

ASSEMBLY - ARRAYS

http://www.tutorialspoint.com/assembly_programming/assembly_arrays.htm

Copyright © tutorialspoint.com

We have already discussed that the data definition directives to the assembler are used for allocating storage for variables. The variable could also be initialized with some specific value. The initialized value could be specified in hexadecimal, decimal or binary form.

For example, we can define a word variable 'months' in either of the following way –

```
MONTHS DW 12
MONTHS DW 0CH
MONTHS DW 0110B
```

The data definition directives can also be used for defining a one-dimensional array. Let us define a one-dimensional array of numbers.

```
NUMBERS DW 34, 45, 56, 67, 75, 89
```

The above definition declares an array of six words each initialized with the numbers 34, 45, 56, 67, 75, 89. This allocates $2 \times 6 = 12$ bytes of consecutive memory space. The symbolic address of the first number will be NUMBERS and that of the second number will be NUMBERS + 2 and so on.

Let us take up another example. You can define an array named inventory of size 8, and initialize all the values with zero, as –

```
INVENTORY DW 0
           DW 0
           DW 0
           DW 0
           DW 0
           DW 0
           DW 0
           DW 0
```

Which can be abbreviated as –

```
INVENTORY DW 0, 0, 0, 0, 0, 0, 0, 0
```

The TIMES directive can also be used for multiple initializations to the same value. Using TIMES, the INVENTORY array can be defined as:

```
INVENTORY TIMES 8 DW 0
```

Example

The following example demonstrates the above concepts by defining a 3-element array x, which stores three values: 2, 3 and 4. It adds the values in the array and displays the sum 9 –

```
section .text
    global _start ;must be declared for linker (ld)

_start:

    mov     eax,3      ;number bytes to be summed
    mov     ebx,0      ;EBX will store the sum
    mov     ecx,x      ;ECX will point to the current element to be summed

top:   add     ebx,[ecx]

    add     ecx,1      ;move pointer to next element
    dec     eax        ;decrement counter
    jnz     top        ;if counter not 0, then loop again
```

```

done:

    add    ebx, '0'
    mov    [sum], ebx ;done, store result in "sum"

display:

    mov    edx, 1      ;message length
    mov    ecx, sum    ;message to write
    mov    ebx, 1      ;file descriptor (stdout)
    mov    eax, 4      ;system call number (sys_write)
    int    0x80        ;call kernel

    mov    eax, 1      ;system call number (sys_exit)
    int    0x80        ;call kernel

section .data
global x
x:
    db    2
    db    4
    db    3

sum:
    db    0

```

When the above code is compiled and executed, it produces the following result –

9