Indirect Addressing

- Indirect Operands
- Array Sum Example
- Indexed Operands
- Pointers

Indirect Operands [1/2]

An indirect operand holds the address of a variable, usually an array or string. It can be dereferenced (just like a pointer).

```
.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
```

Indirect Operands [2/2]

Use PTR when the size of a memory operand is ambiguous.

```
.data
myCount WORD 0

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

Array Sum Example

Indirect operands are ideal for traversing an array. Note that the register in brackets must be incremented by a value that matches the array type.

```
.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                   ; increment ESI by 2
    add ax,[esi]                ; AX = sum of the array
```

ToDo: Modify this example for an array of doublewords.

Indexed Operands

An indexed operand adds a constant to a register to generate an effective address. There are two notational forms:

[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]
   etc.
```

Pointers

You can declare a pointer variable that contains the offset of another variable.

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

TYPEDEF Operator (User-defined Type)

```
TITLE Pointers                          (Pointers.asm)
INCLUDE Irvin32.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
    moc eax,[esi]                        ; 4
main ENDP
END main
```

JMP and LOOP Instructions

- JMP Instruction (Unconditional Transfer)
- LOOP Instruction (Conditional Transfer)
- LOOP Example
- Summing an Integer Array
- Copying a String

JMP Instruction

- JMP is an unconditional jump to a label that is usually within the same procedure.
- Syntax: JMP *target*
- Logic: EIP ← *target*
- Example:

```
top:
    .
    .
    jmp top
```

A jump outside the current procedure must be to a special type of label called a global label (see Section 5.5.2.3 for details).

LOOP Instruction

- The LOOP instruction creates a counting loop
- Syntax: LOOP *target*
- Logic:
  - ECX ← ECX – 1
  - if ECX > 0, jump to *target*
- Implementation:
  - The assembler calculates the distance, in bytes, between the current location and the offset of the target label. It is called the relative offset.
  - The relative offset is added to EIP.

LOOP Example

The following loop calculates the sum of the integers 5 + 4 + 3 +2 + 1:

```
offset      machine code          source code
00000000    66 B8 0000            mov   ax,0
00000004    B9 00000005           mov   ecx,5

00000009    66 03 C1          L1: add   ax,cx
0000000C    E2 FB                 loop  L1
0000000E
```

When LOOP is assembled, the current location = 0000000E. Looking at the LOOP machine code, we see that –5 (FBh) is added to the current location, causing a jump to location 00000009:

00000009 ← 0000000E + FB

Your turn . . .

If the relative offset is encoded in a single byte,

(a) what is the largest possible backward jump?

(b) what is the largest possible forward jump?

(a) –128

(b) +127

Your turn . . .

What will be the final value of AX?

10

```
    mov ax,6
    mov ecx,4
L1:
    inc ax
    loop L1
```

How many times will the loop execute?

4,294,967,296

```
    mov ecx,0
X2:
    inc ax
    loop X2
```

Table 1-4 Ranges of Unsigned Integers.

Storage Type	Range (low–high)	Powers of 2
Unsigned byte	0 to 255	0 to $(2^8 - 1)$
Unsigned word	0 to 65,535	0 to $(2^{16} - 1)$
Unsigned doubleword	0 to 4,294,967,295	0 to $(2^{32} - 1)$
Unsigned quadword	0 to 18,446,744,073,709,551,615	0 to $(2^{64} - 1)$

Nested Loop

If you need to code a loop within a loop, you must save the outer loop counter's ECX value. In the following example, the outer loop executes 100 times, and the inner loop 20 times.

```
.data
count DWORD ?
.code
   mov ecx,100          ; set outer loop count
L1:
   mov count,ecx        ; save outer loop count
   mov ecx,20           ; set inner loop count
L2:.
   .
   loop L2              ; repeat the inner loop
   mov ecx,count        ; restore outer loop count
   loop L1              ; repeat the outer loop
```

Summing an Integer Array

The following code calculates the sum of an array of 16-bit integers.

```
.data
intarray WORD 100h,200h,300h,400h
.code
    mov edi,OFFSET intarray       ; address of intarray
    mov ecx,LENGTHOF intarray     ; loop counter
    mov ax,0                      ; zero the accumulator
L1:
    add ax,[edi]                  ; add an integer
    add edi,TYPE intarray         ; point to next integer
    loop L1                       ; repeat until ECX = 0
```

Your turn . . .

What changes would you make to the program on the previous slide if you were summing a doubleword array?

Copying a String

The following code copies a string from source to target.

```
.data
source  BYTE  "This is the source string",0
target  BYTE  SIZEOF source DUP(0)

.code
    mov  esi,0                    ; index register
    mov  ecx,SIZEOF source        ; loop counter
L1:
    mov  al,source[esi]           ; get char from source
    mov  target[esi],al           ; store it in the target
    inc  esi                      ; move to next character
    loop L1                       ; repeat for entire string
```