,0xD**

**len1 equ $ - msg1**

**msg2 db 'Welcome to the world of,', 0xA,0xD**

**len2 equ $ - msg2**

**msg3 db 'Linux assembly programming! '**

**len3 equ $- msg3**

**When the above code is compiled and executed, it produces the following result: Hello, programmers!**

**Welcome to the world of,**

**Linux assembly programming!**

Exp:7 Display 1-9 on the screen using Loop.

Requirements software: Assembly Compiler

section .text

global \_start ;must be declared for using gcc

\_start: ;tell linker entry point

mov ecx,10

mov eax, '1'

l1:

mov [num], eax

mov eax, 4

mov ebx, 1

push ecx

mov ecx, num

mov edx, 1

int 0x80

mov eax, [num]

sub eax, '0'

inc eax

add eax, '0'

pop ecx

loop l1

mov eax,1 ;system call number (sys\_exit)

int 0x80 ;call kernel

section .bss

num resb 1

When the above code is compiled and executed, it produces the following result:

123456789:

**Chapter: 2**

**Exp: 8 Understanding Microprocessor tinnier kit 8086**

Get the 8086 user Manual from the following address.

www.midaseng.com

**Exp: 9** Program for 8-bit Addition Using IMMIDEATE addressing mode.

Apparatus Required:

8086 Microprocessor Trainer Kit with Power Supply.

Addition:

Algorithm:

1. Start the Program

2. Initialize the contents of memory locations in AL & BL registers

3. Initialize the value 00 for AL register.

4. Add the value of AL and BL by using ADD Mnemonics and store the result in a memory location.

5. Move the value of AL to a location for output.

6. Stop the Program

PROGRAM:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Program: Offset Address | Opcode | | Label | | Mnemonic | | comment |
| 2000 | | B0 05 | | Mov AL,05 | | Data 05 is stored in reg AL | |
| 2002 | | B3 04 | | Mov BL,04 | | Data 04 is stored in reg BL | |
| 2004 | | 02 C3 | | Add AL,BL | | Data of AL and BL are added and stored in AL | |
| 2006 | | CC | | Int 03 | | Stop | |

RESULT**:** AL: 09

**Exp**: 10 Subtraction of two 8-bit numbers using immediate addressing mode.

Apparatus Required:

8086 Microprocessor Trainer Kit with Power Supply.

Addition:

Algorithm:

1. Start the Program

2. Initialize the contents of memory locations in AL & BL registers

3. Initialize the value 00 for AL register.

4. SUB the value of AL and BL by using SUB Mnemonics and store the result in a memory location.

6. Move the value of AL to a location for output.

7. Stop the Program

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Program: Offset Address | Opcode | | Label | | Mnemonic | | comment |
| 2000 | | B0 05 | | Mov AL,05 | | Data 05 is stored in reg AL | |
| 2002 | | B3 04 | | Mov BL,04 | | Data 04 is stored in reg BL | |
| 2004 | | 2B C3 | | Sub AL,BL | | Data of BL is subtracted from AL and stored in AL | |
| 2006 | | CC | | Int 03 | | Stop | |

RESULT:01

**Exp**: 11 Multiplication of two 8-bit numbers using immediate addressing mode.

Apparatus Required:

8086 Microprocessor Trainer Kit with Power Supply.

Multiplication:

Algorithm:

1. start the program

2. initialize the contents for AX and BX from memory locations

3. Multiple the AX and BX value using MUL Mnemonics and store the result in a memory location.

4. move the content of AX and DX to memory locations to check the result.

5. stop the program

PROGRAM:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Program: Offset Address | Opcode | | Label | | Mnemonic | | comment |
| 2000 | | B0 05 | | Mov AL,05 | | Data 05 is stored in reg AL | |
| 2002 | | B3 04 | | Mov BL,04 | | Data 04 is stored in reg BL | |
| 2004 | | F7 E3 | | Mul BL | | Data of AL and BL are multiplied and RESULT is stored in AL | |
| 2006 | | CC | | Int 03 | | Stop | |

RESULT: AL

**Exp: 13** Division of two 8-bit numbers using immediate addressing mode.

Apparatus Required:

8086 Microprocessor Trainer Kit with Power Supply

Division:

Algorithm:

1. start the program

2. initialize the contents for AX and BX from memory locations

3. Multiple the AX and BX value using MUL Mnemonics and store the result in a memory location.

4. move the content of AX and DX to memory locations to check the result.

5. stop the program

PROGRAM

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Program: Offset Address | Opcode | | Label | | Mnemonic | | comment |
| 2000 | | B0 05 | | Mov AL,05 | | Data 05 is stored in reg AL | |
| 2002 | | B3 04 | | Mov BL,04 | | Data 04 is stored in reg BL | |
| 2004 | | F6 F3 | | Div BL | | Data of AL is divided by BL and RESULT stored in AL | |
| 2006 | | CC | | Int 03 | | Stop | |

RESULT : AL: 01Remainder

AH: 01Quotient

**Exp: 14 Addition** and **Subtraction** of two 16-bit numbers using **direct** addressing mode.

Apparatus Required:

8086 Microprocessor Trainer Kit with Power Supply.

Addition:

Algorithm:

1. Start the Program

2. Initialize the contents of memory locations in AX & BX registers

3. Initialize the value 00 for CL register.

4. Add the value of AX and BX by using ADD Mnemonics and store the result in a memory location.

5. If there is carry Increment the value of CL register by 1, else move the content of AX to a location for result.

6. Move the value of CL to a location for output.

7. Stop the Program

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Offset Address | Opcode | | Label | | Mnemonic | | comment |
| 2000 | | 8B060017 | | Mov AX,[1700] | | Move contents of 1700 in reg AX | |
| 2004 | | 8B1E0217 | | Mov BX,[1702] | | Move contents of 1702 in reg BX | |
| 2008 | | 01D8 | | Add AX,BX | | Data of AX and BX are added and RESULT stored in AX | |
| 200A | | CC | | Int 03 | | Stop | |

RESULT : AX=

Input:-

Location Data

1700 88

1701 00

1702 44

1703 00

Subtraction:

Algorithm:

1. Start the Program

2. Initialize the contents of memory locations in AX & BX registers

3. Initialize the value 00 for CL register.

4. Subtract the value of AX and BX by using SUB Mnemonics and store the result in a memory location.

5. If there is carry Increment the value of CL register by 1, else move the content of AX to a location for result.

6. Move the value of CL to a location for output.

7. Stop the Program

Program:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Offset Address | Opcode | | Label | | Mnemonic | | comment |
| 2000 | | 8B 06 00 17 | | Mov AX,[1700] | | Move contents of 1700 in reg AX | |
| 2004 | | 8B 1E 02 17 | | Mov BX,[1702] | | Move contents of 1702 in reg BX | |
| 2008 | | 29 D8 | | Sub AX,BX | | Data of BX is subtracted from AX and RESULT stored in AX | |
| 200A | | CC | | Int 03 | | Stop | |

RESULT : AX=

Input:-

Location Data

1700 88

1701 00

1702 44

1703 00

RESULT: add and subtract 16 bit number using 8086 Microprocessor Kit was written and executed successfully.

**Exp**: 15 **Addition** and **Subtraction** of two 8-bit numbers using **indirect** addressing mode.

**APPARATUS**: 8086 Trainer kit.

**Addition**

**PROGRAM:**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Program: Offset Address | Opcode | | Label | | Mnemonic | | comment |
| 2000 | | BE0017 | | Mov SI,1700 | | Move Data from 2000 to SI | |
| 2003 | | 8A04 | | Mov AL,[SI] | | Move contents of SI to reg AL | |
| 2005 | | 46 | | Inc SI | | Incrementing SI | |
| 2006 | | 8A1C | | Mov BL,[SI] | | Moving data from SI to BL | |
| 2008 | | 02C3 | | Add AL,BL | | Adding data of AL and BL | |
| 200A | | 89060018 | | Mov [1800],AL | | Move contents of AL to 1800 | |
| 200E | | CC | | Int 03 | | Stop | |

RESULT :

Input:-

Location Data

1700 04

1701 04

Output :-

1800 08

**Subtraction**

**PROGRAM:**

**APPARATUS**: 8086 Trainer kit.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Program: Offset Address | Opcode | | Label | | Mnemonic | | Comment |
| 2000 | | BE0017 | | Mov SI,1700 | | Move Data from 2000 to SI | |
| 2003 | | 8A04 | | Mov AL,[SI] | | Move contents of SI to reg AL | |
| 2005 | | 46 | | Inc SI | | Incrementing SI | |
| 2006 | | 8A1C | | Mov BL,[SI] | | Moving data from SI to BL | |
| 2008 | | 2BC3 | | Sub AL,BL | | Subtract data of BL from data of AL | |
| 200A | | 89060018 | | Mov [1800],AL | | Move contents of AL to 1800 | |
| 200E | | CC | | Int 03 | | Stop | |

RESULT :

Input:-

Location Data

1700 09

1701 04

Output :-

1800 05

Exp: 16 **Addition** and **Subtraction** of two 32-bit numbers.

**APPARATUS REQUIRED:**

8086 Microprocessor Kit

Power Chord

Key Board

**32 - BIT ADDITION:**

**ALGORITHM:**

Step1: Start the program.

Step2: Move immediately the number 0000H to CX register.

Step3: Copy the contents of the memory 3000 to AX register.

Step4: Add the content of the memory 3004 with the content of AX register.

Step5: Copy the content to AX register to two memories from 2000.

Step6: Copy the contents of the memory 3002 to AX register.

Step7: Add the content of the memory 3006 with the content of AX register.

Step8: Jump to specified memory location if there is no carry i.e. CF=0.

Step9: Increment the content of CX register once.

Step10: Copy the content to AX register to two memories from 2002.

Step11: Copy the content to CX register to two memories from 2004.

Step12: End.

**MNEMONICS:**

MOV CX, 0000

MOV AX, [3000]

ADD AX, [3004]

MOV [2000], AX

MOV AX, [3002]

ADC AX, [3006]

JNC loop1

INC CX

Loop1 MOV [2002], AX

MOV [2004], CX

HLT

**OUTPUT:**

**INPUT DATA: OUTPUT DATA:**

3000: 9999 2000: 3332

3002: 9999 2002: 3333

3004: 9999 2004: 1

3006: 9999

**32 - BIT SUBTRACTION:**

**ALGORITHM:**

Step1: Start the program.

Step2: Move immediately the number 0000H to CX register.

Step3: Copy the contents of the memory 3000 to AX register.

Step4: Add the content of the memory 3004 with the content of AX register.

Step5: Copy the content to AX register to two memories from 2000.

Step6: Copy the contents of the memory 3002 to AX register.

Step7: Subtract the content of the memory 3006 from AX register.

Step8: Jump to specified memory location if there is no carry i.e. CF=0.

Step9: Increment the content of CX register once.

Step10: Copy the content to AX register to two memories from 2002.

Step11: Copy the content to CX register to two memories from 2004.

Step12: End.

**MNEMONICS:**

MOV CX, 0000

MOV AX, [3000]

ADD AX, [3004]

MOV [2000], AX

MOV AX, [3002]

SBB AX, [3006]

JNC loop1

INC CX

Loop1 MOV [2002], AX

MOV [2004], CX

HLT

**OUTPUT:**

**INPUT DATA: OUTPUT DATA:**

3000: 9999 2000: 0000

3002: 9799 2002: FE00

3004: 9999

3006: 9999

**RESULT:**

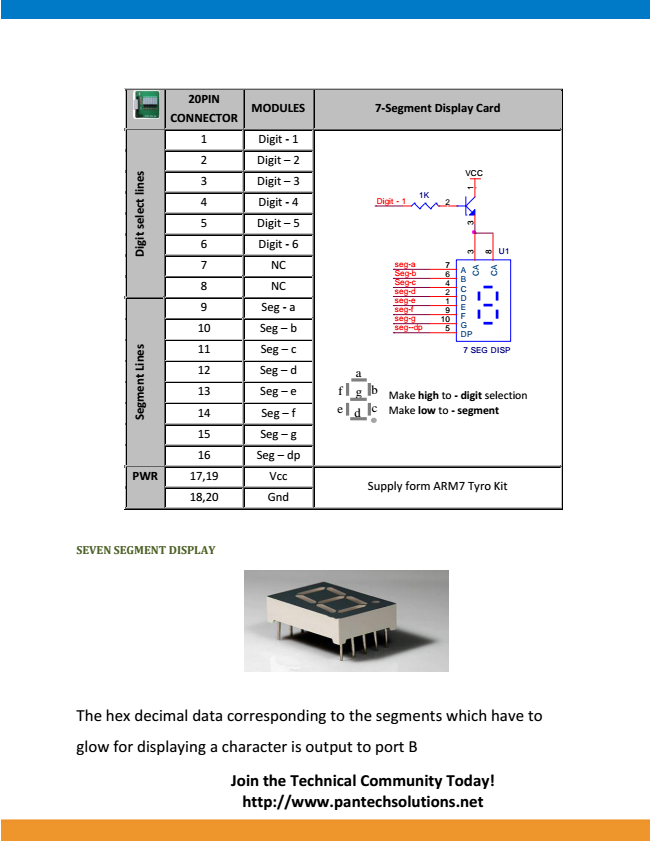
Thus an assembly language program to add and subtract two 32-bit numbers was written and executed using 8086 microprocessor kit.

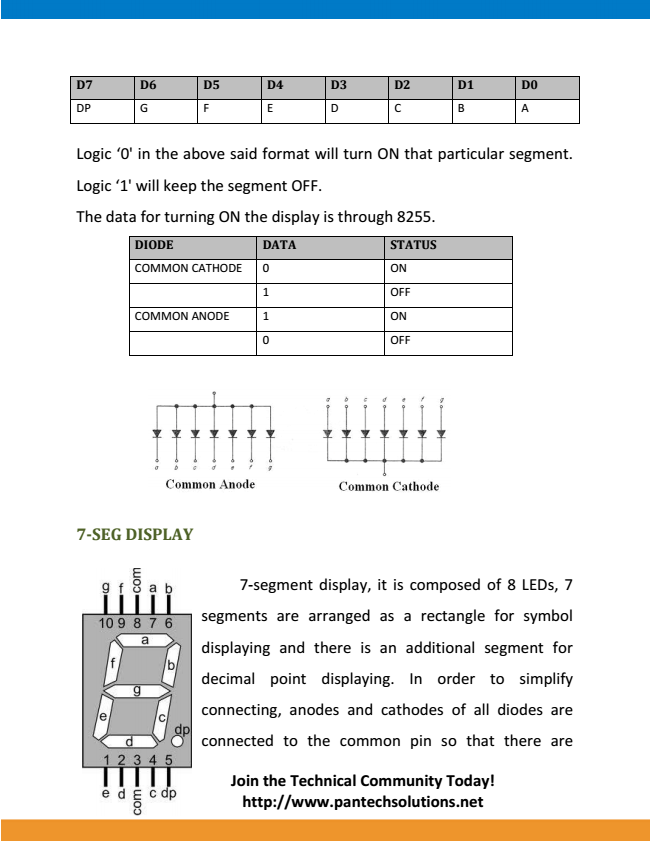
**Exp: 16 (Introductory 7 segment display)** Display Hexadecimal digits A on the 7 segment display.

7-SEG DISPLAY The High Efficiency Red source color devices are made with Gallium Arsenide Phosphate on Gallium Phosphate Orange Light Emitting Diode. 0.56 inch digit height. Low current operation. Excellent character appearance. Easy mounting on P.C. boards or sockets. Mechanically rugged.

SPECIFICATIONS :

Digital Outputs o 6 Nos. of Common Anode (7-SEG) Display. 20-pin Box Header o All LED and Switch | Power lines terminated at box connector 20-pin FRC Cable o To connect host boards (Microcontroller/Processor/FPGA Kits)

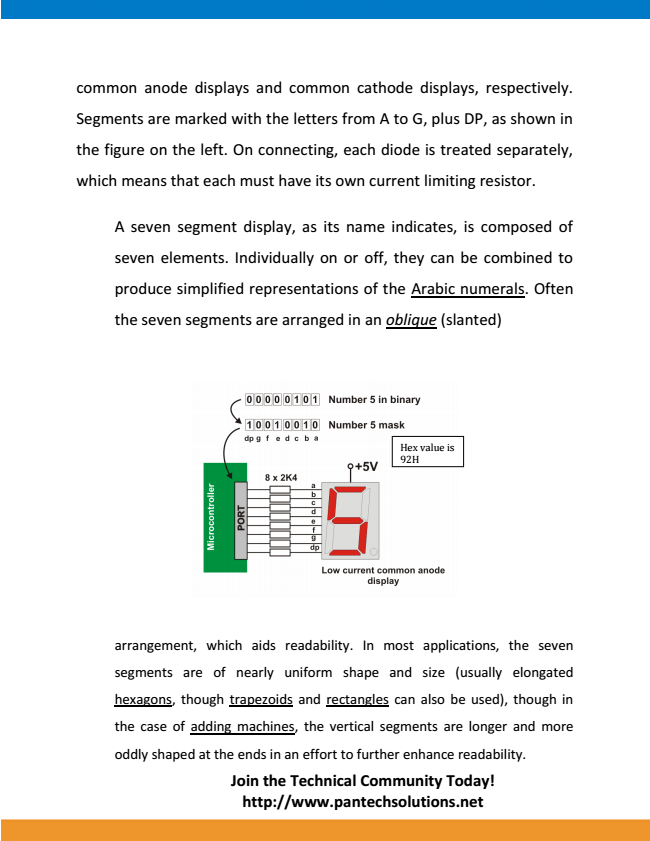


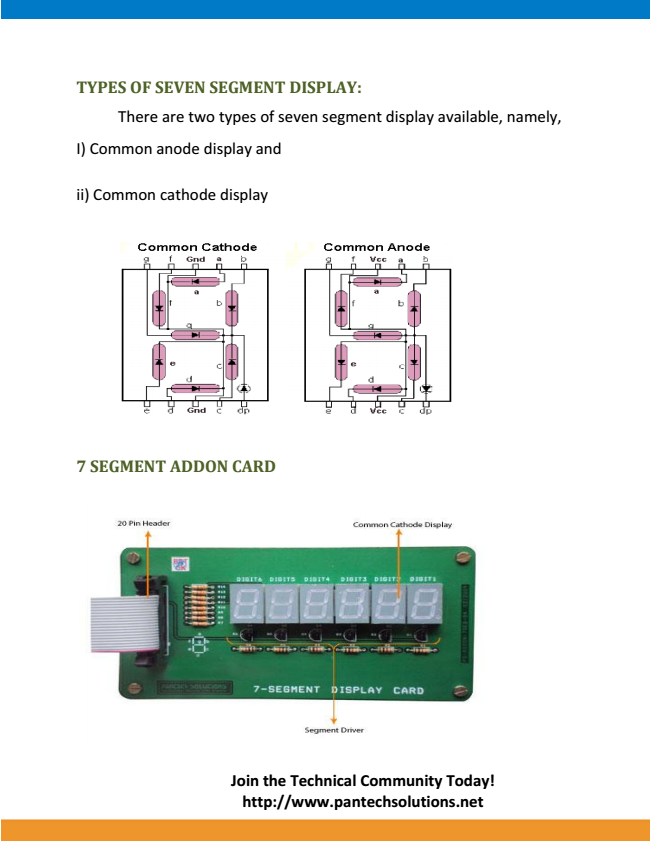


**-SEG DISPLAY**

7-segment display, it is composed of 8 LEDs, 7 segments are arranged as a rectangle for symbol displaying and there is an additional segment for decimal point displaying. In order to simplify connecting, anodes and cathodes of all diodes are

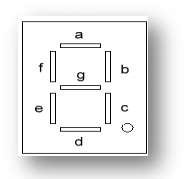
Connected to the common pin so that there are.



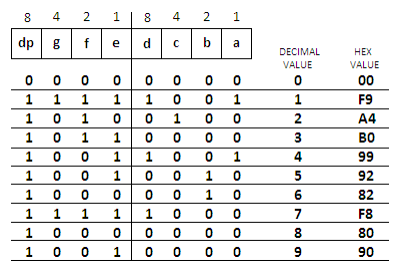


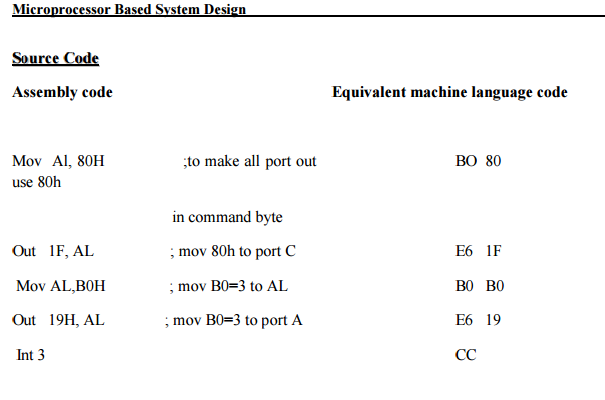
**Exp: 17 Display Hexadecimal Digit 0-9 A,B,C,D,E,F.**

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•



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.





N.B from the code when we change the value of B0 to any other value we will see the Hex digite on the screen notice that we have to write the value by hexadecimal values . like

When we write F9 we will see on the Display 1 A4 for 2 etc.

**Exp: 18 (Introductory Dot matrix (8\*8) display)** Get on one Column.

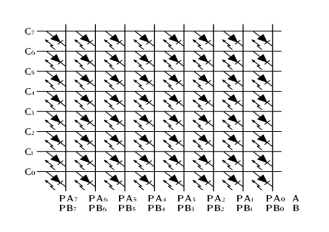
Dot Matrix Display

1. The Dot Matrix inside the MDA – 8086 trainer kit can be used to display any pattern of LEDs in the dot matrix display.

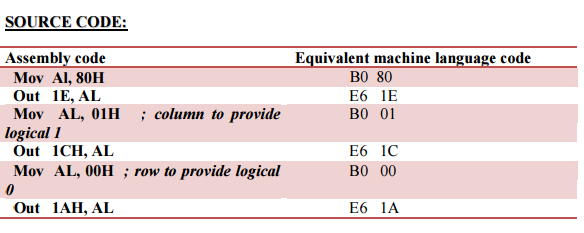
2. This requires PIO 8255 ports which are already connected to the Dot Matrix internally.

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,

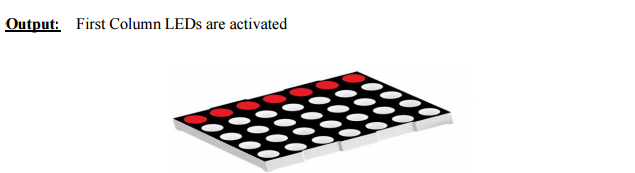


5Any particular shape aur design can be formed by turn on the required LEDs on the Dot Matrix Display.

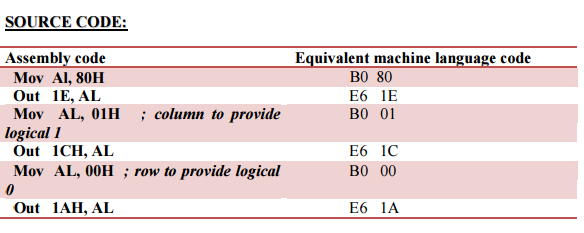


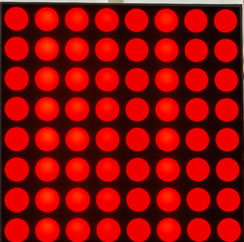
**Notice that Column work on Active High.**

**Row work on active low .**



**Exp: 19** play with row and column (moving Row and column Left to right Top to Bottom.





**When we like to display row and column on the 8\*8 dot matrix display we have to change the value of row and column. Like we like to open all the led’s on for that we have to gave the value column FFH and row 00H**

**Exp: 20 Stepper Motor.**

**AIM:**

To write a program fro inter facing stepper motor and to run the motor in different directions and in different speeds.

**ALGORITHM:**

Step1: Start the program.

Step2: Load HL registers pair with memory address at look up.

Step3: Move the contents of HL pair to accumulator.

Step4: Out the contents of accumulator to run the motor.

Step5: Decrease b register. If register content is not zero then rotate the motor

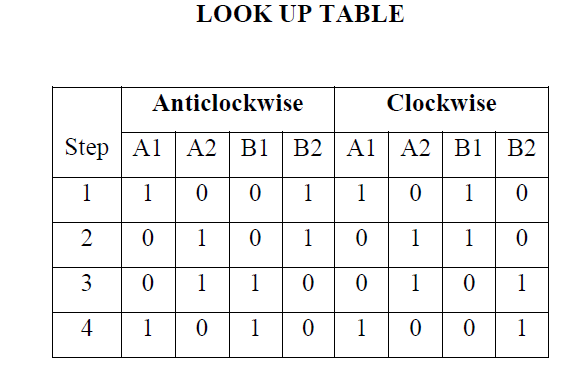
continuously.

Step6: If zero then move to the Seginning of the program.

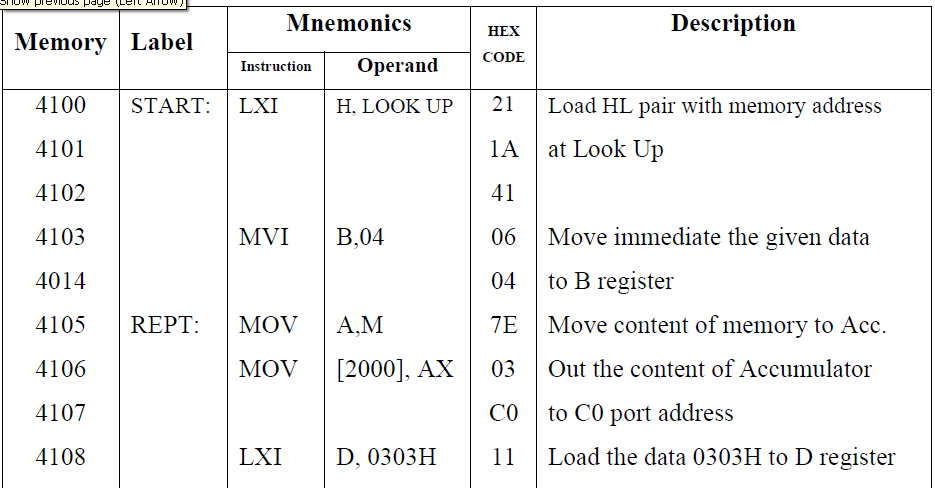
Step7: Stop the process.

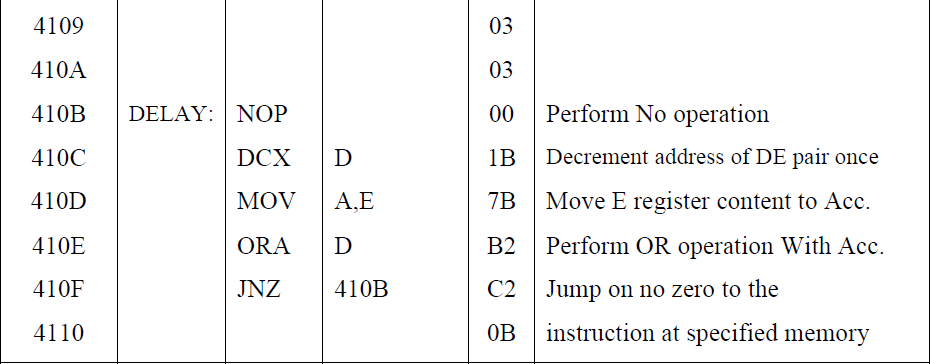
**STEPPER MOTOR:**

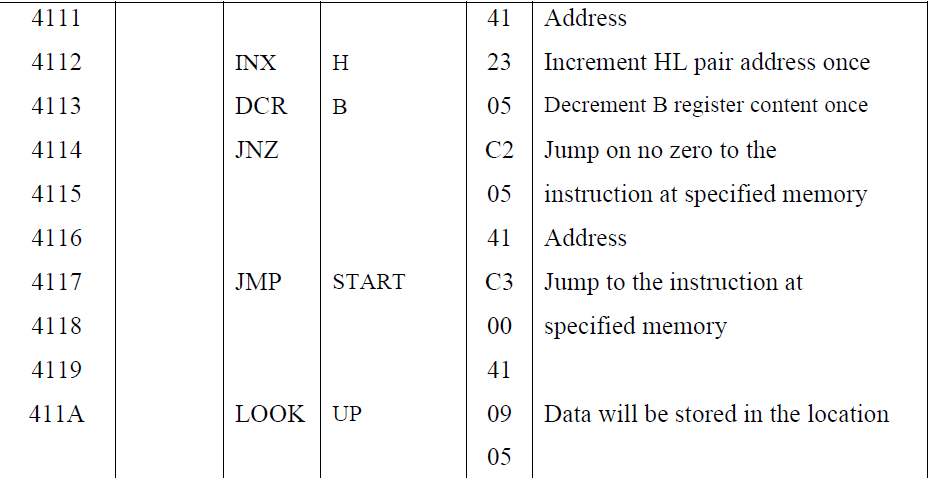
A motor in which the rotor is able to assume only discrete stationary angular position is a Stepper Motor. The rotary motion in a stepper motor is a stepwise manner from one equilibrium position to another.



**Program:**







**RESULT:**

Thus the stepper motor is rotated by varying the speed using COUNT operation and its direction is also changed using program written and executed using