BRAC UNIVERSITY

Inspiring Excellence

Department of Computer Science and Engineering

Course Code: CSE341	**Credits: 1.5**
Course Name: Microprocessors	**Semester: Fall'18**

Lab 09
Arrays and Addressing Modes

I. Topic Overview:

This lab will make students acquainted with how one-dimensional arrays are declared in assembly language along with the operations they are used in. The importance of arrays in assembly language is that, many a time we need to use a collection of values as a group rather than using only a single value in a variable/register. The advantage of using an array to store the data is that a single name can be given to the whole array structure and an element can be accessed by providing an index. Lastly, this lab will provide students with the knowledge of how array elements can be accessed, by using different addressing modes.

II. Lesson Fit:

In order to do the lab with ease, the student must have completed:

a. Lab 1 'Introduction'

b. Lab 2 'Registers, Arithmetic Operations, Instruction: MOV'

c. Lab 3 'Basic I/O, Advanced Arithmetic Operations and Flags'

III. Learning Outcome:

After this lecture, the students will be able to:

a. Have a clear knowledge on arrays and their manipulation in assembly language

b. Know about the different addressing modes and when and how to use them

c. Get hands-on knowledge on incorporating the instructions learned in the previous labs with one-dimensional arrays.

IV. Anticipated Challenges and Possible Solutions

a. Problems with using DB and DW pseudo-ops in declaring byte and word arrays

Solutions:

i. Have a clear knowledge on both operations via examples and practice

V. Acceptance and Evaluation

Students will show the problems to the instructor one by one after completion. Those who won't be able to finish the assigned tasks in time will show them in the next class. There will be a short viva for the students who will show the finished tasks on the next day to check if they completed the tasks by themselves. A deduction of 30% will be there for late submission. The marks distribution is as follows:

Code: 50%

Viva: 50%

VI. Activity Detail

a. Hour: 1

Discussion: Arrays

An array is an ordered list of elements. By "ordered" we mean that there is a first element, second element, third element and so on. There are two pseudo-ops to declare byte and word arrays namely, DB and DW respectively.

i. Array declaration:

In java, which is a high level programming language, initialization of an array can be done by writing the following code:

int [ ] x = {10,20,30}

This line of code creates an integer array 'x' having elements 10, 20 and 30, where the element at the first index is 10, at the second index the element is 20 and 30 at the third and last index respectively.

In assembly language, this line of code for initializing an array can be substituted by writing:

x DB 10,20,30

if it's a byte array

OR,

x DW 10,20,30

if it's a word array

The difference between DB and DW in declaring arrays can be understood by the following example:

Suppose we have two arrays *B* and *W*, where, *B* is a byte array and *W* is a word array, each containing four integers namely, 10, 20, 30 and 40 respectively. We initialize both the arrays by writing the following code:

B DB 10,20,30,40

W DW 10,20,30,40

The address of the array variable is called the *base address* of the array. If the offset address assigned to *B* is 0200h, the array *B* looks like this in memory:

Offset Address	**Symbolic Address**	**Decimal Content**
0200h	B	10
0201h	B+1h	20
0202h	B+2h	30
0203h	B+3h	40

On the other hand, if the array *W* is assigned to the same offset address, i.e. 0200h, *W* will look something like this:

Offset Address	**Symbolic Address**	**Decimal Content**
0200h	W	10
0202h	W+2h	20
0204h	W+4h	30
0206h	W+6h	40

Therefore, in a byte array, the indices are incremented by 1 to access the next element, because each of the elements requires a space of 1 byte or 8 bits, whereas, in a word array, indices are incremented by 2 to get the next element in the array because each element holds a space equivalent to 1 word or 2 bytes or 16 bits.

ii. Array declaration using The DUP operator

It is possible to declare arrays whose elements share a common initial value by using the *DUP* operator. For example,

GAMMA DW 100 DUP(0)

Sets up an array of 100 words with each entry initialized to 0

DELTA DW 212 DUP(?)

Creates an array of 212 uninitialized bytes

iii. Accessing elements of an array

In many applications, we need to perform some operations on each element of an array. For example, suppose array *A* is a 10 element array and we want to add the elements. In a high level programming language, we could do something like this:

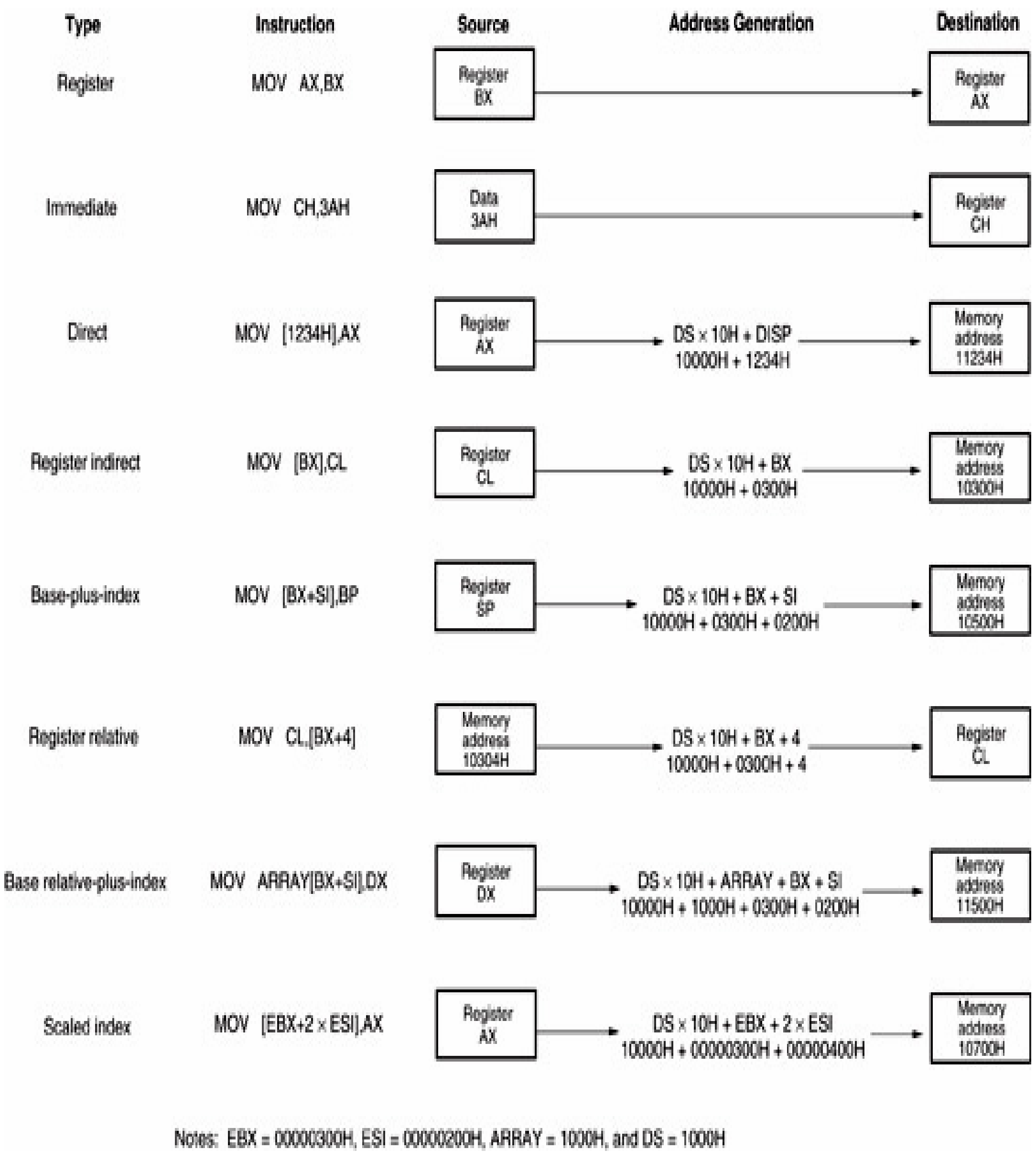
```
sum = 0
N = 1
Repeat
  sum = sum + A[N]
  N = N + 1
UNTIL N >  10
```

To code this in assembly language, we need a way to move from one array element to the next one. In the next section, we will see how to accomplish this using indirect addressing.

b. Hour: 2

Discussion: Addressing Modes

The way an operand is specified is known as its addressing mode. There are a total of 7 such modes 2 of which address registers and data values (register and immediate modes) and rest of the 5 modes address memory operands indirectly. Each of the instructions involving addressing modes is formulated with 'MOV' instruction.



i. **Register addressing mode:** In this addressing mode, both the source and the destination operands are registers.

MOV AX, 12

MOV BX, AX

ii. **Immediate addressing mode:** In this mode, the source operand is a byte/word data and the destination operand is a register.

MOV AL, 22H

iii. **Direct addressing mode:** In this addressing mode, one of the operands is a memory location.

MOV AX, [0200H]

iv. **Register Indirect addressing mode:**

Example:

Write some code to print the elements of the 5 element array W defined by

a DW 1,2,3,4,5

Solution:

JAVA

```
int [] a = {1,2,3,4,5};
for (int i = 0;i<a.length;i++)
{
    System.out.println(a[i]);
}
```

ASSEMBLY

```
.data
a dw 1,2,3,4,5
.code
 mov cx,5
mov ah,2
mov si,0
start:
mov dl,a[si]
int 21h
add si,2
loop start
```

v. **Based addressing mode:**

In these mode, the operand's offset address is obtained by adding a number called a displacement to the contents of a register. Displacement may be any or the following:

- the offset address or a variable
- a constant (positive or negative)
- the offset address or a variable plus or minus a constant

If A is a variable, examples of displacements are:

- A (offset address of a variable)
- -2 (constant)
- A + 4 (offset address of a variable plus a constant)

The syntax of an operand is any of the following equivalent expressions:

- [register + displacement]
- [displacement + register]

- [register] + displacement
- displacement + [register]
- displacement [register]

The register must be BX, BP, SI, or Dl.

If BX, SI, or DI is used, DS contains the segment number of the operand's address.

If BP is used, SS has the segment number.

- ✓ The addressing mode is called **based** if BX (base register) or BP (base pointer) is used.

For example, suppose W is a word array, and BX contains 4. In the instruction MOV AX, W[BX], the displacement is the offset address of variable W. The instruction moves the element at address W + 4 to AX. This is the third element in the array.

The instruction could also have been written in any of these forms:

MOV AX, [W + BX]

MOV AX, [BX + W]

MOV AX, W + [BX]

MOV AX, [BX] + W

vi. Indexed addressing mode:

The addressing mode is called **indexed**, if SI (source index) or DI (destination index) is used. ***The remaining things are quite similar to based addressing mode.***

As another example, suppose SI contains the address of a word array W. In the instruction MOV AX, [SI + 2] the displacement is 2. The instruction moves the contents of W + 2 to AX.

This is the second element in the array. The instruction could also have been written in any of these forms:

MOV AX, [2 + SI]

MOV AX, 2 + [SI]

MOV AX, [SI] +2

MOV AX, 2[Sl]

vii. **Based-Indexed addressing mode:**

In this mode the offset address of the operand is the sum of

1. **the contents of a base register (BX or Bl')**
2. **the contents of an index register (SJ or DI)**
3. **optionally, a variable's offset address**
4. **optionally, a constant (positive or negative)**

If BX is used, DS contains the segment number of the operand's address; if BP is used, SS has the segment number. The operand may be written several ways; four of them are

1. **variable [base_register] [index_register]**
2. **[base_register + index_register + variable + constant]**
3. **variable [base_register + index_register + constant]**
4. **constant [base_register + index_register + variable)**

The order of terms within these brackets is arbitrary.

For example, suppose W Is a word variable, BX contains 2, and SI contains 4. The instruction MOV AX, W[BX][SI] moves the contents of W+2+4 = W+6 to AX. This instruction could also have been written in either of these ways:

MOV AX, [W+BX+SI] or MOV AX, W [BX+SI]

Based indexed mode especially useful for processing two-dimensional arrays.

c. **Hour: 3**

Check progress while students carry out the following tasks.

Problem Task: Tasks 01-09

VII. **Home Tasks:** Tasks 10- 11

Lab 9 Activity List

Task 01

Find the sum of the elements of a *BYTE* array of size 10 by taking input from user

Task 02

Find the sum of the elements of a *WORD* array of size 10 by taking input from user

Task 03

Take an array of size 10 from user and print the elements in reverse order

Task 04

Take two arrays each of size 10 and copy the elements of the first array to the second array and print the second array

Task 05

Take two arrays each of size 10 and copy the elements of the first array to the second array *IN REVERSE ORDER* and print the second array

Task 06

Take two arrays of size 5. Then add the first element of the first array with the last element of the second array, then the second element of the first array with the second-last element of the second array and so on. Store the results of each of the addition operations between the two arrays in a third array in a sequential manner and print the third array

Task 07

Write a code to display the maximum and minimum elements of a given array of size 10 where input is provided by the user.

Task 08

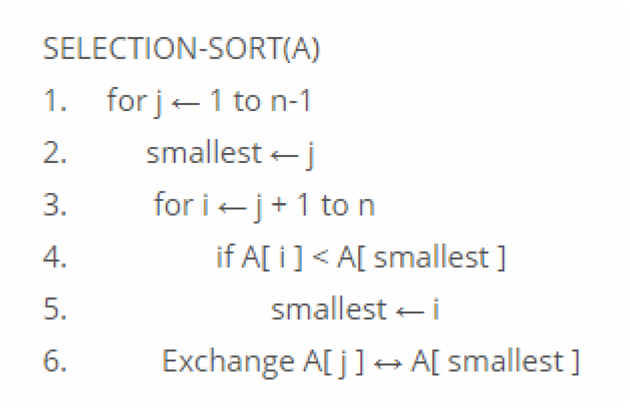
Take an array from the user of size 10 and print how many even numbers are present in the array as well as all the even numbers of the array.

Task 09

Take an array from the user of size 10 and print how many odd numbers are present in the array as well as all the odd numbers of the array.

Task 10 (Home Task)

Sort the elements of a given array using selection-sort algorithm. For your convenience, pseudo-code of the algorithm is provided below:



```
SELECTION-SORT(A)
1.   for j ← 1 to n-1
2.       smallest ← j
3.       for i ← j + 1 to n
4.           if A[ i ] < A[ smallest ]
5.               smallest ← i
6.       Exchange A[ j ] ↔ A[ smallest ]
```

Task 11 (Home Task)

Sort the elements of a given array using bubble-sort algorithm. For your convenience, pseudo-code of the algorithm is provided below:

```
swapped = true
while swapped
  swapped = false
  for j from 0 to N - 1
     if a[j] > a[j + 1]
        swap( a[j], a[j + 1] )
        swapped = true
```

References:

1. Dhanvani, P. (2013, February 17). *Selection Sort | Pseudo Code of Selection Sort | Selection Sort in Data Structure.* Retrieved from https://freefeast.info/general-it-articles/selection-sort-pseudo-code-of-selection-sort-selection-sort-in-data-structure/
2. Bubble sort. (n.d.). In *Wikipedia.* Retrieved October 18, 2018, from http://www.algorithmist.com/index.php/Bubble_sort
3. Ytha, Y., & Marut, C. (1992). *Assembly Language Programming and Organization of the IBM PC.* Singapore, McGraw-Hill.