Computer Organization & Assembly Language

BS (CS) \_Fall\_2024

**Lab\_3 Manual**



Learning Objectives:

1. Variables
2. Array

**Variables:**

**What are variables?**

* Variables are used to store values.
* The values of variable can be changed.

**Where to initialize variables in assembly program?**

* Variables are defined in .data directive of program structure.

**Naming Conventions for Variables:**

* Do not use reserved keywords for variable names.
* Reserved keywords are
  + Operands (ADD, SUB, MUL, DIV, MOV, POP, PUSH)
  + Registers (AX, BX, CX, DX, DS, CS etc)

**Initializing a Variable:**

Variable name Variable Initializer Initializing value

Initializer directive defines the size of data. It is also known as Data type directive. Initializing value is the value assigned to the variable.

Example:

**VAR1 DB 49;** Declare a byte, referred to as location Var1, containing the value 49

**VAR2 DB ‘A’;** Declare a byte, referred to as location Var1, containing the value 65

**VAR3 DB ?;** Declare an uninitialized byte, referred to as location Var3

**Initializer Directives (Data Types)**

|  |  |  |
| --- | --- | --- |
| **NAME** | **STAND FOR** | **SIZE** |
| DB | Define Byte | 1 Byte/8 Bits |
| DW | Define Word | 2 Bytes/16 Bits |
| DD | Define Double Word | 4 Bytes/32 Bits |
| DQ | Define Quad Word | 8 Bytes/64 Bits |

**Variable Declaration in Program:**

Example Code:

*.DATA*

*VAR DB 49*

*.CODE*

*MOV AL, VAR*

**Arithmetic Operations on Variables:**

Addition and subtraction in variables is done the same way as in registers. One thing to look for is that the arithmetic operation cannot be perform on two memory locations. There has to be one register and one memory location in order to perform addition and subtraction.

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]*

.386

.model flat,stdcall

.stack 4096

.data

Array dw 10h, 20h, 30h, 40h, 50h ; Initialize an array of 5 words

.code

main PROC

; Direct indexing to access the 1st element

mov ax, Array[0] ; AX = 10h (first element)

; Direct indexing to access the 3rd element

mov ax, Array[4] ; AX = 30h (third element, 4 bytes offset)

; Indirect indexing to access the 3rd element

mov bx, 4 ; Set BX to 4 (offset for 3rd element)

mov ax, Array[bx] ; AX = 30h (accessing via BX)

; Modify the 3rd element using indirect indexing

mov Array[bx], 60h ; Set 3rd element to 60h

; Access the modified value (still via indirect indexing)

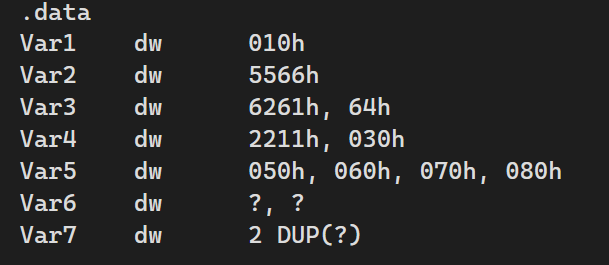
mov ax, Array[bx] ; AX = 60h (modified value)

main ENDP

END main

## Tasks:

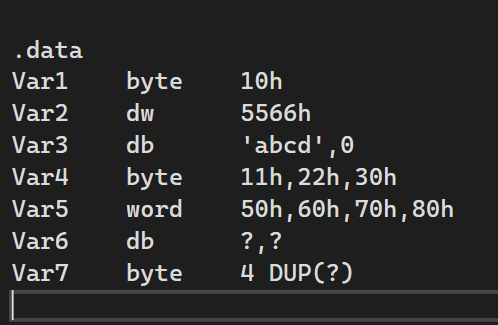
Q1



Fill the memory after execution and also write which format the it is saved in your computer(little endian or big endian).

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 01 | 02 | 03 | 04 | 05 | 06 | 07 | 08 | 09 | 0A | 0B | 0C | 0D | 0E | 0F |
| 0000 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 0010 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 0020 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |

Q2:



Fill the memory in term of big endian format.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 01 | 02 | 03 | 04 | 05 | 06 | 07 | 08 | 09 | 0A | 0B | 0C | 0D | 0E | 0F |
| 0000 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 0010 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 0020 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |

Q3: **Verify Little Endian Memory Storage**

* **Objective**: Confirm that your system uses little-endian format by storing and inspecting a multi-byte value.
* **Steps**:
  1. Declare and initialize a 32-bit variable with a specific value, e.g., 0x12345678.
  2. Store this value in memory.
  3. Verify that the value is stored in little-endian format (i.e., 78 56 34 12).

Q4: **Calculate and Save Array Length**

* **Objective**: Compute the length of an array (number of elements) and store this value.
* **Steps**:
  1. Calculate the length of the byteArray (e.g., 5 elements) and store this in a variable, e.g., arrayLength db 5.
  2. Repeat for wordArray and dwordArray.

Q5: **Calculate and Save Array Size**

* **Objective**: Determine the total memory size occupied by the array and store this information.
* **Steps**:
  1. For the byteArray, the size is simply the number of elements (e.g., 5 bytes).
  2. For the wordArray, the size is the number of elements multiplied by 2 (e.g., 4 words \* 2 bytes = 8 bytes).
  3. For the dwordArray, the size is the number of elements multiplied by 4 (e.g., 3 dwords \* 4 bytes = 12 bytes).
  4. Store these sizes in variables, e.g., arraySize db 5 for bytes, arraySize dw 8 for words, etc.