COMP 2280 - Introduction to Computer Systems

Module 4- Branching and Control Statements

Introduction

- In this module we will learn
  - The LC-3 branching instructions,
  - The LC-3 condition codes,
  - How to implement basic control structures.

Condition Codes

- CPUs have bits that represent conditions that have occurred during the last arithmetic or logic instruction executed.
- These bits may indicate that the result was zero, negative, positive, overflow, carry, etc.
- Each of these bits is known as a *condition code*.
- LC-3 has three condition codes
  - N – negative
  - P – positive
  - Z – zero
- Condition codes can be used by branching/control instructions to change the order in which instructions are executed.

Condition Codes

- The instructions in LC-3 that affect the condition codes are: **add**, **and**, **ld** **(direct)**, **ldi** **(indirect)**, **ldr** **(relative/base-offset)**, and **not**.
- Each time one of these instructions is executed, the three condition codes are set/reset based on the result.
- eg)
  - AND R0,R0,#0 ; N=0,P=0,Z=1
  - ADD R0,R0,#-1 ; N=1,P=0,Z=0

Condition Codes

- Condition codes may be examined by some LC-3 instructions to determine what to do next.
- This is the idea of branching.
- We use this idea in High-Level-Languages all the time.

Example:

```
x = y + 10;
if (x > 0)    //a branch instr
        …
```

- In order to write useful assembly language programs, we must be able to implement control structures like those we have in HLL.
- Let's look at the LC-3 branching instructions that will allow us to do this.

Branching/Control Statements

- These statements are used to alter the flow of control within a program.
- This is done by changing the program counter (PC) register.
- Three types
  - conditional - branch based on result of last instruction
  - unconditional - branch always (ever heard of "goto"?)
  - trap - changes PC to execute an "OS service routine". Once trap finishes, returns to execute next instruction.

Conditional Branching

- Can be used to implement
  - if/else statement
  - while loop
  - for loop
  - do-while loop
  - and other looping structures
- The key when using conditional branches is the condition codes
  - n - negative
  - z - zero
  - p - positive

Conditional Branching

- The conditional branch instruction has the form

  brX label

  where X = n, z, p, nz, np, zp, or nzp
- eg) This example loops 10 times.

```
        AND R0,R0,#0
        ADD R0,R0,#10
loop
        ADD R0,R0,#-1
        BRP loop
```

Conditional Branching

- In order to use conditional branching, you usually do some operation that affects the condition codes and use the appropriate branch instruction
  - brn - branch if z = 0, n = 1, p = 0
  - brz - branch if z = 1, n = 0, p = 0
  - brp - branch if z = 0, n = 0, p = 1
  - brnz - branch if (z = 1 or n = 1) and (p = 0)
  - brnp - branch if (z = 0) and (n = 1 or p = 1)
  - brzp - branch if (z = 1 or p = 1) and (n = 0)
  - brnzp (same as br) - branch if n=1 or z=1 or p=1 (this is an unconditional branch)

Conditional Branching

- An Example of an If/Else statement (see if-else.asm).

```
;This example implements a simple if/else statement
          .orig x3000
          and R2,R2,#0            ;R2 <- 0
          and R1,R1,#0
          add R1,R1,#10           ;R1 <- 10
; if R1 > 0 R2++, R1-=2 else R2=R1, R1--
          brnz else               ;R1 <= 0 do else clause
if                                ;if ( R1 >  0) do if clause
          add R2,R2,#1
          add R1,R1,#-2
          br endif
else                              ;else (R1 <=0)
          add R2,R1,#0
          add R1,R1,#-1
endif                             ;end of if/else
          add R3,R1,R2
          halt
          .end
```

Conditional Branching

- An example of a while loop (see while.asm).
- Read a char and print it out 12 times.

```
;This example implements a simple while loop

        .orig x3000
        trap x20            ; read char into r0

        and R1,R1,#0
        add R1,R1,#12
loop
        brz done            ; while (R1 != 0)
        trap x21            ; display char in r0
        add R1,R1,#-1
        br  loop
done                        ; end of while loop
        halt
        .end
```

Conditional Branching

- An example of a do-while loop (see do-while.asm)

```
;This example implements a simple do-while loop

          .orig x3000
          trap x20            ;read char into r0

          and R1,R1,#0
          add R1,R1,#5        ;loop 5 times

          ;notice no test is done before entering the loop
loop
          trap x21            ;print character in r0
          add R1,R1,#-1
          brp loop
done                          ;end of while loop
          halt
          .end
```

Converting an if to a branch

- Given this if statement, which would be the equivalent branch?

```
x=x+x;
if(x == 0) { x++; }
else { x = 0; }
x+=2;
```

```
        ADD R0, R0, R0
        BRz if   ;if true, goto the if code
        BRnp else ; else, go to else code
if      ADD R0, R0, #1   ;if code
        BR endif  ;skip over the else
else    AND R0, R0, #0 ; else code
        BR endif ;jump to end of if
endif   ADD R0, R0, #2
```

Too much branching!

```
        ADD R0, R0, R0
        BRnp else ;if false, go to else
        ADD R0, R0, #1  ;if code
        BR endif ;skip over the else
else    AND R0, R0, #0 ; else code
endif   ADD R0, R0, #2
```

Much better!

Converting an if to a branch

- When choosing an if condition it's better to choose one that results in the least number of "else" branches.
- The reason for this is exactly what we saw in the last slide.
  - Branch if the condition is **false**, otherwise keep running the code as usual.
- Here is an example of a multi-condition if (easily written as a nested if):

```
ADD R3, R1,#-10  ;check if R1>10
BRNZ endif ;not >10, we skip
ADD R3, R2,#2  ;check if R2==-2
BRNP endif  ;was not -2, we skip
;do stuff here
;do stuff here
;do stuff here
endif
;last thing here
```

```
if(x>10){ //check if R1>10
        if(y==-2){ //check if R2==-2
            //do stuff here
            //do stuff here
            //do stuff here
        }
}
//last thing here
```

Conditional Branching

- Note:
  - branching is PC-relative, ie. target address is made by adding signed offset (IR[8:0]) to current PC. (More meaningful when we see addressing modes)
  - PC has already been incremented by FETCH stage.
  - Target must be within -255 and 256 words of BR instruction.
  - This is because we only have 9 bits to hold this offset.

Conditional Branching

- When a branch is not taken, the next sequential instruction is executed.
- Some instructions do not affect the CC. In that case, the CC is unchanged.

Unconditional Branching

- Assembler Inst.

```
JMP Rx
```

- Jump is an unconditional branch -- *always* taken
  - Target address is the contents of a register
  - Copies the register contents to the PC
  - Allows any target address
- E.g.:

```
         LD    R4,EXITPTR
         JMP   R4   ; target can be anywhere
EXITPTR  .fill EXIT
```

Or simply use:

```
BR    EXIT   ; target must be within range
```

Unconditional Branching

- Why have both a JMP and a BR?
  - JMP can go beyond the -255 to 256 distance, whereas BR can't.
  - JMP can jump to anywhere in memory.

Reading from Keyboard

- A character can be read from the keyboard using trap x20 (or GETC)
- The character is read into the low 8 bits of register R0.
- The top 8 bits of R0 is set to 0…0.

Printing to Console

- A trap can be used to print a string to the console.
- The trap # is x22. You can also use PUTS.
- The address of the string needs to be stored into R0 (use LEA instruction)

Eg)

```
    .orig x3000
    lea  R0,strHi ;get addr of string
    trap x22      ;print string
    …
    halt
strHi    .stringz "Hello\n";string
    .end
```