Lab3 Assembly Language

Computer Organization & Architecture

Nedaa Hussein Ahmed

nh1179@fayoum.edu.eg

EMU8086

- Emu8086 combines an advanced source editor, assembler, with debugger, step by step tutorials ,and software emulator (Virtual PC) this completely blocks your program from accessing real hardware.
- Emu8086 is the emulator of 8086 intel microprocessor.
- The emulator runs programs like the real microprocessor in step-by-step mode. It shows registers, memory, stack, variables and flags.
- All memory values can be investigated and edited by a double click.

Integer Constants

An integer constant is made up of an optional leading sign, one or more digits and an optional suffix character(called a radix) indicating the number's base:

[{+I-}] digits [radix]

- IF there's no radix, decimal is default.
- Radix:

H | h	hexadecimal
O | o	octal
D | d	decimal
B | b	binary

Integer Constants cont…

- A hexa decimal constant beginning with a letter must have a leading zero to prevent the assembler from interpreting it as an identifier.

- Examples:

  26
  26d
  10101111b
  1Ah
  0A3h

Integer Expressions

- An integer expression is a mathematical expression involving integer values and arithmetic operators. The expression must evaluate to an integer which can be stored in 32bit (16 bit in our case).

Operator	Name	Precedence Level
( )	parentheses	1
+,-	unary plus, minus	2
*, /	multiply,divide	3
%	modulus	4
+,-	add, subtract	5

Character Constants

- A character constant is a single character enclosed in either single or double quotes.
- The assembler converts it to the binaryASCII code matching the character.
- Examples:

  'A'

  "d"

String Constants

•A string constant is a string of characters enclosed in either single or double quotes.

•Embedded quotes are permitted when used in the manner.

•Examples:

"Goodnight , Gracie"
'4096'
"This isn't a test"
'Say "Goodnight," Gracie'

Reserved Words

- Are List of words that have special meaning and can only be used in their correct context. Some of these words:
  - **Instruction mnemonics,** such as MOV, ADD or MUL, which correspond to built-in operations performed by Intel processors.
  - **Directives,**which tell MASM how to assemble programs or a specific command that can only run on that assembler.
  - **Attributes,** which provide size and usage information for variables and operands. Examples are DB and DW.
  - **Operators,** used in constant expressions.
  - **Predefined symbols** such as @data, which return constant integer values at assembly time.

Identifiers

- An identifier is a programmer chosen name. It might identify a variable, a constant, a procedure, or a code label.
- Rules:
  - between 1 and 247characters.
  - They are not case-sensitive.
  - The first character must be either a letter (A..Z.a..z), _, @@, or $. Subsequent characters may also be digits.
  - An identifier cannot be the same as an assembler reserved word.
- Examples:

```
var1
main
@@myfile
$first
```

Directives

- A directive is a command that is recognized and acted upon by the assembler as the program's source code is being assembled.
- Directives are part of the assembler's syntax, but are not related to the Intel instruction set.
- Directives aren’t case sensitive.
- Example:
  - The .DATA directive identifies the area of a program that contains variables.
  - The .CODE directive identifies the area of a program that contains instructions.
  - The PROC directive identifies the beginning of a procedure. Name may be any identifier:

    name PROC

Instructions

- An instruction is a statement that is executed by the processor at runtime after the program has been loaded into memory and started.
- An instruction contains four basic parts:
  - Label(optional)
  - Instruction mnemonic(required)
  - Operand(s)(usually required)
  - Comment(optional)

Label

- A label is an identifier that acts as a place marker for either instructions or data.
- In the process of scanning a source program, the assembler assigns a numeric address to each program statement. A label placed just before an instruction implies the instruction's address. Similarly, a label placed just before a variable implies the variable's address.

Label cont...

- Code Labels:
  - A label in the code area of a program must end with a colon(:) character.

```
target: mov ax,bx
……
jmp target
```

- Data Labels:
  - If a label is used in the data area of a program, it cannot end with a colon.

```
first DB 10
```

Instruction Mnemonic

- An instruction mnemonic is a short word that identifies the operation carried out by an instruction.
- Some Mnemonics:

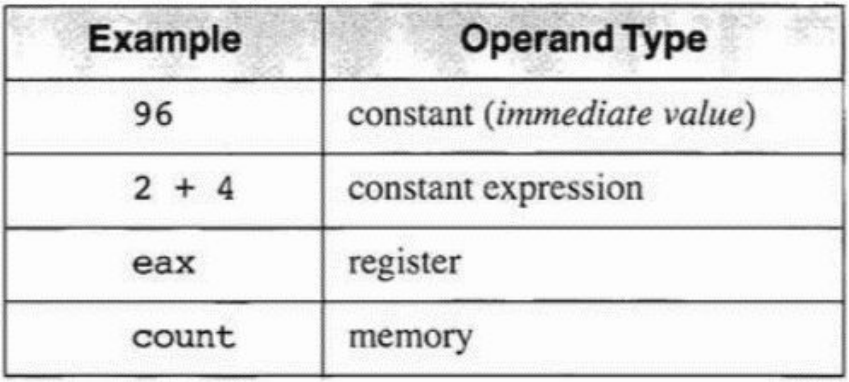
•mov	Move (assign) one value to another
•add	Add two values
•sub	Subtract one value from another
•mul	Multiply two values
•jmp	Jump to a new location
•call	Call a procedure

- We will talk about each one soon.

Operands

- An assembly language instruction can have between zero and three operands, each of which can be a register, memory operand, constant expression, or I/Oport.
- Example:

```
Stc                  ;set Carry flag
inc ax               ;add 1 to ax
mov count,bx         ;move BX to count
```



Example	Operand Type
96	constant (*immediate value*)
2 + 4	constant expression
eax	register
count	memory

Comments

- Comments, as you probably know, are an important way for the writer of a program to communicate information about how the program works to a person reading the source code.
- Comments can be specified in two ways:
  - Single-line comments, beginning with a semicolon character (;)

```
;This is a comment
```

  - Block comments, beginning with the COMMENT directive and a user-specified symbol.

```
COMMENT !
This line is a comment.
This line is also a comment. !
```

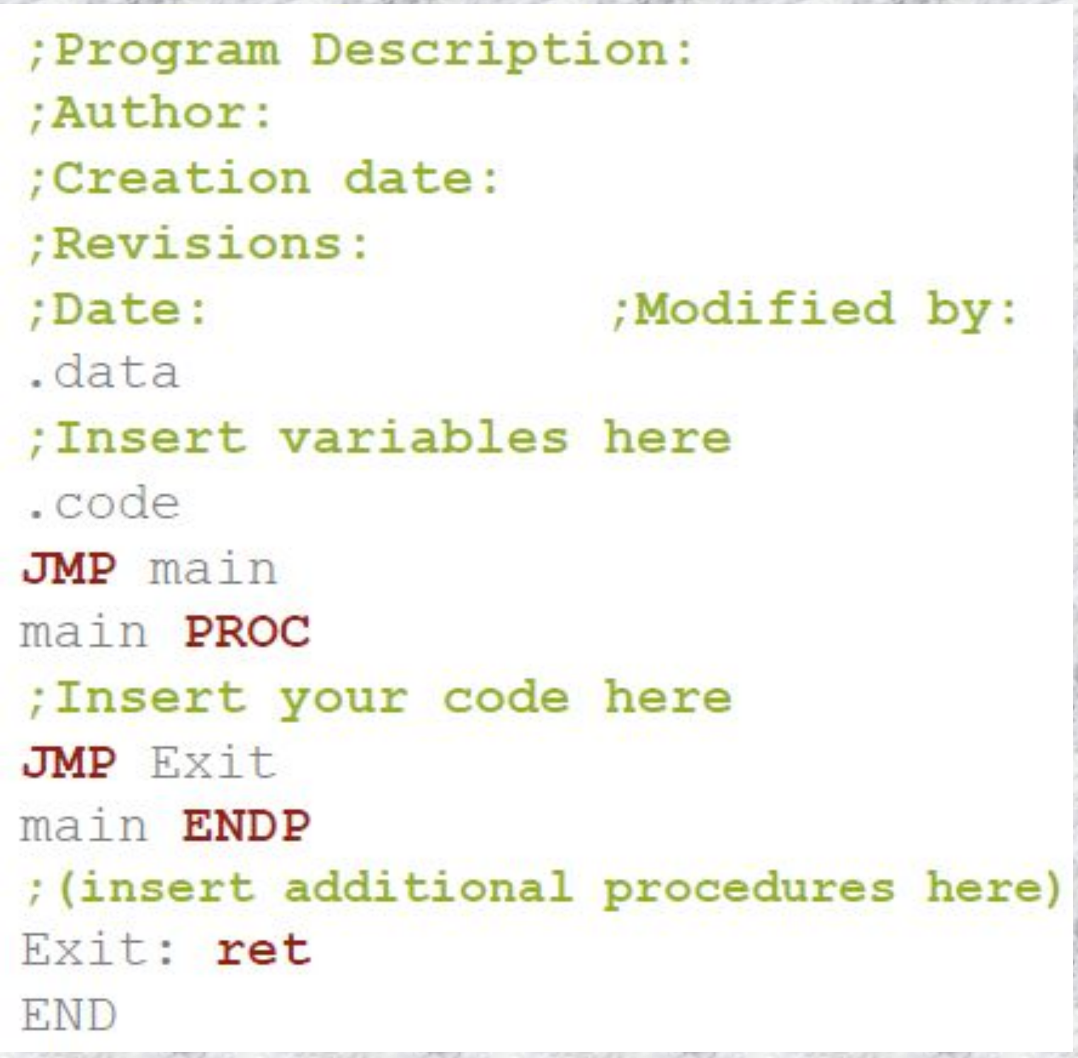
Adding three integers program

```
org 100h
; This program adds 16-bit integers.
.code
jmp main

main proc
    mov ax,100h
    add ax,400h
    add ax,400h
    jmp exit
main endp

exit: ret
END
```

Program template



```
;Program Description:
;Author:
;Creation date:
;Revisions:
;Date:                 ;Modified by:
.data
;Insert variables here
.code
JMP main
main PROC
;Insert your code here
JMP Exit
main ENDP
;(insert additional procedures here)
Exit: ret
END
```

Data Definition Statement

- A data definition statement sets aside storage in memory for a variable and may optionally assign a name to the variable:

[name] directive initializer [,initializer].

- At least one initializer is required in a data definition, even if it is the ? expression, which does not assign a specific value to the data.
- All initializers, regardless of their number format, are converted to binary data by the assembler.

Data Definition Statement cont...

- Variable is a memory location. For a programmer it is much easier to have some value be kept in a variable named "**var1**" then at the address 5A73:235B, especially when you have 10 or more variables.
- Syntax for a variable declaration:

  name **DB** value

  name **DW** value

- **DB** → Define Byte.
- **DW** → Define Word.
- **Name** - can be any letter or digit combination, though it should start with a letter.
- **Value** - can be any numeric value in any supported numbering system (hexadecimal, binary, or decimal), or "**?**" symbol for variables that are not initialized.

Intrinsic Data Types

MASM defines various intrinsic data types, each of which describes a set of values that can be assigned to variables and expressions of the given type.

Keyword MASM	Usage	Keyword 8086
BYTE	8-bit unsigned integer	DB
WORD	16-bit unsigned integer = 2byte	DW
DWORD	32-bit unsigned integer =2 word = 4 byte	DD
QWORD	64-bit integer = 4 word =8 byte	DQ
TBYTE	80-bit integer= 5 word =10 byte	DT

Defining BYTE

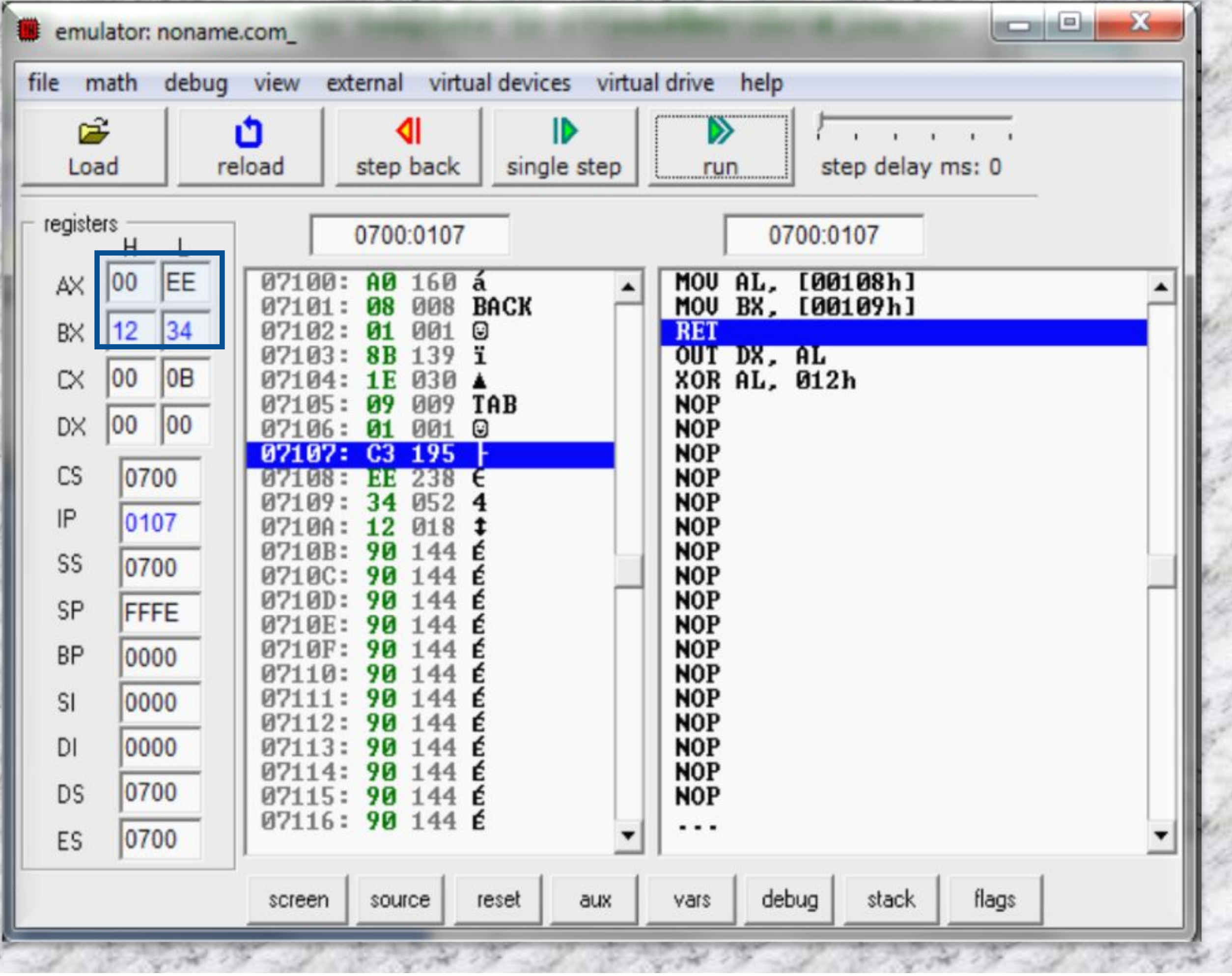
- valuel DB ‘A’ ; character constant
- value2 DB 0 ; smallest unsigned byte
- value3 DB 255 ; largest unsigned byte
- value4 DB ? ; Empty byte
- value5 DB 255 ; unsigned byte
- value6 DB -128 ; signed byte

Example:

Write assembly program that define two variable -12h and 1234h and copy the first number to register AL and the second number to register BX.

```
ORG 100h
 MOV  AL,var1
 MOV  BX,var2
 RET
 VAR1  DB    -12h
 var2     DW   1234h
```

```
ORG 100h
 .data
 VAR1  DB    -12h
 var2     DW   1234h
.code
 MOV  AL,var1
 MOV  BX,var2
 RET
```



	H	L
AX	00	EE
BX	12	34
CX	00	0B
DX	00	00

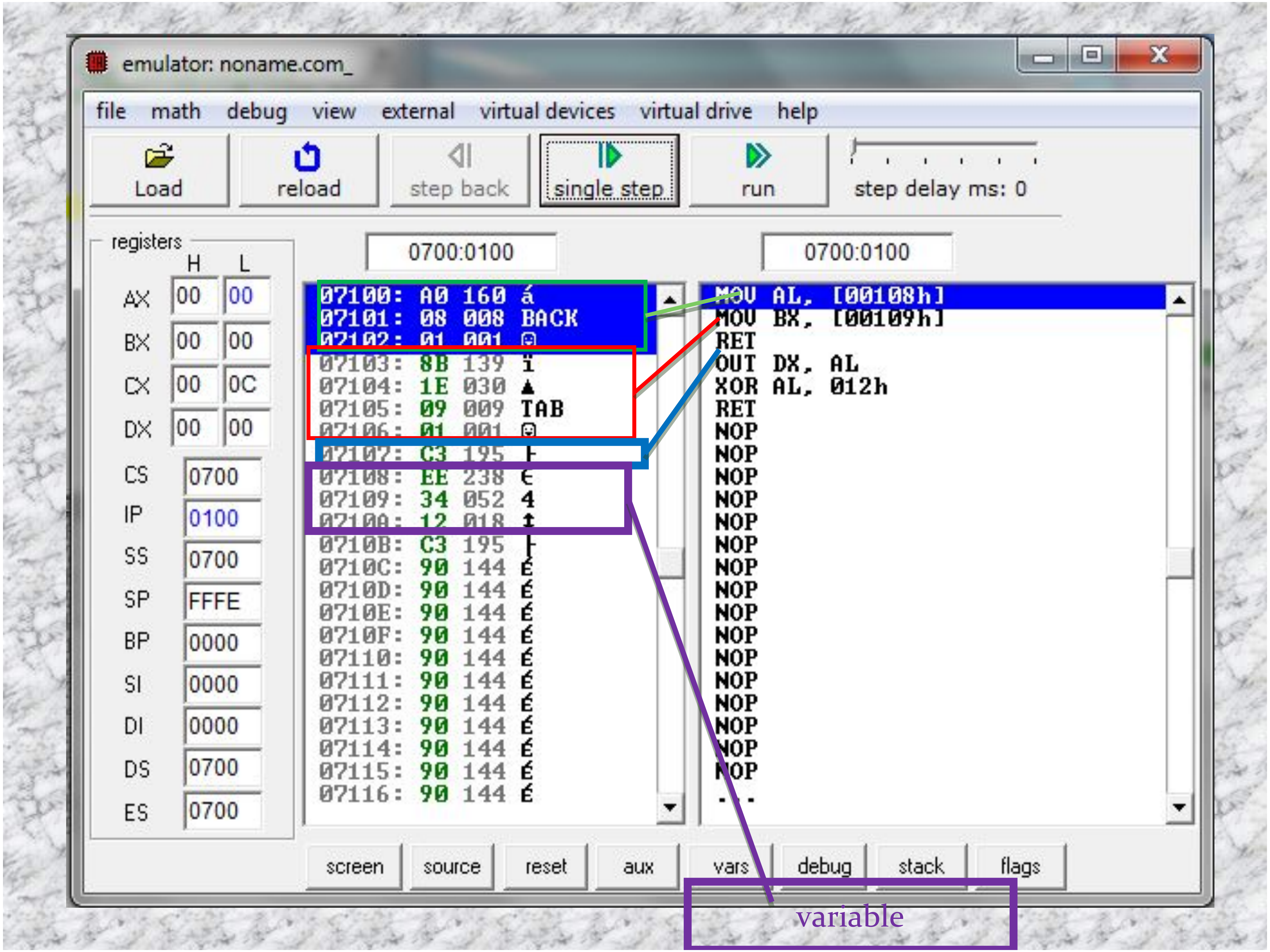
CS	0700
IP	0107
SS	0700
SP	FFFE
BP	0000
SI	0000
DI	0000
DS	0700
ES	0700

```
0700:0107

07100: A0 160 á
07101: 08 008 BACK
07102: 01 001 ☺
07103: 8B 139 ï
07104: 1E 030 ▲
07105: 09 009 TAB
07106: 01 001 ☺
07107: C3 195 ├
07108: EE 238 ε
07109: 34 052 4
0710A: 12 018 ↕
0710B: 90 144 É
0710C: 90 144 É
0710D: 90 144 É
0710E: 90 144 É
0710F: 90 144 É
07110: 90 144 É
07111: 90 144 É
07112: 90 144 É
07113: 90 144 É
07114: 90 144 É
07115: 90 144 É
07116: 90 144 É
```

```
0700:0107

MOV AL, [00108h]
MOV BX, [00109h]
RET
OUT DX, AL
XOR AL, 012h
NOP
NOP
NOP
NOP
NOP
NOP
NOP
NOP
NOP
NOP
NOP
NOP
NOP
NOP
NOP
NOP
NOP
...
```



registers

	H	L
AX	00	00
BX	00	00
CX	00	0C
DX	00	00

CS	0700
IP	0100
SS	0700
SP	FFFE
BP	0000
SI	0000
DI	0000
DS	0700
ES	0700

0700:0100

```
07100: A0 160 á
07101: 08 008 BACK
07102: 01 001 ☺
07103: 8B 139 ï
07104: 1E 030 ▲
07105: 09 009 TAB
07106: 01 001 ☺
07107: C3 195 ├
07108: EE 238 ε
07109: 34 052 4
0710A: 12 018 ↕
0710B: C3 195 ├
0710C: 90 144 É
0710D: 90 144 É
0710E: 90 144 É
0710F: 90 144 É
07110: 90 144 É
07111: 90 144 É
07112: 90 144 É
07113: 90 144 É
07114: 90 144 É
07115: 90 144 É
07116: 90 144 É
```

0700:0100

```
MOV  AL, [00108h]
MOV  BX, [00109h]
RET
OUT  DX, AL
XOR  AL, 012h
RET
NOP
NOP
NOP
NOP
NOP
NOP
NOP
NOP
NOP
NOP
NOP
NOP
NOP
NOP
NOP
NOP
...
```

variable

Multiple initializers

- If multiple initializers are used in the same data definition, its label refers only to the offset of the first byte.
- Example:

```
        .data
        list DB 10,20,30,40
```

Offset	**Value**
0000:	10
0001:	20
0002:	30
0003:	40

Multiple initializers cont…

- Not all data definitions require labels. If we wanted to continue the array of bytes begun with list, example:

```
list   DB 10, 20, 30, 40
       DB 50, 60, 70, 80
       DB 81, 82, 83, 84
```

- Within a single data definition, its initializers can use different radixes.

```
listl   DB  10, 32, 4lh, 00l00010b
list2  DB  0Ah, 20h, 'A', 22h
```

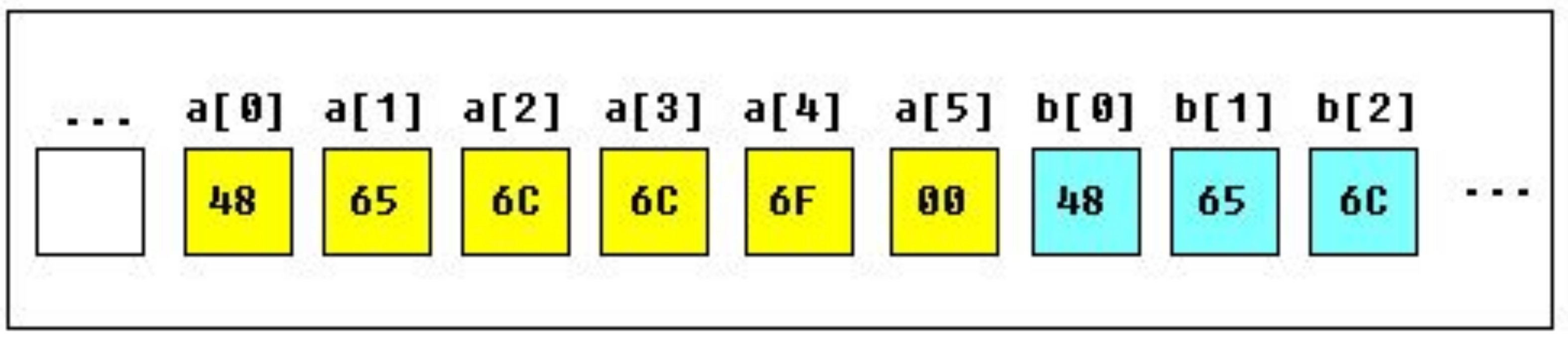
Arrays

- Arrays can be seen as chains of variables.
- A text string is an example of a byte array, each character is presented as an ASCII code value (0..255).

Ex:

a DB 48h, 65h, 6Ch, 6Ch, 6Fh, 00h
b DB 'Hello', 0

- b is an exact copy of the a array, when compiler sees a string inside quotes it automatically converts it to set of bytes.
- This chart shows a part of the memory where these arrays are declared:



...	a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	b[0]	b[1]	b[2]	
	48	65	6C	6C	6F	00	48	65	6C	...

Arrays cont...

- You can access the value of any element in array using square brackets, for example:

```
MOV AL, a[3]
```

- You can also use any of the memory index registers **BX, SI, DI, BP**, for example:

```
MOV     SI, 3
 MOV     AL, a[SI]
```

Defining Strings

- To create a string data definition, enclose a sequence of characters in quotation marks.
- The most common type of string ends with a null byte, a byte containing the value 0. This type of string is used by C/C++, by Java, and by Microsoft Windows functions:

```
greeting1  DB   "Good afternoon",0
```

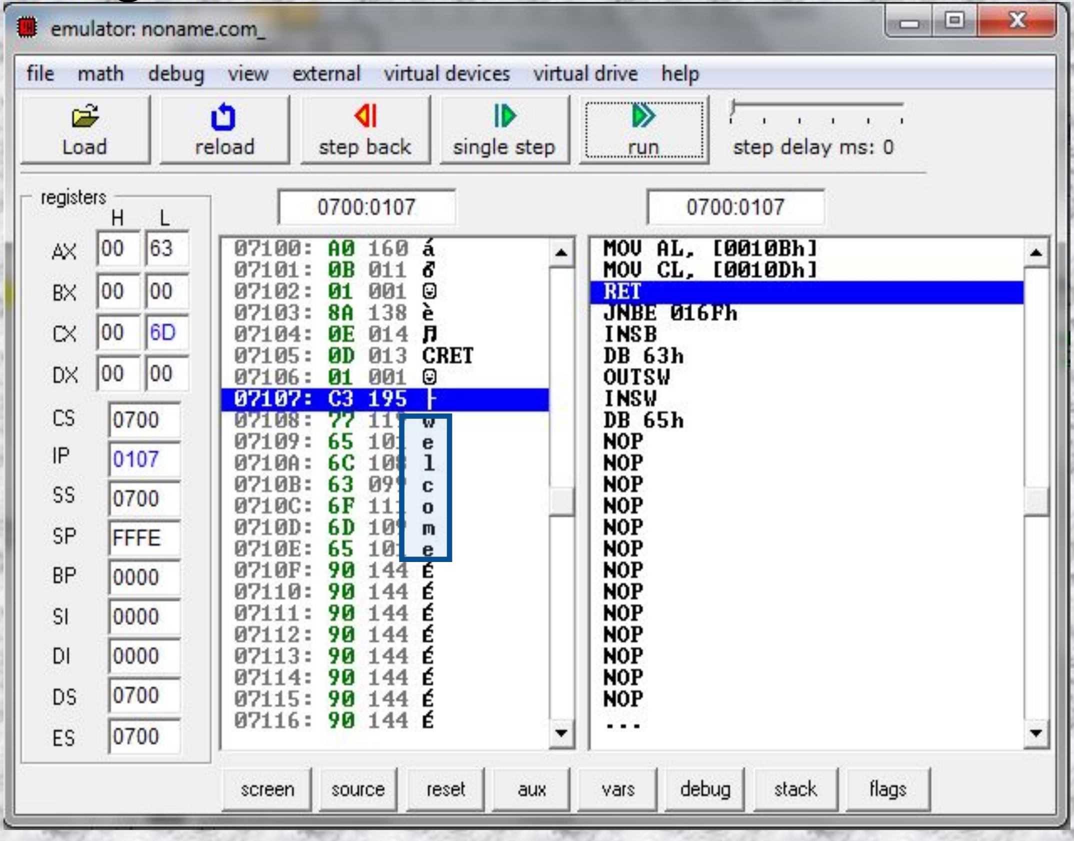
- String multiple lines:

```
greeting2 DB "Welcome to the Encryption Demo program "
          DB "created by Kip Irvine.",0dh,0ah,0
```

Example

Write assembly code that define firstly a variable take the value of “Wel**c**o**m**e” and then put the character ‘c’ in register AL and the character ‘m’ in register CL

```
Org 100h
Mov  al, a[3]
Mov  cl, a[5]
Ret
A db “welcome”
```



	H	L
AX	00	63
BX	00	00
CX	00	6D
DX	00	00

CS	0700
IP	0107
SS	0700
SP	FFFE
BP	0000
SI	0000
DI	0000
DS	0700
ES	0700

```
0700:0107

07100: A0 160 á
07101: 0B 011 ♂
07102: 01 001 ☺
07103: 8A 138 è
07104: 0E 014 ♫
07105: 0D 013 CRET
07106: 01 001 ☺
07107: C3 195 ├
07108: 77 119 w
07109: 65 101 e
0710A: 6C 108 l
0710B: 63 099 c
0710C: 6F 111 o
0710D: 6D 109 m
0710E: 65 101 e
0710F: 90 144 É
07110: 90 144 É
07111: 90 144 É
07112: 90 144 É
07113: 90 144 É
07114: 90 144 É
07115: 90 144 É
07116: 90 144 É
```

```
0700:0107

MOV AL, [0010Bh]
MOV CL, [0010Dh]
RET
JNBE 016Fh
INSB
DB 63h
OUTSW
INSW
DB 65h
NOP
NOP
NOP
NOP
NOP
NOP
NOP
NOP
NOP
NOP
NOP
NOP
NOP
...
```

Using the DUP Operator

If you need to declare a large array you can use **DUP (Duplicate)** operator. The syntax for **DUP**:

number DUP (value(s))

number - number of duplicate to make (any constant value).
value - expression that DUP will duplicate.

Ex:

1. c DB 5 DUP(9) ; 5 bytes , all equal to 9

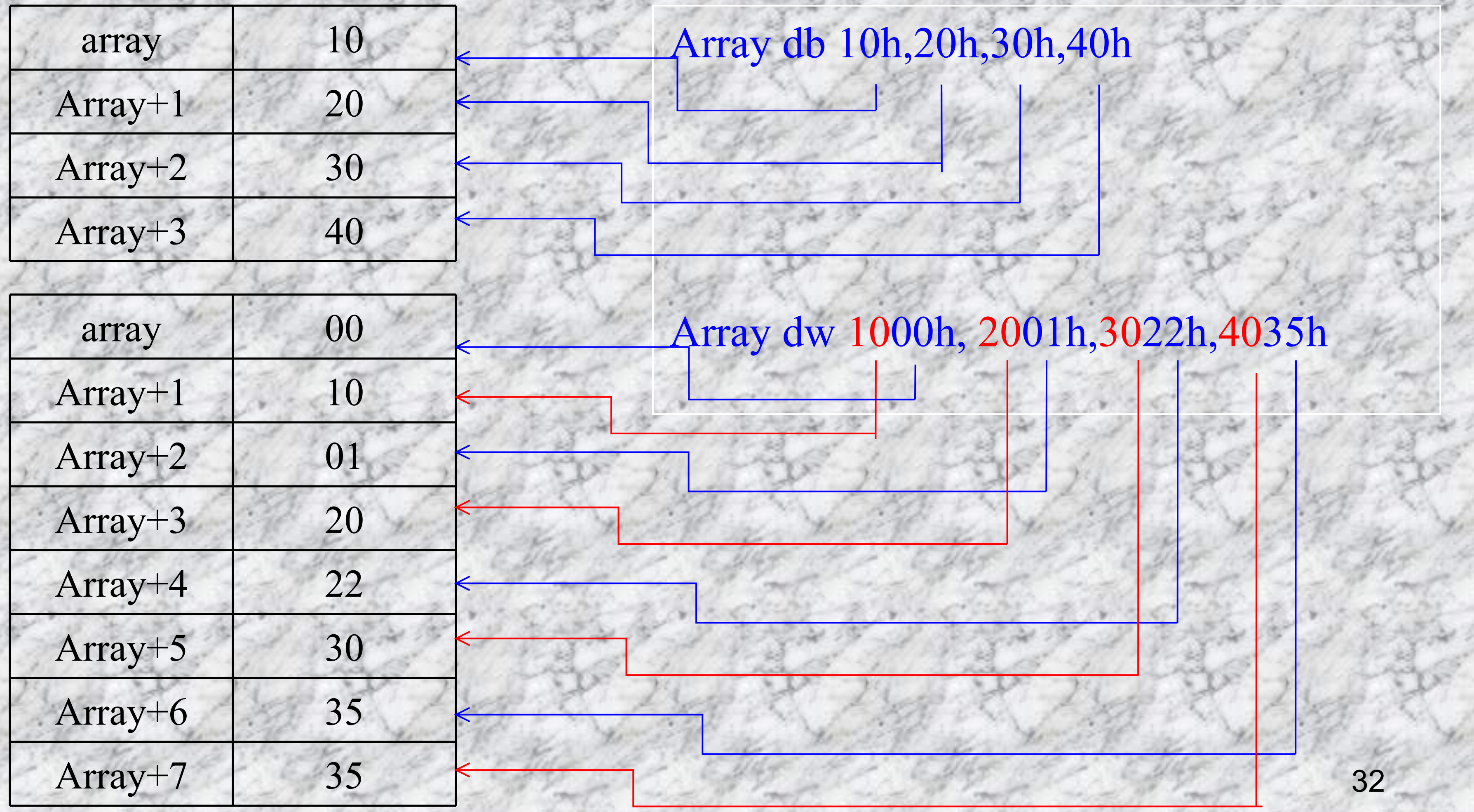
is an alternative way of declaring: c DB 9, 9, 9, 9, 9

2. d DB 5 DUP(1, 2)

is an alternative way of declaring: d DB 1, 2, 1, 2, 1, 2, 1, 2, 1, 2

3. DB 20 DUP(?) ; 20 bytes, uninitialize

Array



array	10
Array+1	20
Array+2	30
Array+3	40

Array db 10h,20h,30h,40h

array	00
Array+1	10
Array+2	01
Array+3	20
Array+4	22
Array+5	30
Array+6	35
Array+7	35

Array dw 1000h, 2001h,3022h,4035h

Array

	H	L
AX	00	10
BX	00	20

- If you define array as

Array db 10h,20h,30h,40h

Mov al,array ; move the first byte in the array

Mov bl,[array+1] ; access the second byte in the array by adding 1 to the offset of array

Or Mov bl, array+1

	H	L
AX	10	00
BX	20	01
CX	30	22

- If you define array as

Array dw 1000h, 2001h,3022h,4035h

Mov ax,array

Mov bx,[array+2] ; the offset of each array element is two bytes beyond the previous one.

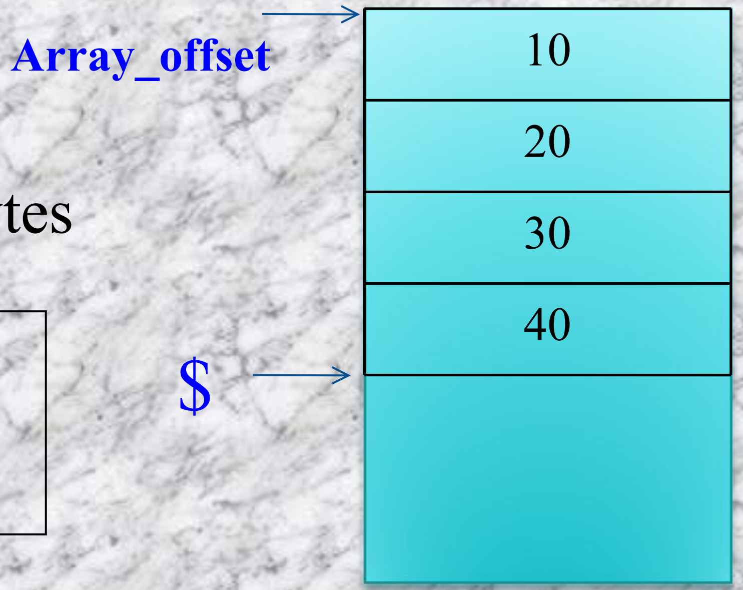
Mov cx,[array+4]

Calculating the Size of a Byte Array

Current location counter: $

- subtract address of list
- difference is the number of bytes

```
Array db 10,20,30,40
ArraySize = ($ - Array)
```

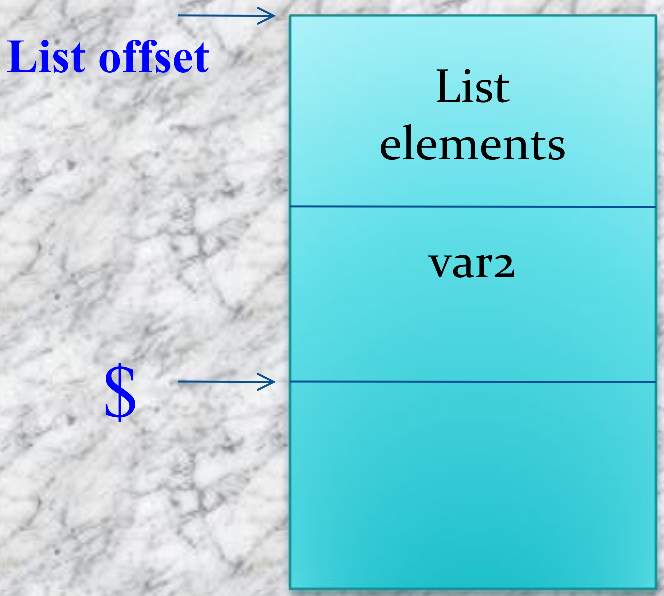


It is important for Array Size to follow immediately after list

Calculating the Size of a Byte Array cont…

The following, for example, would produce too large a value for ListSize because of the storage used by var2:

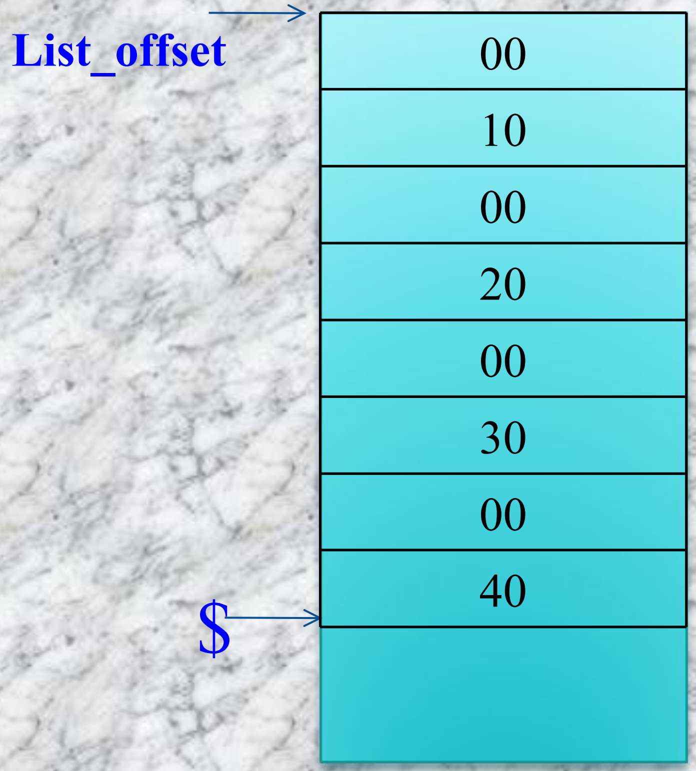
```
list db 10 , 20 ,30 , 40
var2 db 20 DUP(5 )
List Size = ($ - list )
```



Calculating the Size of a Word Array

Divide total number of bytes by 2 (the size of a word)

```
list dw 1000h,2000h,3000h,4000h
ListSize = ($ - list) / 2
```



List_offset

00

10

00

20

00

30

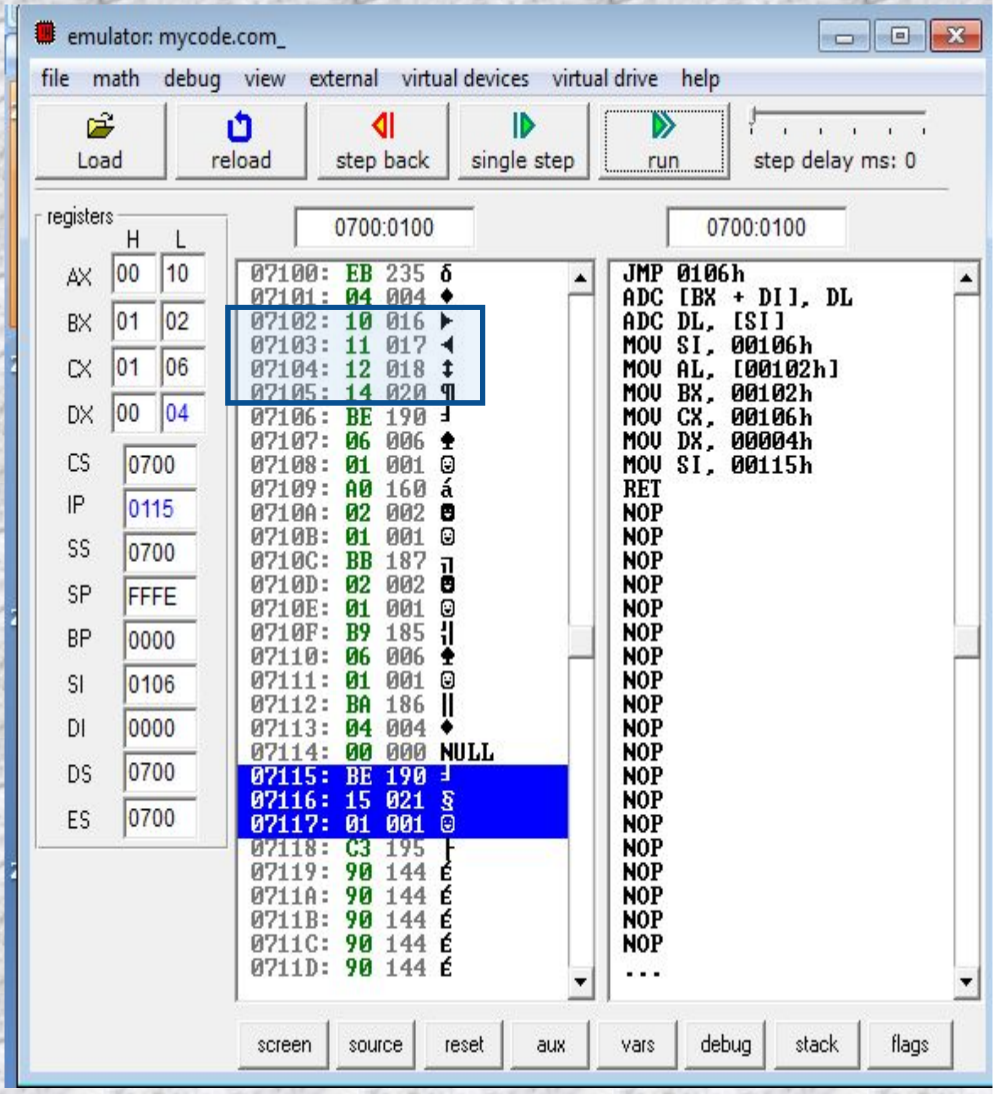
00

40

$

Example:

```
org 100h
.data
        Array db 10h,11h,12h,14h
        currentLoc=$
        ArraySize=($-Array)
.code
        mov si,$                    ; si=106
        mov al, Array               ; al=10h
        lea bx, Array               ; bx=102
        mov cx,currentLoc           ; cx=106
        mov dx, ArraySize           ;dx=4 ➔ $- Array =106-102=4
        mov si,$                    ; si=115
ret
```



```
registers
     H   L
AX   00  10
BX   01  02
CX   01  06
DX   00  04
CS   0700
IP   0115
SS   0700
SP   FFFE
BP   0000
SI   0106
DI   0000
DS   0700
ES   0700
```

```
0700:0100                    0700:0100

07100: EB 235 δ              JMP 0106h
07101: 04 004 ♦              ADC [BX + DI], DL
07102: 10 016 ►              ADC DL, [SI]
07103: 11 017 ◄              MOV SI, 00106h
07104: 12 018 ↕              MOV AL, [00102h]
07105: 14 020 ¶              MOV BX, 00102h
07106: BE 190 ╛              MOV CX, 00106h
07107: 06 006 ♠              MOV DX, 00004h
07108: 01 001 ☺              MOV SI, 00115h
07109: A0 160 á              RET
0710A: 02 002 ☻              NOP
0710B: 01 001 ☺              NOP
0710C: BB 187 ╗              NOP
0710D: 02 002 ☻              NOP
0710E: 01 001 ☺              NOP
0710F: B9 185 ╣              NOP
07110: 06 006 ♠              NOP
07111: 01 001 ☺              NOP
07112: BA 186 ║              NOP
07113: 04 004 ♦              NOP
07114: 00 000 NULL           NOP
07115: BE 190 ╛              NOP
07116: 15 021 §              NOP
07117: 01 001 ☺              NOP
07118: C3 195 ├              NOP
07119: 90 144 É              NOP
0711A: 90 144 É              NOP
0711B: 90 144 É              NOP
0711C: 90 144 É              NOP
0711D: 90 144 É              ...
```

Constants

- Constants are just like variables, but they exist only until your program is compiled (assembled). After definition of a constant its value cannot be changed.
- To define constants **EQU** directive is used:

  name **EQU** < any expression >

- For example:

```
k EQU 5

MOV AX, k
```

- The above example is functionally identical to code:

```
MOV AX, 5
```

Equal-Sign Directive

- The equal-sign directive associates a symbol name with an integer expression
- The syntax is:

```
   name = expression
```

- Example

```
  COUNT = 500
  mov ax,COUNT
```

- it generates and assembles the following statement:

```
   mov ax,500
```

Assignment 2

1. Write a constant expression that divides 10 by 3 and returns the integer remainder.
2. Write an assembly program to find the sum of 12 numbers stored in array the sum will be stored to variable sum . All the numbers will be less than 18h
3. write a program that subtracts three integers using only 16-bit registers.
4. Write a program that defines symbolic constants for all of the days of the week . Create an array variable that uses the symbols as initializers.

Thank You