**Lab Session 03**

**OBJECTIVE:**

**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 CLASSIFICATION: 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 REGISTERS:

* 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 SEGMENTS

* 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 INDEX REGISTERS**

* 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 (FLAGS REGISTER)**

* This is a special register with individual bit positions assigned to show he 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 direcitive ; 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:

**Function No. Routine**

02h Single character output

**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.