CS 2110 - Lab 8

LC-3 Assembly Programming - Conditional Branching and Examples

Wednesday, June 15, 2022

Lab Assignment: Assembly Quiz

1. Go to Quizzes on Canvas
2. Select Lab 08, password: **Branch**
3. Get 100% to get attendance!
   a) Unlimited attempts
   b) Collaboration is **allowed**!
   c) Ask your TAs for help :)

Homework 4

- Released!
- **Due Monday, June 20th at 11:59 PM**
- Files available on Canvas
- Submit on Gradescope (unlimited submissions)
- Will be demoed
- Please don't wait until the very last hours before the homework is due to ask for help!

Homework 4: Demos are Next Week!

- Sign up under Canvas Calendar—slots up by tonight at 9:00 PM
  - Sign up by **Friday at 11:59 PM** to be guaranteed a slot
- Each demo is about 10 minutes—please be on time!
- The demo is worth 50% of your Homework 4 grade
- If you miss your demo or cancel within 24 hours, you will not receive the 50 demo points
- If you can't make any available slots work, please email Shawn **by Friday at 11:59 PM**
- More details in the **Homework 4 Demo Logistics Canvas announcement**

Homework 5

- Covers basic assembly programming topics
- Will be released on Friday, June 17th
- **Due Monday, June 27th at 11:59 PM**
- Files available on Canvas
- Submit to Gradescope (unlimited submissions)

Review

What would store in R0, R1, R2, R3 if this piece of code ran?

```
.orig x3000
     LD R0, A
     LDR R1, R0, 1
     LEA R2, A
     LDI R3, A
     HALT

A    .fill x4000
.end

.orig x4000
     .fill 3
     .fill 4
     .fill 5
.end
```

Conditional Branching

- We don't have any control structures!
- Everything is just a linear sequence of instructions
- How do we do "if"s and loops?
- Conditional branching lets us skip to a specific instruction
  - This lets us selectively execute some blocks of code, or skip over them
  - We can use this to "translate" familiar if/else statements and loops

Practice

Write a snippet of assembly to compute the absolute value of R1, and place the result back in R1.

```
// Pseudocode
if (R1 < 0) {
    R1 = -R1;
}
```

Answer

Note the inverted condition (BRzp) to skip over the negation, just like the "if" would if R1 was not less than 0.

```
// Pseudocode
if (R1 < 0) {
    R1 = -R1;
}
```

```
     ADD  R1, R1, #0  ; load CC with R1
     BRzp SKIP        ; if R1 >= 0, skip negation
     NOT  R1, R1      ; negate R1
     ADD  R1, R1, #1
SKIP ...
```

Example

It's really easy to compare to zero, but how can we compare a register to another register?

Try to figure out how to express "if (R1 > R2)" in assembly using conditional branching.

```
// Pseudocode
if (R1 > R2) {
    ...
}
...
```

Answer

"R1 > R2" is the same as "R1 - R2 > 0". We know how to do subtraction, and we know how to check if something is greater than zero!

Note that the assembly uses R3 as a temporary register.

```
// Pseudocode
if (R1 > R2) {
    ...
}
...
```

```
     NOT  R3, R2
     ADD  R3, R3, #1   ; R3 = -R2
     ADD  R3, R1, R3   ; R3 = R1 + (-R2)
     BRnz SKIP         ; if R1 - R2 <= 0, skip
     ...
SKIP ...
```

Control Structure Templates: If-Else

```
// Pseudocode
if (R1 > 0) {
    // do option 1
} else {
    // do option 2
}
...
```

```
     ADD   R1, R1, #0
     BRnz  ELSE         ; if R1 <= 0, skip option 1

     ; do option 1
     BRnzp END          ; skip over the else block

ELSE ; do option 2

END  ...
```

Example

Compute the maximum of R1 and R2. Put the result in R3.

```
// Pseudocode
if (R1 > R2) {
    R3 = R1
} else {
    R3 = R2
}
```

Answer

```
// Pseudocode
if (R1 > R2) {
    R3 = R1
} else {
    R3 = R2
}
```

```
     NOT   R4, R2
     ADD   R4, R4, #1  ; R4 = -R2
     ADD   R4, R1, R4  ; R4 = R1 + (-R2)
     BRnz  ELSE        ; if (R1 – R2 <= 0), skip to else

     ADD   R3, R1, #0  ; R3 = R1
     BRnzp END         ; skip past else

ELSE ADD   R3, R2, #0  ; R3 = R2

END  ...
```

Practice

```
if (R0 > R1) {
     R3 = R0 - R1;
} else if (R0 < R1) {
     R3 = R0 + R1;
} else {
     R3 = 2 * R0;
}
```

```
.orig x3000
     NOT R2, R1
     ADD R2, R2, #1
     ADD R2, R0, R2

     BRp FIRSTCONDITION
     BRn SECONDCONDITION
     ADD R3, R0, R0
     BR DONE

FIRSTCONDITION

     NOT R3, R1
     ADD R3, R3, #1
     ADD R3, R0, R3
     BR DONE

SECONDCONDITION

     ADD R3, R0, R1

DONE
     HALT
.end
```

Control Structure Templates: Do-While Loop

```
// Pseudocode
do {
    // do something
} while (R1 > 0);
```

```
LOOP  ; do something
      ADD  R1, R1, #0
      BRp  LOOP        ; if R1 > 0, go back to top
```

Control Structure Templates: While Loop

```
// Pseudocode
while (R1 > 0) {
    // do something
}
```

```
LOOP  ; check condition
      ADD   R1, R1, #0
      BRnz  ENDLOOP     ; if R1 <= 0, break out of loop
      ; do something
      BRnzp  LOOP       ; go back to top

ENDLOOP ...
```

For Loops

for loops are just fancy while loops. To translate a for loop into assembly, first translate it into a while loop, and then translate the while loop into assembly.

```
// Loop to translate
for (int i = 0; i < 20; i++) {
    ...
}
```

```
// Equivalent while loop
int i = 0;
while (i < 20) {
    ...
    i++;
}
```

Demo — `arraysum.asm`

Live coding example: how can we compute the sum of an array?

Use Complx to step through and check/debug your answer!

TRAPs

- Subroutines built into LC-3 to simplify instructions
- Look like normal instructions, but are aliases for TRAP calls
  - "`HALT`" is exactly the same as "`TRAP x25`"
- Each has a corresponding 8-bit Trap Vector
- Usually used for Input/Output

`HALT` (x25): stops running the program
`OUT` (x21): takes character (in ASCII) in `R0` and prints it on console
`PUTS` (x22): given mem address in `R0`, print characters until NULL terminating character (`'\0'`)
`GETC` (x20): takes character input from console and stores it in `R0`

TRAP Demo — `sum.asm`

Live coding example: how can we print (using ASCII encoding) the sum of two numbers using PUTS?

Note: You can see any output from an assembly program in the floating Complx I/O window