National University of Computer and Emerging Sciences



**Laboratory Manual**

*for*

*Computer Organization and Assembly Language*

Course Instructors

Lab Instructor(s)

Section

Semester

**Department of Computer Science**

# COAL Lab 5 Manual

|  |
| --- |
| **Objectives:**     * Zero/Sign Extension of Integers, MOVZX, MOVSX * Status Flags * Data related Operators & Directives * Problems & Assignments |

## 5.1 Zero Extension of Integers

|  |  |  |  |
| --- | --- | --- | --- |
| **Pseudo-op** | **Explanation** | **Syntax** | **Example** |
| **MOVZX** | Move with zero-extend | **MOVZX reg32, reg16/mem16**  **MOVZX reg32, reg8/mem8**  **MOVZX reg16, reg8/mem8** | **MOVZX eax,bx**  **MOVZX eax,bl**  **MOVZX ax,bl** |

## 5.2 Sign Extension of Integers

|  |  |  |  |
| --- | --- | --- | --- |
| **Pseudo-op** | **Explanation** | **Syntax** | **Example** |
| **MOVSX** | Move with sign-extend | **Same as above** | **Same as above** |

**5.3 Flags Register:**

Flags Register determines the current state of the processor. They are modified automatically by CPU after mathematical operations, this allows to determine the type of the result, and to determine conditions to transfer control to other parts of the program. Generally you cannot access these registers directly. Two types of flags register are available.

1. Status Flags: CF, PF, AF, ZF, SF, OF
2. Control Flags: TF, IF, DF

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|  |  |  |  | O | **D** | **I** | **T** | S | Z |  | A |  | P |  | C |

The individual flags are explained in the following table.

|  |  |  |
| --- | --- | --- |
| CY | Carry Flag | This flag is set to 1 when there is an ***unsigned overflow***. |
| PE | Parity Flag | This flag is set to 1 when there is ***even number of one bits*** in result. |
| AC | Auxiliary Carry Flag | During addition or subtraction if a carry goes from one nibble to the next this flag is set. |
| ZR | Zero Flag | It is set to 1 when last mathematical or logical instruction has ***produced a zero in its destination***. |
| PL | Sign Flag | It is set to 1 when ***result is negative***. |
| **TF** | **Trap Flag** | Used for on-chip debugging. |
| **EI** | **Interrupt Flag** | When this flag is set to 1 CPU reacts to interrupts from external devices. |
| **UP** | **Direction Flag** | This flag is used by some instructions to process data chains, when this flag is set to 0 - the processing is done forward, when this flag is set to 1 the processing is done backward. |
| OV | Overflow Flag | It is set to 1 when there is a ***signed overflow***. |

## 5.4 Data Related Operators & Directives

* OFFSET Operator, discussed in previous lab.
* PTR Operator, discussed in previous lab.
* **TYPE** Operator

|  |  |  |  |
| --- | --- | --- | --- |
| **Pseudo-op** | **Explanation** | **Syntax** | **Example** |
| **TYPE** | Returns size in bytes of a single elements of a variable | **MOV reg, TYPE mem** | **.data**  **Var1 BYTE ?**  **Var2 DWORD ?**  **.code**  **MOV AX, TYPE VAR2**  **; AX = 0004** |

* **LENGTHOF** Operator

|  |  |  |  |
| --- | --- | --- | --- |
| **Pseudo-op** | **Explanation** | **Syntax** | **Example** |
| **LENGTHOF** | Counts the number of elements in an array | **MOV reg, LENGTHOF mem** | **.data**  **Var1 BYTE 10, 20, 30**  **Var2 DWORD 5 DUP (3 DUP (?)) .code**  **MOV AX, LENGTHOF VAR2 ; AX = 000F** |

* **SIZEOF** Operator

|  |  |  |  |
| --- | --- | --- | --- |
| **Pseudo-op** | **Explanation** | **Syntax** | **Example** |
| **SIZEOF** | Returns  LEGNTHOF\*TYPE | **MOV reg, SIZEOF mem** | **.data**  **Var1 DWORD 10H, 20H, 30H**  **Var2 DWORD 5 DUP (3 DUP (?)) .code**  **MOV AX, SIZEOF VAR1 ; AX = 000C** |

# Problem(s) / Assignment(s)

|  |  |
| --- | --- |
| **Discussion & Practice** | **Estimated completion time: 1 hr, 30 mins** |
|  |  |
| **Problem 5.1:** *Monitoring**Status Flag* **Estimated completion time:15 mins** | |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Sr. No.** | **Instructions** |  | **Fl** | **ags** |  |
| **CY** | **OV** | **PL** | **ZR** |
| 1. | **MOV** **AX,710H** | NA | NA | NA | NA |
| 2. | **SUB AH,0F3H** | 0 | 0 | 0 | 1 |
| 3. | **MOV** **AX,510H** | 1 | 0 | 0 | 0 |
| 4. | **SUB AX,0E4H** | 1 | 0 | 0 | 0 |
| 5. | **MOVZX** **EBX**,**AX** | 0 | 0 | 0 | 0 |
| 6. | **MOV** **BL,-127** | 0 | 0 | 0 | 0 |
| 7. | **MOVSX** **EAX**,**BL** | 0 | 0 | 0 | 0 |
| 8. | **DEC BL** | 0 | 0 | 0 | 0 |
| 9. | **MOV** **AX,60H** | 0 | 0 | 1 | 0 |
| 10. | **SUB AH,0F6H** | 0 | 0 | 1 | 0 |

|  |
| --- |
| **Problem 5.2:** *Array Manipulation* **Estimated completion time:20 mins**  Let us have an array,  A\_array WORD 10FFH, 6120H, 0ABCDH, 818H  Compute the sum for higher and lower bytes of each value, Store result of each addition in AX, BX, CX and DX registers and display them on console. Observe CF, OF, ZF and SF during each addition. You may Ignore the carry in each addition.  Hint: AX = 10+FF  Solution:  include Irvine32.inc  .data  A\_array WORD 10FFH, 6323H, 0ABCDH, 828H  var1 word ?  var2 word ?  .code  main proc  mov eax,0  mov ax,A\_array  mov ebx,0  movzx bx,al  movzx ax,ah  add ax,bx  mov ebx,0  mov bx,A\_array+2  mov ecx,0  movzx cx,bl  movzx bx,bh  add bx,cx  mov ecx,0  mov cx,A\_array+4  mov edx,0  movzx dx,cl  movzx cx,ch  add cx,dx  mov edx,0  mov dx,A\_array+6  mov var1,cx  mov ecx,0  movzx cx,dl  movzx dx,dh  add dx,cx  mov cx,var1  exit  main endp  end main |

|  |  |
| --- | --- |
| **Problem 5.3:** *Array Manipulation*  Repeat Problem 5.2 with alteration of   1. Putting the result in a 5 elements DWORD size array namely B\_array. 2. Keeping Status Flag intact at the end of the program and store it as 5th element of B\_array.   Also display   1. Type of B\_array in AH. 2. No. of characters of B\_array in BH. 3. Size of B\_array in BL.   Solution:  include Irvine32.inc  .data  var1 word ?  A\_array WORD 10FFH, 6323H, 0ABCDH, 828H  B\_array dword 5 dup(?)  .code  main proc  mov eax,0  mov ax,A\_array  mov ebx,0  movzx bx,al  movzx ax,ah  add ax,bx  mov ebx,0  mov bx,A\_array+2  mov ecx,0  movzx cx,bl  movzx bx,bh  add bx,cx  mov ecx,0  mov cx,A\_array+4  mov edx,0  movzx dx,cl  movzx cx,ch  add cx,dx  mov edx,0  mov dx,A\_array+6  mov var1,cx  mov ecx,0  movzx cx,dl  movzx dx,dh  add dx,cx  mov cx,var1  mov B\_array,eax  mov B\_array+2,ebx  mov B\_array+4,ecx  mov B\_array+6,edx  mov ah ,type B\_array  mov bh ,lengthof B\_array  mov bl ,sizeof B\_array  mov eax,0  LAHF  mov B\_array+16,eax  exit  main endp  end main | **Estimated completion time:20 mins** |

**You are done with your exercise(s), make your submission** ☺

***Submission Guidelines:***

➢ Attach your .asm files of all programs and examples