**Computer Organization & Assembly Language**

**Lab 04**

**Topics:**

1. Directives ($, =, equ, textequ)
2. Arrays and its operations
3. Indirect Addressing
4. PTR operator
5. **Symbolic Definitions:**

A symbolic definition is created by associating an identifier (a symbol) with an integer expression or some text. Symbols do not reserve any memory storage, and their values cannot be changed at runtime. Different directives will be used to create different symbols.

‘$’ is the current location counter and it tells the current location of the variable. It returns the offset associated with the current program statement.

*list db 10,20,30,40,50*

*listlen = ($-list)*

‘Equal sign, =’ directive associates a symbol with an integer expression.

*Name = expression # count = 100*

Any symbol defined with the equal sign directive can be redefined in the code later.

‘equ’ directive associates a symbolic name with an integer expression or some arbitrary text.

*name equ expression # a equ 2+3*

*name equ symbol # a equ var (var defined by = or equ)*

*name equ <text> # a equ <2+3>*

Any symbol defined with the equ directive cannot be redefined in the same source code file.

‘textequ’ directive creates a text macro. There are three different formats:

*var1 = 5*

*name textequ text # count textequ %(var1 \* 2)*

*name textequ <text macro> # move textequ <mov>*

*name textequ %constant\_expression # Copy\_AL textequ <move al,count>*

Any symbol defined with the directive textequ can be redefined at any time.

1. **Arrays and its operations:**

Array is a collection of elements either values or variables, and each element is identified by an array index.

For indexing, registers ‘Destination Index, di’ and ‘Source Index, si’ are used. These registers are 16 bits each and are used to access index of an array. Register bx can also be used for indexing purpose and this is a special purpose of bx register.

Array can be of type byte, word, double word etc. but then extra care must be taken while calling the indices of the array.

For a byte type array +1 is added in the index to access the next element, for a word type array +2 is added and for double word type array +4 is added to the index to access the next element.

1. **Indirect Addressing:**

Indirect addressing is used when we need to fetch a value from a memory location. It is used in case of arrays and in case the variables are saved in memory, and we do not know the memory location. So firstly, the memory location is known and is saved in ‘si’ or ‘di’ registers and then value from that memory location is fetched.

*var db 5*

*mov si, offset var*

*mov al, [si]*

*arr\_1 db 10h, 20h, 30h, 40h, 50h*

*mov si, offset arr\_1*

*mov al, [si+2]*

1. **PTR operator:**

A double word type variable saved in memory cannot be saved in 16 bits registers. For that purpose, ptr operator is used. The type of the pointer determines what values are to be fetched and copied in register. A byte type pointer will copy one byte of that variable, a word type pointer will copy 2 bytes of that variable.

*var 3 dd 12345678h*

*mov al, byte ptr var3*

**Tasks:**

1. What is the difference between the directives, ‘=’, ‘equ’, and ‘textequ’?
2. Verify little endian order of saving variables in memory.
3. Declare and initialize arrays of type byte, word, double word with non-zero elements of your choice.
   1. Find out the type of array and save this value in a variable.
   2. Find out the length of array and save this value in a variable.
   3. Find out the size of the array and save this value in a variable.
   4. Find out the offset of the array and view the array in memory.
4. Using the array declared and initialized in above question, you need to add any number at the even/odd indices of the array. (even/odd determined based on your roll number.)
5. Declare and initialize a variable and access its value using indirect addressing method and view its value in memory location.
6. Using the array in question 2, replace an element in array at the index based on the last digit of your roll number by a variable. View the new array in memory.
7. Declare and initialize a double word type variable in memory, fetch its most significant 2 bytes and make it dividend, make its least significant byte a divisor and perform division. (e.g., if 12345678h is saved in memory, divide 1234h by 08h)
8. Declare and initialize a double or quad word type variable in memory, fetch its most significant byte and make it one multiplier, make its least significant byte second multiplier, and perform multiplication. (e.g., if 12345678h is saved in memory, multiply 12h by 78h)