Lab 5

Variables, Arrays & Loops

Objectives

After completing this lab,

- Students will be able to declare and initialize variables.
- Students will be able to declare and initialize a linear array.
- Students will be able to define a constant.
- Students will be able to implement built-in loop instructions.
- Students will be able to apply base plus index addressing mode to access arrays.
- Students will be able to traverse arrays and perform various operations on them.

Variables

A variable is a memory location. It is easier for a programmer to remember a variable name like "Var1" than an address like **5A73:235B**, especially when there are 10 or more variables. The Emu8086 supports bytes and words as the DB and DB variables, respectively.

Syntax for a variable declaration:

name **DB** value

name **DW** value

DB - for Define Byte.
DW - for Define Word.

name - can be any letter or digit combination, though it should start with a letter. It's possible to declare unnamed variables by not specifying the name (this variable will have an address but no name).

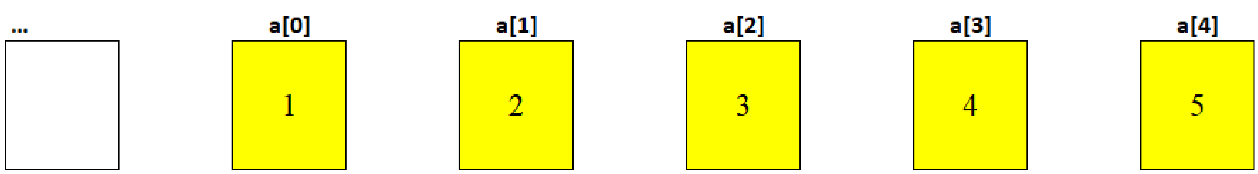
value - can be any numeric value in any supported numbering system (hexadecimal, binary, or decimal), or the "?" symbol for variables that are not initialized.

Arrays

Arrays can be seen as chains of variables. It is a set of consecutive memory locations having the same data type.

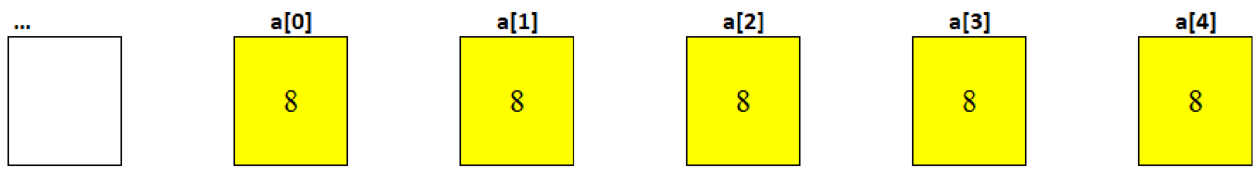
Declaring and initializing an array

a **db** 1,2,3,4,5



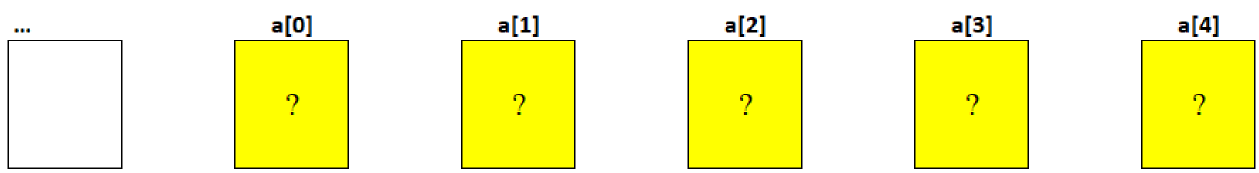
Declaring array of 5 elements and initializing it with 8.

a **db** 5 dup(8)



Declaring array of 5 elements without initializing it.

a **db** 5 dup(?)



Constant

Constants are just like variables, but they exist only until your program is assembled. After assembling, the definition of a constant is replaced with its value. Constant is defined as follows:

name EQU < any expression >

Example:

```
.data
k EQU 5
.code
MOV AX, k
```

The above example is functionally identical to code: MOV AX, 5

Base-plus-Index addressing mode

Arrays can be accessed using direct or register-indirect addressing modes. However, the base-plus-index addressing mode facilitates the programmer in accessing arrays better. In this addressing mode, as the name suggests, there is a base register that points to the base address of an array, while the index register points to the index of the array that is to be accessed. The offset part of the logical address is made by adding the contents of the base and index registers.

There are two base registers: BX and BP, and two index registers: SI and DI. One base register and one index register must be combined to form an offset. There cannot be two base registers or two index registers. If BX is used as the base register, segment addresses will be taken from DS by default. However, in the case of BP, the segment address will be taken from SS by default. However, segment registers can be overridden, as we have already seen in previous labs. The following table shows how physical addresses are calculated.

Instructions	**Default Segment**	**Physical Address Calculation**
MOV AX, [BX + SI]	DS	DS:[BX+SI]
MOV AX, [BP + SI]	SS	SS:[BP + DI]

Program 1: Program to move elements of array to AL,AH, CL,CH and DL registers.

```
.model small

.data

Array db 1,2,3,4,5

.code

Mov ax,@data
Mov ds,ax

Mov bx, offset Array
Mov si, 0

Mov al, [bx + si]
Inc si

Mov ah, [bx + si]
Inc si

Mov cl, [bx + si]
Inc si

Mov ch, [bx + si]
Inc si

Mov dl, [bx + si]

.exit
```

- The **offset** keyword is used to get the address of a variable or an array.
- **SI** is incremented by 1 in the case of a byte array. It will be incremented by 2 for the word array.

Labels

A label is an identifier that is optional and can be placed at the beginning of an instruction. When a program is assembled, the reference to the label is replaced by the offset address.

Example:

```
L1: Mov ax,bx

    Mov bx, L1
```

Loops

Loops are used to execute a set of instructions multiple times until specific conditions are met. There are some built-in loop instructions. One of them is "Loop," which transfers the control to the label specified with it if the counter register (CX) is not zero.

Instruction	Description

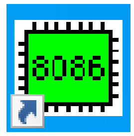
Loop label	CX ← CX – 1
	
	If (CX ! = 0)
	    Jump to label
	else
	    Jump to next instruction

Program 2: Program to increment AX and decrement DX register 100 times

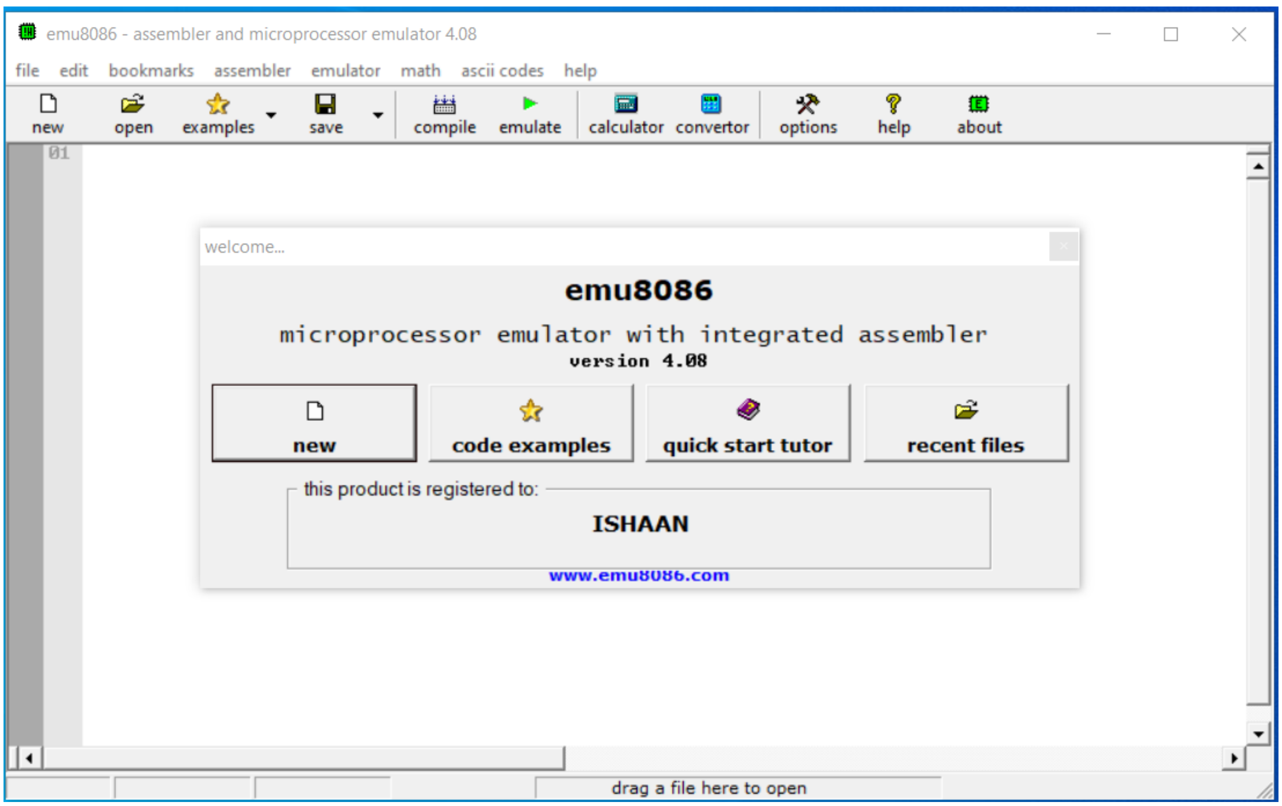
```
.model small

.data

.code
Mov DX,0xffff
Mov CX,100

L1:

Inc AX
Dec DX

Loop L1

.exit
```

Emu8086 Tutorial Step-by-Step

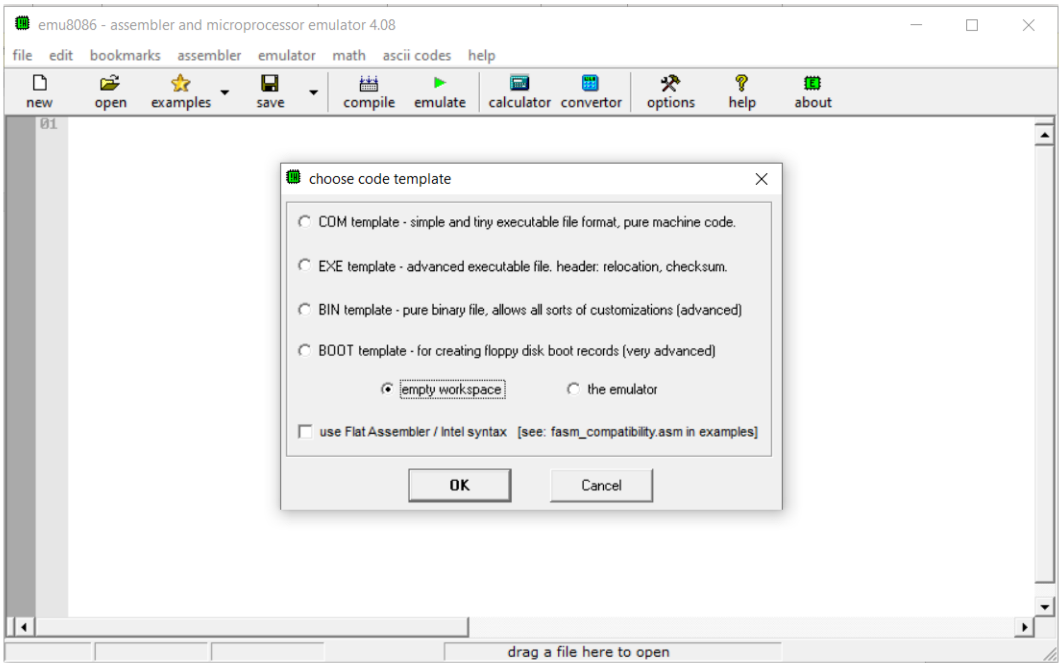
Step-1:

Double-click on the icon on the desktop 

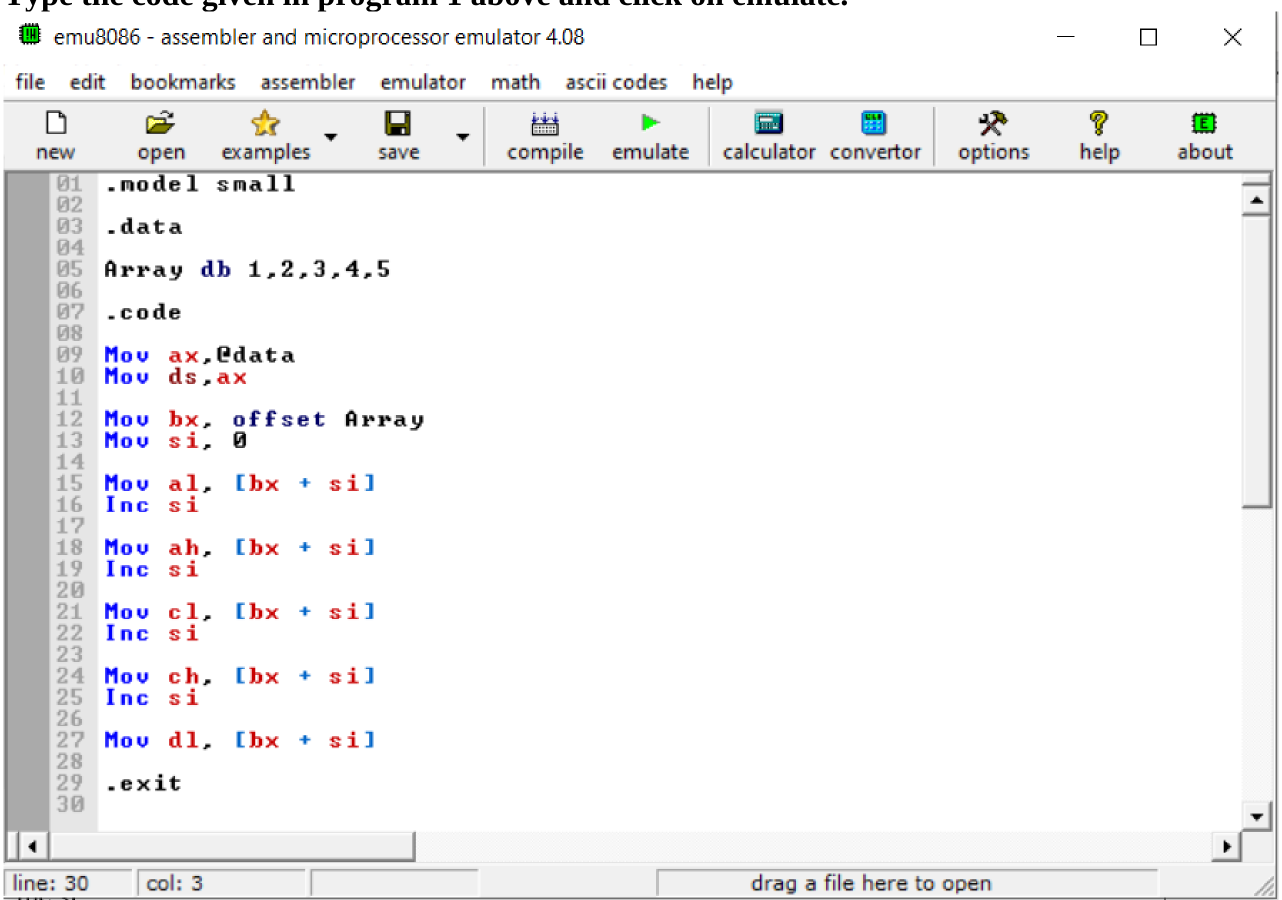
Step-2:
The following window will appear. Click on new.



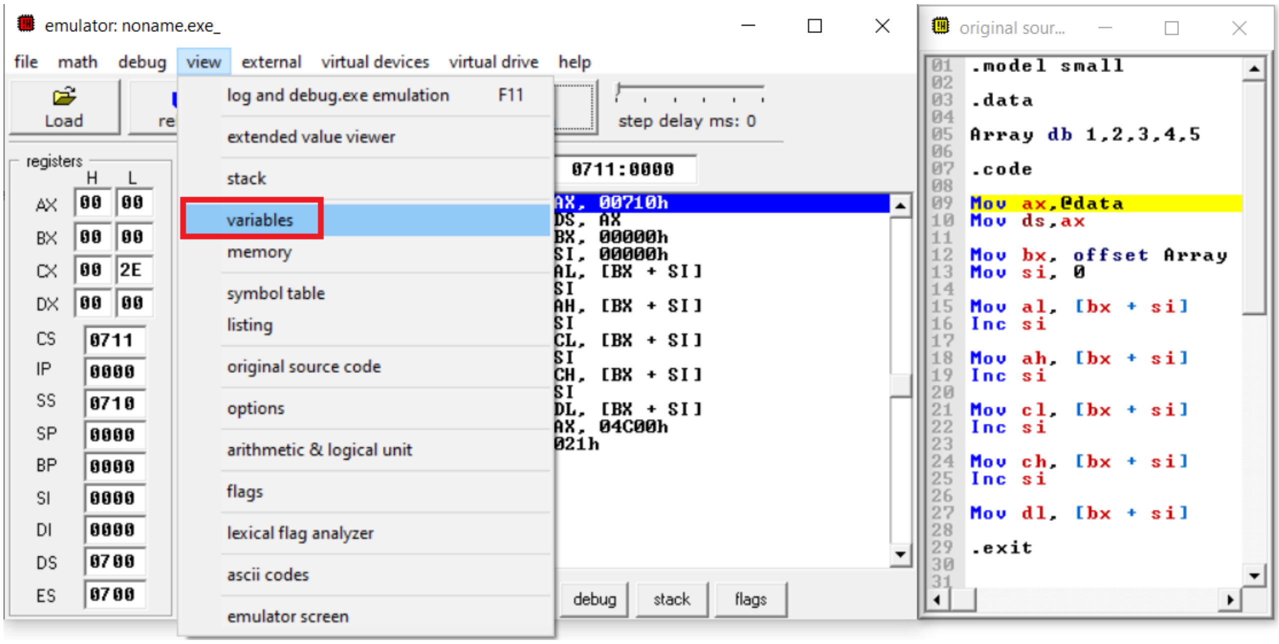
Step-3:
Click on empty workspace and press OK.



Step-4:
Type the code given in program 1 above and click on emulate.

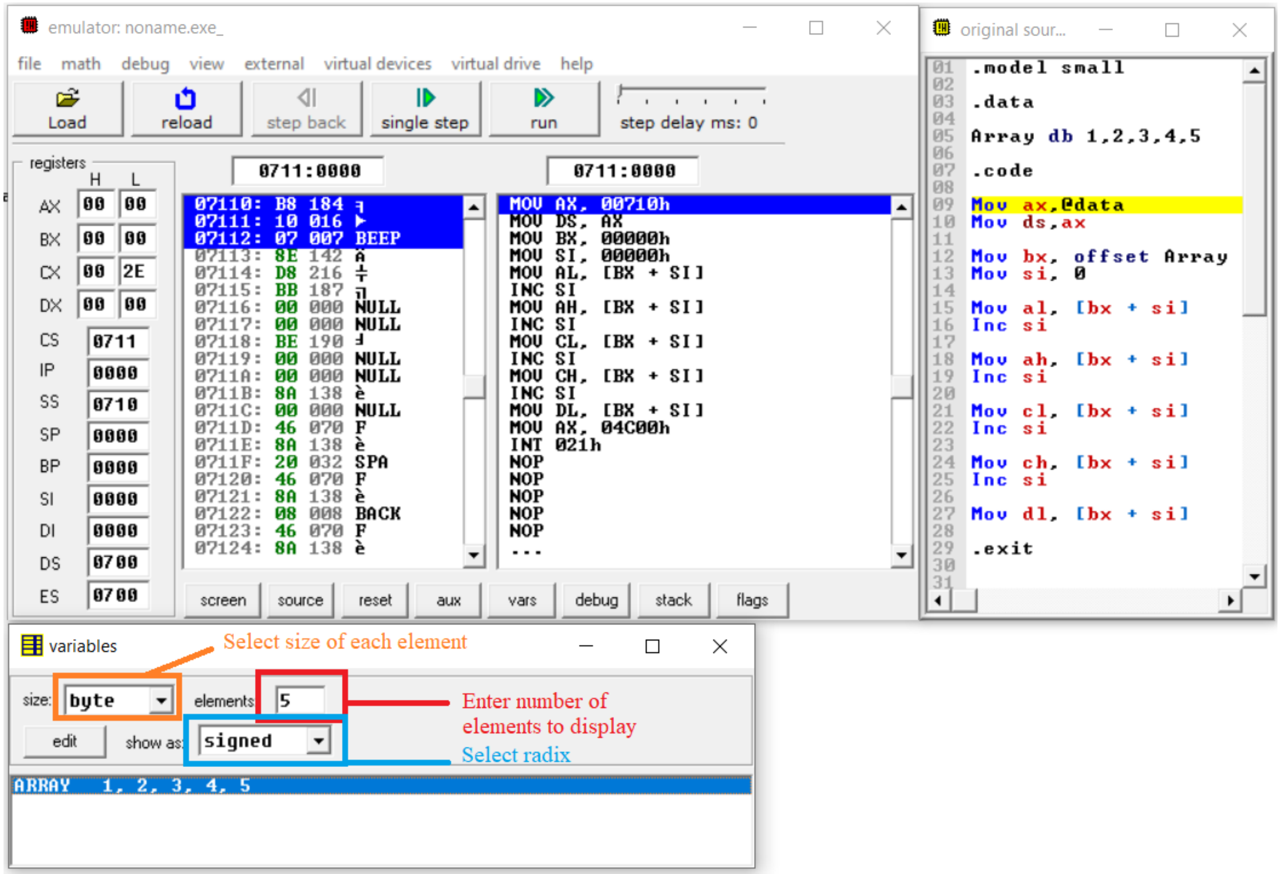


Step-5:
Click on "variables" from the view menu.



Step-6:

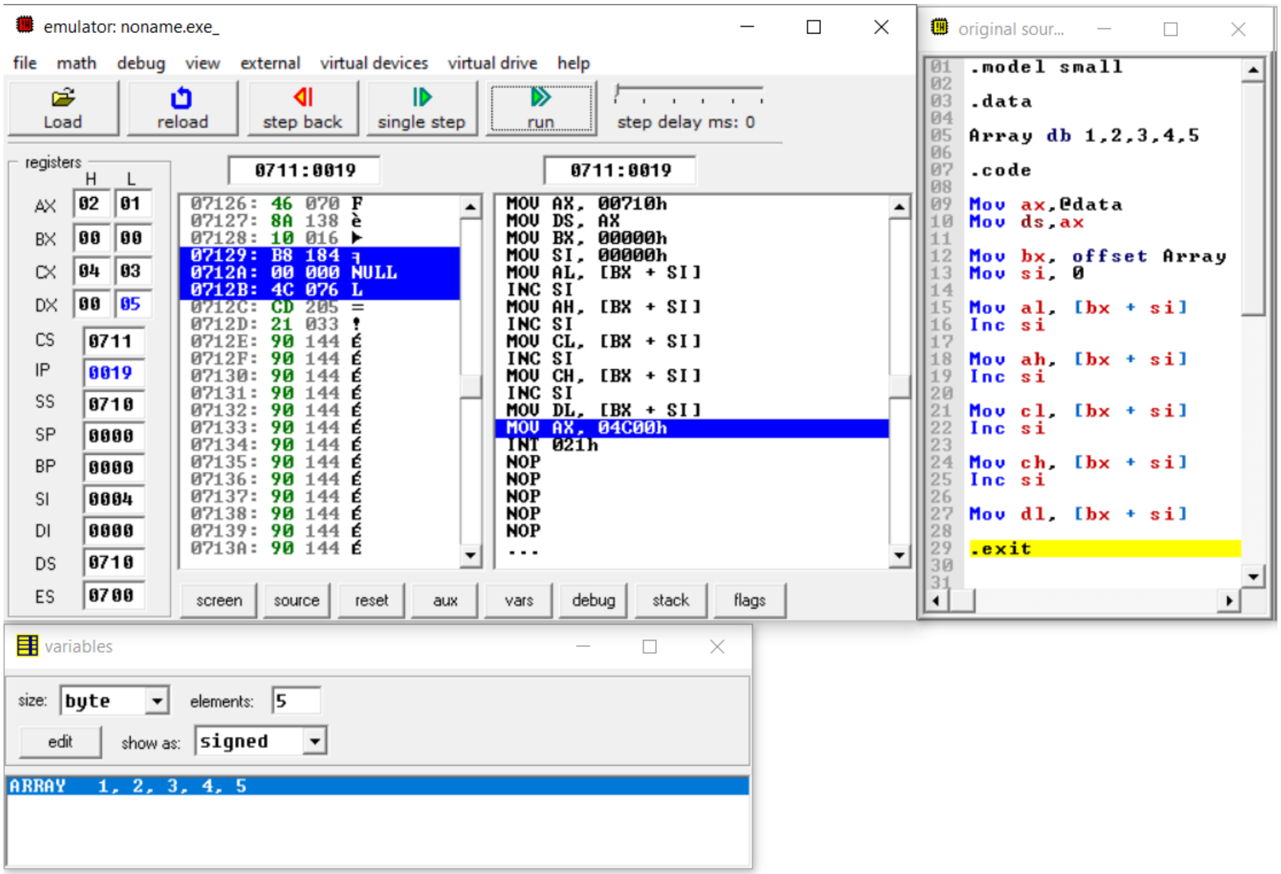
Set size, number of elements, and radix in the variable window.



Step-7:

Keep clicking on "Single step" to execute program instructions one by one and observe the register values side by side.

Stop clicking "Single step" when the ".exit" is highlighted to observe the register values.



Practice Exercise

Task-1

Write a program that declares and initializes an array with 10 elements, then uses a loop to find the sum of those elements and stores the result in a variable named "SUM".

Task-2

Write a program that declares and initializes two word-type arrays: A and B, each of which has 20 elements. The program then adds the corresponding elements of these two arrays and stores the result in the third array: C.