Microprocessors (0630371)
Fall 2010/2011 – Lecture Notes # 14

Control Transfer Instructions

Outline of the Lecture

- **JMP Instruction**
- **LOOP Instruction**
- **LOOP Example**
- **Nested Loop**
- **Programming Examples**

The control transfer instructions control the flow of program execution.

Types of Transfer

- Unconditional - Go somewhere
- Conditional - Go based on **ecx or cx** registers or flags

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
```

; Loop will continue endlessly unless we find a way to terminate it.

- A jump outside the current procedure must be to a special type of label called a **global** label
- JMP causes the modification of the EIP register

LOOP Instruction

- The **LOOP** instruction creates a counting loop
- **ECX** register is used as a counter to count the iterations in protected mode for **LOOP** Instruction and **CX** is used for real-address mode.
- The **LOOPD** instruction uses **ECX** as the loop counter.
- The **LOOPW** instruction use **CX** as the loop counter.
- **Syntax**: `LOOP target`
- **Logic**:
  - `ECX ← ECX – 1`
  - `if ECX != 0, jump to target`
- **Implementation**:
  - The assembler calculates the distance, in bytes, between the offset of the following instruction and the offset of the target label. It is called the relative offset.
  - The relative offset is added to EIP.

- A **common programming error** is to initialize ECX or CX to zero before beginning a loop, In this case the loop instruction decrements ECX to FFFFFFFFh or CX to FFFF and it repeats 4,294,967,296 times for ECX or 65,536 for CX.

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 (offset of the next instruction). –5 (FBh) is added to the the current location, causing a jump to location 00000009:

```
00000009 ← 0000000E + FB
```

- Note: Loop destination must be within -128 to +127 bytes of the current location counter, else MASM error.
- If the relative offset is encoded in a single signed byte,
  a. What is the largest possible backward jump?
  b. What is the largest possible forward jump?

Answer

```
a) -128
b) +127
```

- If you modify ECX inside a loop, the loop instruction may not work as expected, for example

```
top:
     .
     .
     inc ecx
     loop top ; this loop will never stop
```

- If you need to modify ECX inside a loop, you can save it in a variable at the beginning of the loop and restore it before the loop instruction, for example

```
.data
count DWORD ?
.code
     mov ecx,100    ; set loop count
L1:
     mov count,ecx  ; save the count
     .
     .
     mov ecx, 30    ; modify ECX
     .
     .
     mov ecx,count  ; restore outer loop count
     loop L1
```

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
```

Programming Examples

Example 1: Summing an Integer Array
The following code calculates the sum of an array of 16-bit integers.

Steps

1. Assign the array's address to a register that will serveas an indexed operand
2. Set ecx to the number of elements in the array
3. Assign zero to the register that accumulates the sum
4. Creat a label to mark the start of the loop
5. Use indirect addressing to add one element to the accumulator
6. Set the index register forward to the next element
7. Use loop to jump to the label

```
.data
intarray WORD 100h,200h,300h,400h
main PROC
.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
main ENDP
END main
```

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

Example 2: 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); Good use of SIZEOF
main PROC
.code
     mov  esi,0               ; index register
                              ;ESI is used to index source
                              ; and target strings
     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
main ENDP
END main
```

- Rewrite the program shown in the previous slide, using indirect addressing rather than indexed addressing.