Lecture 10

- Jump
- Loop
- Homework 3
- Outputting prompts
- Reading single characters

Transfer of Control

- The CPU loads and executes programs sequentially.
- You'd like to be able to implement if statements, gotos, and loops.
- These involve *transfer of control*, where the order of statement execution is modified.
- Two types:
  - Unconditional Transfer – the program branches to a new location in *all* cases – execution continues at the new address
  - Conditional Transfer – the program branches is a certain condition is true (like the conditional jumps)

JMP

- JMP provides *unconditional transfer*.

```
JMP address      ; unconditional xfer
                 ;to address
                 ;(IP gets a new
                 ; value)
```

address is a user-defined label

- In machine code, to JMP to an address which is calculated by adding the displacement in the instruction to the value in IP *or*, for longer jumps, replacing the value in IP (and CS if it's a very long jump).
- In assembly, the assembler replaces the user labels with the offset during assembly (much easier!).

JMP Formats

- Three basic forms of (direct) JMP instructions:

  JMP SHORT destination

  JMP NEAR PTR destination

  JMP FAR PTR destination
- There are different machine code formats for each of these.

JMP SHORT

- If the target label (address) is within –128 to 127 locations (bytes) of the *instruction following* the JMP (remember, the offset is added to the current value of the IP, which is pointing to the *next* instruction), it is assembled as a SHORT instruction (2 bytes). Only eight bits are needed to specify the address (these eight bits are added to the IP)

Example

```
Offset   Machine Code Source Code
0100     B4 02     start:  mov ah, 2  ;loop start
0102     B2 41             mov  dl, 'A' ;
0104     CD 21             int     21h     ;disp A
0106     EB  F8            jmp     start   ;jmp back
0108     ….. (rest of program)
```

How does it know it's a SHORT jump?
You're jumping back so the assembler already knows the distance!

Symbol Table

Symbol	Value
start	0100

When assembling, the assembler uses a location counter to keep track of where it is. When "jmp start" is assembled, the location counter = 0108
To compute displacement:
0100 – 0108 = -8 = F8

JMP NEAR PTR

- What if the jump in the example had been forward?
  - If the JMP was used without the SHORT qualifier it would have assumed to be JMP NEAR PTR.
- NEAR PTR indicates that the label can be anywhere in the current code segment.
- A 16-bit displacement is *moved* to the IP.

JMP FAR PTR

- FAR PTR allows you to jump outside of your current code segment.
- The label's segment address is moved to CS, and its offset is moved to IP.
- This is a 5-byte instruction:
  - 1 for opcode
  - 2 for displacement (-> IP)
  - 2 for segment (->CS)

LOOP

- LOOP is the easiest way to do iteration.
- It's like a for-loop – it's for count-controlled loops only where the number of repetitions are known before the loop is entered.
- What is it good for:
  - execute a loop 12 times (for example)
  - repeat instructions x times
- What it isn't good for:
  - keep looping until the user types enter

LOOP, continued

- Format:

  LOOP destination
- What loop does:

  CX is the loop counter

  LOOP subtracts one from CX

  If CX is NOT equal to zero, control transfers to destination
- Example:

```
    mov cx, 5       ;cx = 5
    mov ax, 0       ;ax = 0
start:
    add ax, 1
    loop start              ;jump to start

after loop: ax = 5, cx = 0
```

Another LOOP example

- In C:

```
ax = 0;
for (i = 23; i >= 1; i--)
{
   ax = ax + bx;
}
```

- In Assembly:

```
        sub ax, ax          ;or mov ax, 0
        mov cx, 23
start: add ax, bx           ;ax = ax + bx
        loop  start         ;if cx >= 1,
                            ;jumps to start
```

Loop: Errors to Avoid!

- Starting with CX = 0
  - LOOP will decrement CX to FFFFh and the loop will repeat 65,535 times!!!
- Altering the loop counter
  - If CX (or CH or CL) is modified inside the loop then the loop will not behave the way you want it to! For example, if you increment CX then the loop will never stop.
- Also: Flags are not affected when LOOP decrements CX – even when CX goes to zero!

Conditional Loops

- The assembler also has conditional loops.
- These loops still decrement CX and terminate when it is zero but they check other flags as well.
- LOOPZ/LOOPE (loop if zero, loop if equal)
- LOOPNZ/LOOPNE (loop if not zero, loop if not equal)

LOOPZ/LOOPE

- LOOP while ZF = 1 and CX > 0
- Example, p. 203 in Irvine

LOOPNZ/LOOPNE

- Loop while ZF = 0 and CX > 0
- Example from p. 204, Irvine

Conditional Loop Pitfalls

- When you used an unconditional loop, you needed to be careful that you did not modify CX.
- You still need to be careful with CX but now you ALSO need to make sure you don't modify the flags unintentionally.
- Example p. 203, Irvine

In-Class Exercise

- Write assembly code that calculates the Fibonacci series: 1,1,2,3,5,8,13,… (Except for the first two numbers in the sequence, each number is the sum of the preceding two numbers). Use LOOP and set the limit for 12 iterations.

Outputting Prompts

- Friday's lecture will talk about MS-DOS function calls.
- You'll need to use one in your homework to output prompts to the user.

INT 21h

- INT 21h is a DOS function call (DOS services)
- You saw one example of this in hw2:

```
mov ax, 4c00h
int 21h          ;don't forget h!
```

- This returns to DOS from an executing program.
- The 4Ch that goes into AH tells DOS which function to perform (in this case, returning to DOS).

Outputting the Prompt

- For hw3:

```
  .data
prompt1 db  "Enter the month:  $"
  .code
  .startup
  ;prompt for first input
  mov ah, 09h
  mov dx, offset prompt1
  int 21h
```

- What is this doing?
  - 09h: String output function – writes a string to standard output
  - AH = 09, DX = offset of the string.
  - The string must be terminated by a $ (dollar sign character)

More on Output

- If you want to output a string, followed by a carriage return and line feed (crlf), you will need to put them in your string also!
- example, p. 148 of Irvine

Warning!

- The $ at the end of the string tells DOS when the string has ended.
- If you try to print out a string that includes a $, it will not work…(and you need to use a different method to output the $)
- If you forget the $… DOS will output all subsequent characters in memory until it finds the ASCII value for $ (24h). This could involve many screens worth of garbage output before it stops!

Reading in Characters

- 01h – filtered input with echo
  - waits for a single character to be entered (or, if one is in the input buffer already just grabs it)
  - stores it in AL
  - Input: ah = 1
  - Output: al = the character read

  (filtered? filters out control characters)

  (echo? it displays the character you as you type it. If you weren't using echo you would not be able to see what you typed!)

Example

- For hw 3, put 01h into ah, then use int 21h to read a character typed by the user:

```
mov     ah, 01h       ;get first
                      ;month digit
int     21h
```

- The character read in will be in AL. You'll need to convert it from an ASCII character to a decimal number:

```
sub     al, 30h       ;convert to
                      ;decimal
```

- Disadvantages of using 01h:
  - only one character at a time
  - it will read any character typed, as it is typed.
    - if you type the wrong character you can't backspace and correct it – it has already been read and processed by your program