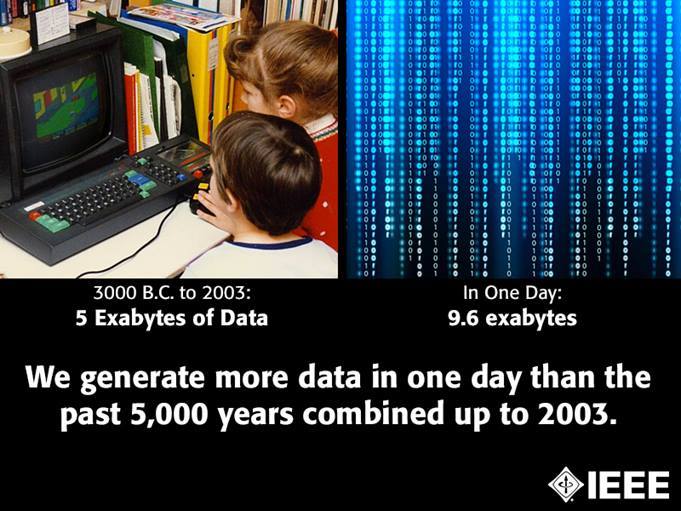
**nLab 04:**

**Data transfer and looping**



**Lab 04**

**Pre-lab:**

Revision of basic concepts and instructions taught in class is mandatory to complete this lab successfully. Few instructions/directive used in lab 04 are

1. **type**
2. **lengthof**
3. **sizeof**
4. **offset**
5. **labels**
6. **loop**

**Software required\*:**

* **MASM 6.15 (Assembler)**
* **32-bit operating systems(Windows)**
* **Notepad editor**

*\*All these stuff are available on group.*

**Goal:**

Upon completion, you will be able to create arrays, find its parameters like storage type (byte, word, and double word), length (total number of elements) and size (total no. of bytes). Further you will learn how to address elements of array using different addressing modes (See lecture slides).Finally; you will implement loops in your program.

**Target of Lab 04:**

1. Learning how to find parameters of declared array.
2. Learning how to use arrays through indirect addressing.
3. Learning how to use loops in Assembly Language.

**Lab Tasks:**

#### Lab task 01: Find Parameters Of Declared Array.

**Procedure:**

1-Declare three arrays of different types in ***.data*** section of your AL program.

*array1* ***byte*** *1, 2, 3, 4*

*array2* ***word*** *10, 20, 30, 40*

*array3* ***Dword*** *100, 200, 300, 400*

2-Write code segment in ***.code*** section using *type,lengthof and sizeof* operator on above declared arrays and record your outputs in the column below.

|  |  |  |  |
| --- | --- | --- | --- |
| **Array name** | **Type** | **Length** | **Size** |
| array1 |  |  |  |
| array2 |  |  |  |
| array3 |  |  |  |

#### Lab task 02: Use indirect addressing with arrays.

**How to use array indexing in your program:**

**Indirect Addressing:** The concept of indirect addressing is very important when working with arrays in assembly language. It allows you to use registers as pointers to memory locations in you programs. You can indirectly access a memory location through a register containing an address by putting square brackets ([ ]) around the name of the register. For Example,

mov ax, bx ; ax = bx – copies the contents of bx into ax, while

mov ax, [bx] ; loads ax with the value stored at the memory location who’s address is in bx

**Example :**

The subsequent code declares an array of 3 elements and reads them into the ax register one-by-one.

|  |  |
| --- | --- |
| **1**  **2**  **3**  **4**  **5**  **6**  **7**  **8**  **9** | **.data**  **arrayW WORD 1000h, 2000h, 3000h**  **.code**  **mov si, OFFSET arrayW ; si now has the offset of arrayW**  **mov ax, [si] ; ax = 1000h**  **add si, 2 ; increment pointer by one word**  **mov ax, [si] ; ax = 2000h**  **add si, 2**  **mov ax, [si] ; ax = 3000h** |

**Program 5.1,Indirect addressing code\***

**\*Note: If on some systems,you got error due to size mismatch,try to change si and ax by esi and eax,respectively.**

**Line – 2**: On line two we have declared a list of 3, 16-bit unsigned integers with the name ‘arrayW’

**Line – 4**: Using the ‘OFFSET’ operator the offset of the first element of ‘arrayW’ from the beginning of the data segment is moved into the general purpose register ‘si’ (source index). In other words, arrayW always points to the address of the first element in the array.

**Line – 5**: ‘[si]’ returns the contents of the memory location stored in the ‘si’ register. This value is 1000h, hence ax will now have 1000h.

**Line – 6**: We now need to increment the pointer to point to the next element. Since the array is of type WORD i.e. 16-bit, we need to increment the offset by 2 bytes, hence we add 2 to the ‘si’ register to make it point to the second element.

**Line – 7**: Reads the second element from memory.

#### Lab task 03: Use loops in programming

**How to create loops**

Loops can be setup using the ‘LOOP’ instruction along with the cx register (which functions as a loop counter) to specify the number of iterations. The syntax is

**LOOP *destination***

The execution of the LOOP instruction involves two steps:

1. Subtract 1 from ‘cx’.
2. Compare ‘cx’ with zero.
3. If ‘cx’ equals zero, execution continues with the instruction following the loop instruction. Otherwise the program branches back to the destination and the next iteration of the loop takes place.

The following code is a simple loop that increments ax 5 times.

**mov ax, 0 ; initializing ax to 0**

**mov cx, 5 ; intitalizing cx to 5**

**doLoop: ; destination to be used in loop**

**inc ax ; incrementing ax**

**loop doLoop ; subtract 1 from cx, if it is not equal to**

**;zero jump to doLoop**

The following example illustrates using loops and arrays together.

**Example for Loops**

The following code sample adds all the integers in an array in memory.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11 | **.data**  **intarray WORD 100h, 299h, 300h, 400h**  **.code**  **mov si, OFFSET intarray ; address of the 1st element**  **mov cx, LENGTHOF intarray ; initialize loop counter**  **mov ax, 0 ; initialize ax**  **loopThing:**  **add ax, [si]**  **add si, TYPE intarray**  **loop loopThing** |

**Program 5.2,loop code \*\***

**\*\*Note: If on some systems,you got error due to size mismatch,try to change si and ax by esi and eax,respectively.**

**Line – 2**: Initializes an array of type WORD

**Line – 5**: Uses the ‘OFFSET’ instruction to place the starting offset of ‘intarray’ in the register ‘si’

**Line – 6:** Uses the ‘LENGTHOF’ instruction to determine the length of the array i.e. the number of elements in the array. This value is copied to ‘cx’ so that it can be used to iterate through all the elements.

**Line – 8:** The label subsequently used by the LOOP instruction indicating the beginning of the loop.

**Line – 9:** Add the current element of the array (pointed to by si) to ax

**Line – 10:** Important: The offset in ‘si’ has to be incremented by the size of the data type used in the array. The ‘TYPE’ instruction can be used to determine the size of one unit of the array. Thus ‘TYPE intarray’ will return 2 because a WORD is made up of 2 bytes. Thus ‘si’ will be incremented by 2 to point to the next element in the array.

**Line – 11:** As mentioned before, the ‘LOOP’ instruction will subtract 1 from cx, compare it with zero and then determine whether to loop again or not. If cx is not zero, the program will branch back to the label loopThing.

**Notes / Comments:**

This lab will help develop concepts related to assembly language programming.