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 6 Manual

|  |
| --- |
| **Objectives:**     * Indirect Addressing o Indirect operands o Indexed operands * Pointers * Using DumpMEM * Problems & Assignments |

## 6.1 Indirect Addressing

|  |  |
| --- | --- |
| **Protected Mode** | **Real Mode** |
| 1. Indirect operand can be any 32-bit generalpurpose register (**EAX**, **EBX**, **ECX**, **EDX**, **ESI**, **EDI**, **EBP**, and **ESP**) surrounded by brackets. 2. Example  **include irvine32.inc**   **.data byteVal BYTE 10h**  **.code Main proc**  **mov ESI,OFFSET byteVal mov AL,[ESI] ;AL = 10h** | 1. Indirect operand can be any 16-bit general-purpose register (**AX**, **BX**, **CX**, **DX**, **SI**, **DI**, **BP**, and **SP**) surrounded by brackets.      1. Example **include irvine16.inc**   **.data byteVal BYTE 10h**  **.code Main proc startup**  **mov SI,OFFSET byteVal mov AL,[SI] ;AL = 10h** |

The size of an operand may not be evident from the context of an instruction. The following instruction causes the assembler to generate an “operand must have size” error message:

**inc** [**esi**] ; **error: operand must have size**

The PTR operator confirms the operand size:

**inc BYTE PTR [esi]**

## 6.2 Arrays Manipulation

|  |  |  |
| --- | --- | --- |
| **Using Indirect Operands** | **Using Indexed Operands** | |
| **.data**  **arrayW WORD 1000h,2000h,3000h**    **.code**  **mov esi,OFFSET arrayW mov ax,[esi] ; AX = 1000h add esi,2 mov ax,[esi] ; AX = 2000h add esi,1 mov ax,[esi]**  **; AX = ???** | **.code**  **mov esi,0**  **mov ax,[arrayW+esi] ; AX = 1000h add esi,2 mov ax,[arrayW+esi] ; AX = 2000h add esi,1 mov ax,[arrayW+esi]** | ***Using Displacement***        **.code**  **mov esi,OFFSET arrayW mov ax,[esi]**  **; AX = 1000h mov ax,[esi+2]**  **; AX = 2000h mov ax,[esi+4]**  **; AX = 3000h** |

## 6.3 DumpMEM

The ***DumpMem*** procedure writes a range of memory to the console window in hexadecimal. Pass it the starting address in ESI, the number of units in ECX, and the unit size in EBX (1= byte, 2= word, 4=doubleword). The following sample call displays an array of 11 doublewords in hexadecimal:

**.data**

**array DWORD 1,2,3,4,5,6,7,8,9,0Ah,0Bh**

**.code main PROC**

|  |  |
| --- | --- |
| **mov esi,OFFSET array** | **; starting OFFSET** |
| **mov ecx,LENGTHOF array** | **; number of units** |
| **mov ebx,TYPE array call DumpMem** | **; doubleword format** |

**The following output is produced:**

00000001 00000002 00000003 00000004 00000005 00000006 00000007

00000008 00000009 0000000A 0000000B

# Problem(s) / Assignment(s)

|  |  |  |
| --- | --- | --- |
| **Discussion & Practice** |  | **Estimated completion time: 1 hr, 30 mins** |

|  |  |  |  |
| --- | --- | --- | --- |
| **Problem 6.1:** *Fill the requested Register*  **myBytes BYTE 10h,20h,30h,40h myWords WORD 1Ah,2Bh,3Ch,4Dh,5Eh myDoubles DWORD 2,4,6,8,5 myPointer DWORD myDoubles** |  |  | **Estimated completion time:20 mins** |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Instructions** |  | | | **Registers** |
| mov esi, OFFSET myBytes |  | | |  |
| mov al, [esi] | **a. AL =** | 9010 | | |
| mov al, [esi+3] | **b. AL =** | 9040 | | |
| mov esi, OFFSET myWords + 2 |  | | |  |
| mov ax, [esi] | **c. AX =** 003B | | |  |
| mov edi, 8 |  | | |  |
| mov edx, [myDoubles + edi] | **d. EDX =** | | 00000003 | |
| mov edx, myDoubles[edi] | **e. EDX =** | | 00000003 | |
| mov ebx, myPointer |  | | |  |
| mov eax, [ebx+4] | **f. EAX =** | | 00000002 | |
| mov eax,DWORD PTR myWords | **g. EAX =** | | 003B008A | |
| mov esi,myPointer |  | | |  |
| mov ax,[esi+2] | **h. AX =** | | 003B | |
| mov ax,[esi+6] | **i. AX =** | | 003B | |
| mov ax,[esi-4] | **j. AX =** | | 003B | |

|  |
| --- |
| **Problem 6.2 (a):** *Fibonacci number generation* **Estimated completion time:15 mins**  Write a program that generate the first six Fibonacci number sequence (1, 1, 2, 3, 5, 8). Use an array named Fibonacci of word type. You may initialize the first two places of array with 1’s and next four places with 0. Use the following rule and save results in the same array by **using indirect operand addressing**.  The rule to generate sequence is Fn= Fn-1+Fn-2. Call DumpMEM to display Fibonacci sequence. |

**(b):**

Repeat a part using indexed operands without displacement.

Solution:

include Irvine32.inc

.data

Fibonacci word 1, 1 , 4 dup (?)

.code

main proc

mov esi ,offset Fibonacci

mov eax,0

mov eax,[esi]

add eax,[esi+2]

mov [esi+4],eax

mov eax,0

mov eax,[esi+2]

add eax,[esi+4]

mov [esi+6],eax

mov eax,0

mov eax,[esi+4]

add eax,[esi+6]

mov [esi+8],eax

mov eax,0

mov eax,[esi+6]

add eax,[esi+8]

mov [esi+10],eax

mov esi,OFFSET Fibonacci

mov ecx,LENGTHOF Fibonacci

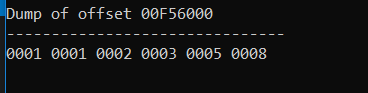
mov ebx,TYPE Fibonacci

call DumpMem

exit

main endp

end main



|  |
| --- |
| **Problem 6.3:** *Array Manipulation* **Estimated completion time:20 mins**  Let us have an array,  A\_array BYTE 1FH, 63H, 0ABH, 88H   1. Compute the sum for higher and lower bytes of each value (by ignoring carry bit). 2. Putting the result in another 4 elements DWORD size array namely B\_array. 3. Use indexed operands addressing with displacement.   Display A\_array and B\_array on console.  Solution:  include Irvine32.inc  .data  A\_array WORD 10FFH, 6323H, 0ABCDH, 828H  B\_array dword 4 Dup(0)  var1 word ?  .code  main proc  mov eax,0  mov ebx,0  mov esi,offset A\_array  mov ax,[esi]  mov bl,ah  mov ah,0  add ax,bx  mov ebx,0  mov ecx,0  mov bx,[esi+2]  mov cl,bh  mov bh,0  add bx,cx  mov ecx,0  mov edx,0  mov cx,[esi+4]  mov dl,ch  mov ch,0  add cx,dx  mov edx , 0  mov dx,[esi+6]  mov var1,0  mov var1,ax  mov eax,0  mov al,dh  mov dh,0  add dx,ax  mov eax,0  mov ax,var1  mov esi,0  mov esi ,offset B\_array  mov [esi],eax  mov [esi+4],ebx  mov [esi+8],ecx  mov [esi+12],edx  mov esi,OFFSET B\_array  mov ecx,LENGTHOF B\_array  mov ebx,TYPE B\_array  call DumpMem  mov esi,OFFSET A\_array  mov ecx,LENGTHOF A\_array  mov ebx,TYPE A\_array  call DumpMem  exit  main endp  end main |

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