EL213: Computer Org. & Assembly Language Lab

# Lab#05

## Agenda

* Operators
  + OFFSET Operator
  + PTR Operator
  + TYPE Operator
  + LENGTHOF Operator
  + SIZEOF Operator
  + LABEL Operator
* Indexed Operands
* Pointers
* JMP Instruction
* LOOP Instruction

## Operators

Operators and directives are not part of Intel instruction Set but only interpreted by the assemblers. MASM has many effective directives as effective tools for describing and addressing variables.

### OFFSET Operator

OFFSET operator returns the distance of a label from the beginning of the data segment in bytes corresponding the relative modes (Real/Protected). In the following example, it is supposed that data segment’s starting address is 00404000

*; copy this code into .data section as variable declaration section*

*bVal BYTE ?*

*wVal WORD ?*

*dVal DWORD ?*

*dVal2 DWORD ?*

*dValArray DWORD 5 DUP(?)*

*; copy this code into main procedure*

*mov esi,OFFSET bVal ; ESI = 00404000*

*call DumpRegs*

*mov esi,OFFSET wVal ; ESI = 00404001*

*call DumpRegs*

*mov esi,OFFSET dVal ; ESI = 00404003*

*call DumpRegs*

*mov esi,OFFSET dVal2 ; ESI = 00404007*

*call DumpRegs*

*mov esi,OFFSET dValArray ; ESI = 0040400B*

*call DumpRegs*

The OFFSET operator can also be used with a direct – offset operand as in the following example.

*; copy this code into .data section as variable declaration section*

*dValArray DWORD 1,2,3,4,5*

*; copy this code into main procedure*

*mov esi,OFFSET dValArray ; ESI = 00404000*

*; moves the starting address of dValArray as it*

*; is same as address of data segment*

*; Note: it is also equivalent to the pointer in C/C++*

*mov eax,[esi]*

*call DumpRegs*

*mov esi,OFFSET dValArray + 8 ; adds 8 to starting of data*

*; segment and returns the sum in esi*

*mov eax,[esi]*

*call DumpRegs*

### PTR Operator

PTR Operator is used to override the default size of an operand. Also provides the facility to access part of variable.

*; copy this code into .data section as variable declaration section*

*myDouble DWORD 12345678h*

*myBytes BYTE 12h,34h,56h,78h*

*; copy this code into main procedure*

*mov ax,WORD PTR myDouble ; getting least significant portion of myDouble*

*call DumpRegs*

*mov al,BYTE PTR myDouble ; AL = 78h*

*mov al,BYTE PTR [myDouble+1] ; AL = 56h*

*mov al,BYTE PTR [myDouble+2] ; AL = 34h*

*mov ax,WORD PTR myDouble ; AX = 5678h*

*mov ax,WORD PTR [myDouble+2] ; AX = 1234h*

*mov ax,WORD PTR [myBytes] ; AX = 3412h*

*mov ax,WORD PTR [myBytes+2] ; AX = 7856h*

*mov eax,DWORD PTR myBytes ; EAX = 78563412h*

*; call DumpRegs where you want to print the content s of registers*

### TYPE Operator

Used to get the size of any variable/declaration.

*; copy this code or declare these variables into .data section as variable declaration section*

*var1 BYTE ?*

*var2 WORD ?*

*var3 DWORD ?*

*var4 QWORD ?*

*; copy/type this code into main procedure*

*mov eax,TYPE var1 ; 1*

*mov eax,TYPE var2 ; 2*

*mov eax,TYPE var3 ; 4*

*mov eax,TYPE var4 ; 8*

*; call DumpRegs where you want to print the contents of registers*

### LENGTHOF Operator

This operator counts the number of elements in a single data declaration like arrays.

*; copy this code or declare these variables into .data section as variable declaration section*

*byte1 BYTE 10,20,30*

*array1 WORD 30 DUP(?),0,0*

*array2 WORD 5 DUP(3 DUP(?))*

*array3 DWORD 1,2,3,4*

*digitStr BYTE "12345678",0*

*; copy/type this code into main procedure*

*mov eax,LENGTHOF byte1 ; 3*

*mov ebx,LENGTHOF array1 ; 32*

*mov ecx,LENGTHOF array2 ; 15*

*mov edx,LENGTHOF array3 ; 4*

*mov esi,LENGTHOF digitStr ;9*

*call DumpRegs*

*; call DumpRegs where you want to print the content s of registers*

### SIZEOF Operator

This operator returns a value that is equivalent to multiplying LENGTHOF by TYPE.

*; copy this code or declare these variables into .data section as variable declaration section*

*byte1 BYTE 10,20,30*

*array1 WORD 30 DUP(?),0,0*

*array2 WORD 5 DUP(3 DUP(?))*

*array3 DWORD 1,2,3,4*

*digitStr BYTE "12345678",0*

*; copy/type this code into main procedure*

*mov eax,SIZEOF byte1 ; 3 \* 1 = 3*

*mov ebx,SIZEOF array1 ; 32 \* 2 = 64*

*mov ecx,SIZEOF array2 ; 15 \* 2 = 30*

*mov edx,SIZEOF array3 ; 4 \* 4 = 16*

*mov esi,SIZEOF digitStr ; 9 \* 1 = 9*

*call DumpRegs*

*; call DumpRegs where you want to print the content s of registers*

### LABEL Operator

This operator assigns an alternate label name and type to an existing storage location but it does not allocate any storage of its own. It is just like an operator which can read or write the data of its specified length.

*; copy this code or declare these variables into .data section as variable declaration section*

*dwList LABEL DWORD*

*wordList LABEL WORD*

*intList BYTE 00h,10h,00h,20h*

*; copy/type this code into main procedure*

*XOR EAX,EAX*

*XOR EBX,EBX*

*XOR ECX,ECX*

*mov eax,dwList ; 20001000h*

*mov bx,wordList ; 1000h*

*mov cl,intList ; 00h*

*call DumpRegs*

*; call DumpRegs where you want to print the content s of registers*

## Indexed Operands

An Indexed operand adds a constant to a register to generate an effective address. Any of the 32-bit general purpose registers may be used as index registers.

There are different notational forms permitted by MASM.

Label [reg]

[label + reg]

.data

arrayW WORD 1000h,2000h,3000h

.code

mov esi,0

mov ax,[arrayW + esi] ; AX = 1000h

mov ax,arrayW[esi] ; alternate format

add esi,2

add ax,[arrayW + esi]

## Pointers

A variable that contains the address of another variable is called a pointer variable.

.data

arrayW WORD 1000h,2000h,3000h

ptrW DWORD arrayW ; ptrW DWORD OFFSET arrayW

.code

mov esi,ptrW

mov ax,[esi] ; AX = 1000h

INCLUDE Irvine32.inc

; Create user-defined types.

PBYTE TYPEDEF PTR BYTE ; pointer to bytes

PWORD TYPEDEF PTR WORD ; pointer to words

PDWORD TYPEDEF PTR DWORD ; pointer to doublewords

.data

arrayB BYTE 10h,20h,30h

arrayW WORD 1,2,3

arrayD DWORD 4,5,6

; Create some pointer variables.

ptr1 PBYTE arrayB

ptr2 PWORD arrayW

ptr3 PDWORD arrayD

.code

main PROC

; Use the pointers to access data.

mov esi,ptr1

mov al,[esi] ; 10h

mov esi,ptr2

mov ax,[esi] ; 1

mov esi,ptr3

mov eax,[esi] ; 4

exit

main ENDP

END main

## JMP Instruction

The JMP instruction causes an unconditional transfer to a target location inside the code segment. The location must be identified by a code label, which is translated by the assembler into an offset.

**Syntax**

JMP targetlabel

When the CPU executes this instruction, the offset of the target label is moved into the instruction pointer, causing the execution to immediately continue at the new location.

**For example**

*top:*

*.*

*.*

*Jmp top ;loop*

## LOOP Instruction

The LOOP instruction provides a simple way to repeat a block of statements a specific number of times. ECX is automatically used as a counter and is decremented each time the loop repeats.

**Syntax**

LOOP destination

Two steps:

1. ECX 🡨 ECX – 1
2. if ECX != 0, jump to target

*mov ax, 0*

*mov ecx,5*

*L1:*

*inc ax*

*loop L1*

*INCLUDE Irvine32.inc*

*.data*

*array WORD 100h,200h,300h,400h*

*.code*

*main PROC*

*mov edi,OFFSET array ; address of array*

*mov ecx,LENGTHOF array ; loop counter*

*mov ax,0 ; zero the accumulator*

*L1:*

*add ax,[edi]*

*add edi,TYPE array ; point to next value*

*loop L1 ; repeat until ECX = 0*

*exit*

*main ENDP*

*END main*

**What about nested loops?**