Microprocessors (0630371)

Fall 2010/2011 – Lecture Notes # 13

Memory Addressing Modes

Objectives of the Lecture

- **Indirect Addressing**
  - **Protected mode**
  - **Real-Addressing mode**
  - **Array Sum Examples**
- **Indexed Addressing**
- **Pointers**
- **Programming Examples**

Indirect Addressing

- Indirect Addressing is very important for array and data structure processing.
- Use a register as a pointer when processing the pieces of an array, the operand in this case is called **Indirect Operand**.

Indirect Operands - Protected Mode

- Any general-purpose 32-bit register **eax, ebx, ecx, edx, esi, edi, ebp, esp** surrounded by **brackets.**
- The register in this case contains the address of some data.

```
.data
val1 BYTE 10h
.code
mov esi,OFFSET val1
mov al,[esi]   ;moves 10h into al, in other words
               ;[esi] points to the data by containing
               ;the offset of the data from the start
               ;of the data segment
```

Indirect Operands - Real-address Mode

- 16-bit register (either **si, di** or **bx**) holds the offset of a variable

```
.data
val1 BYTE 10h
.code
main proc
     startup
     mov si,OFFSET val1
     mov al,[si]    ;moves 10h into al
```

- Note: Use 16-bit registers si, di, bx as indexed operands in real-address mode

General Protection Fault (GP Fault)

- If the effective address in protected mode points to the area outside your data segment, the GP Fault will be generated.
- Make sure indirect register has a value within your data segment (must be initialized)

Example:

If ESI were uninitialized, the instruction

```
mov ax,[esi]
```

will generate a GP fault

Using PTR with Indirect Operands

```
inc [esi]
```

Causes problems since assembler doesn’t know if **esi** points to a byte, word, dword, etc.

```
inc BYTE PTR [esi];
```

To make operand size clear

Example:

```
.data
myCount WORD 0
.code
mov esi,OFFSET myCount
inc [esi] ; error: ambiguous
inc WORD PTR [esi]  ; ok
```

Indirect Addressing with Arrays

- Use indirect operand to point to each array element

Example 1:

```
.data
val1 BYTE 10h,20h,30h
.code
mov esi,OFFSET val1
mov al,[esi]   ; dereference ESI (AL = 10h)
inc esi
mov al,[esi]   ; AL = 20h
inc esi
mov al,[esi]   ; AL = 30h
```

Example 2:

```
.data
arrayW WORD 1000h,2000h,3000h
.code
mov esi,OFFSET arrayW
mov ax,[esi]
add esi,2 ; or: add esi,TYPE arrayW
add ax,[esi]
add esi,2
add ax,[esi]   ; AX = sum of the array
```

Example 3: Add up the 3 elements at eax

```
.data
arrayD DWORD 10h,20h,40h
.code
mov esi,OFFSET arrayD
mov eax,[esi]
add esi,4
add eax,[esi]
add esi,4
add eax,[esi]
```

Indexed Operand

- use register as an index to be added to the start of a memory address
- An indexed operand adds a constant to a register to generate an effective address. There are two notational forms:

```
memory[reg]          or
```

```
     [memory + register]
```

Example 1

```
     .data
     arrayB BYTE 10h,20h,30h
     mov esi,0
     mov al,[arrayB + esi]    ;to move  10h into al
```

Example 2

```
     .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]
          etc.
```

Example 3: Adding Displacement

```
     .data
     arayW WORD 10h,20h,30h
     .code
     mov esi,OFFSET arrayW
     mov ax,[esi]             moves 10h to ax
     mov ax,[esi + 2]         moves 20h to ax
     mov ax,[esi + 4]         moves 30h to ax
```

Example 4: Index Scaling Factor

You can scale an indirect or indexed operand to the offset of an array element. This is done by multiplying the index by the array's **TYPE**:

```
     .data
     arrayB BYTE  0,1,2,3,4,5
     arrayW WORD  0,1,2,3,4,5
     arrayD DWORD 0,1,2,3,4,5
     .code
     mov esi,4
     mov al,arrayB[esi*TYPE arrayB]          ; 04
     mov bx,arrayW[esi*TYPE arrayW]          ; 0004
     mov edx,arrayD[esi*TYPE arrayD]    ; 00000004
```

Pointers

- Variable that contains the address of another variable
- Pointers are essential when manipulating arrays and other data structures in memory
- **Pointer Types**

Pointer Type	16-bit real address mode	32-bit protected mode
NEAR	16-bit offset from the beginning of the data segment	32-bit offset from the beginning of the data segment
FAR	32-bit segment offset address	48-bit segment selector-offset address

- We'll use **Near Pointers in protected mode**

Example 1:

```
.data
arrayB BYTE 10h,20h,30h,40h
arrayW WORD 1000h,2000h,3000h,4000h
ptrB DWORD arrayB
ptrW DWORD arrayW      pointers contain the offsets
```

or

```
ptrB DWORD OFFSET arrayB
prtW DWORD OFFSET arrayW;
```

Example 2:

```
.data
arrayW WORD 1000h,2000h,3000h
ptrW DWORD arrayW
.code
     mov esi,ptrW
     mov ax,[esi]   ; AX = 1000h
```

Using TYPEDEF

- To create user-defined types.
- The TYPEDEF operator creates a **user-defined type**.
- A **user-defined type** has the status of a built-in type when defining variables.
- TYPEDEF is ideal for creating pointer variables.
- Syntax :

```
     name TYPEDEF type
```

- For example, **PBYTE** and **PWORD** are pointers to bytes and words, respectively:

```
PBYTE TYPEDEF PTR BYTE
PWORD TYPEDEF PTR WORD
```

then

```
.data
arrayB BYTE 10h,20h,30h,40h
ptr1   PBYTE ?
ptr2   PBYTE arrayB
```

Programming Examples

Example 1:

TITLE Pointers (Pointers.asm)

```
INCLUDE Irvine32.inc
PBYTE  TYPEDEF PTR BYTE
PWORD  TYPEDEF PTR WORD
PDWORD TYPEDEF PTR DWORD
.data
arrayB BYTE   10h,20h,30h
arrayW WORD   1,2,3
arrayD DWORD  4,5,6
ptr1   PBYTE  arrayB
ptr2   PWORD  arrayW
ptr3   PDWORD arrayD
.code
```

```
     main PROC
          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
```

Example 2:

TITLE Scaling an Array Index (Scale.asm)

```
     INCLUDE Irvine32.inc
     .data
     arrayB BYTE  0,1,2,3,4,5
     arrayW WORD  0,1,2,3,4,5
     arrayD DWORD 0,1,2,3,4,5
     .code
     main PROC
          mov  esi,4
          mov  al,arrayB[esi*TYPE arrayB]    ; 04
          mov  bx,arrayW[esi*TYPE arrayW]    ; 0004
          mov  edx,arrayD[esi*TYPE arrayD]   ; 00000004
          call DumpRegs
          exit
     main ENDP
     END main
```