CS061: Machine Organization & Assembly Language Lab 3

Agenda

1. Presentation:
    a. If-Statements
    b. 2’s Complement
    c. Loads/Stores Review
    d. Console I/O
    e. Arrays
    f. Lab Descriptions

2. Work Time / Questions / Demos

If Statement - 1

- Goal: Execute IF statement if R0 is positive!

- If R0 is positive, jump to IF_START
- Otherwise, jump to IF_END

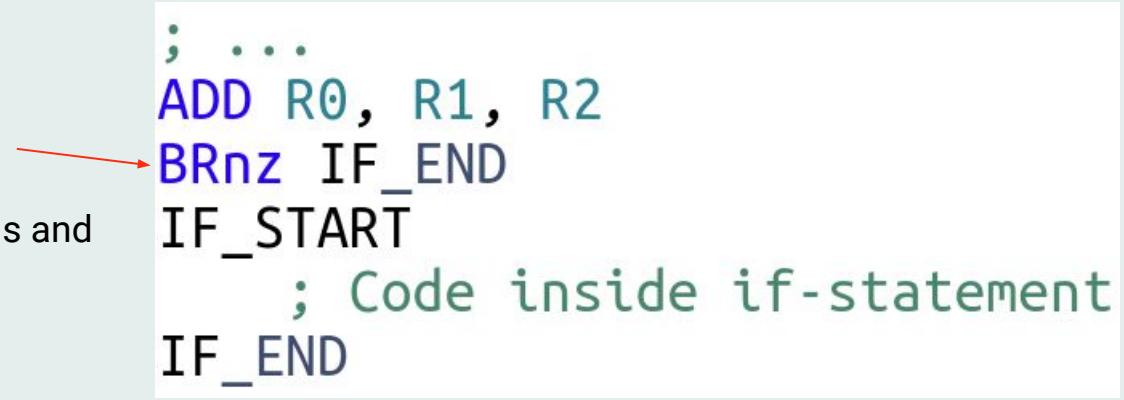
- Code only executes if R0 is positive.

```
; ...
ADD R0, R1, R2
BRp IF_START
BR  IF_END
IF_START
    ; Code inside if-statement
IF_END
```

If Statement - 2

- Code executes sequentially.
- What if we only jump if it's not positive?

- Skip to end if negative of zero.
- If positive, then nothing happens and continues to IF_START.



Positives and Negatives

- Goal: Convert a positive number to a negative number!
- How: 2's Complement.

- Like in class:
  - 1. Flip all the bits.
  - 2. Add 1.

```
; R0 = 5
NOT R0, R0 ; Flip the bits
ADD R0, R0, #1 ; Add 1
; R0 = -5
```

- **NOT**: Unary operator.
  - Destination Register (e.g. R0)
  - Source Register (e.g. R0)
  - Flips all the bits of the source registers and stores it in the destination register.

Loads/Stores Review

- 3 types of Loads & Stores.
- Load/Store Direct (**LD**):
  - Load value from memory address (label)
  - Rn <- Mem[LABEL]
- Load/Store Indirect (**LDI**):
  - Read value from memory address
  - Use that value as a pointer to a different location
  - Rn <- Mem[Mem[LABEL]]
- Load/Store Relative(**LDR**):
  - Use a base register as a pointer to memory address.
  - Rn <- Mem[Rb + Offset]

x3000	LD R0, DATA_PTR
x3001	LDI R1, DATA_PTR
x3002	LDR R2, R0, #0
...	
x3005	DATA_PTR .FILL x4000
...	
x4000	DATA .FILL #42

Loads/Stores Review

- Check out this great video review about LC-3 loading and storing by Westin!

https://youtu.be/7y_D7M_qkP0

(Found in Canvas -> Files -> General Resources -> Misc. Tutorials -> Video tutorials)

From Keyboard to Console

- **GETC** (Trap x20) reads a character that user typed into console and stores its corresponding ASCII number into R0.
  - E.g. User types ‘A’ into console, GETC will store #65 into R0.

- **OUT** (Trap x21) prints out the value in R0 as an *ASCII character* to console.
  - E.g. if R0 has the value #65, then OUT will print ‘A’ to console.

- **Ghost Typing**: When a user inputs a character, but it’s not printed to the console.
  - E.g. if a user types in ‘A’, but an ‘A’ is not printed out.
  - Leads to confusion.
  - Rule of thumb: Put an “OUT” after every “GETC”

I/O Example

```
.ORIG x3000

GETC
OUT

ADD R0, R0, #1

OUT

HALT
.END
```

Questions:

1. What is outputted if the user types in an ‘A’?
2. What if the user types in a ‘1’?

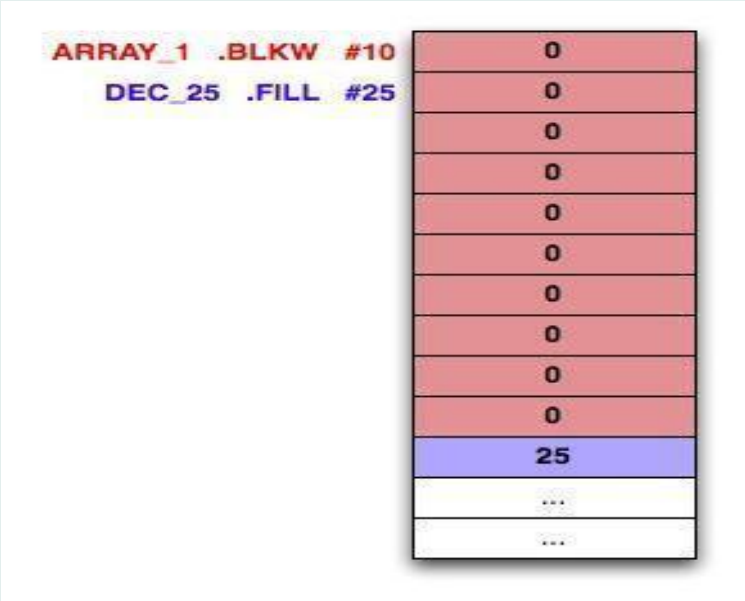
Why is this needed?

Answers:

1. Outputs ‘B’
2. Outputs ‘2’

Carving a Block

- .BLKW is a *pseudo-op* that reserves a certain amount of memory spaces.
  - E.g. “.BLKW #10” would occupy 10 memory spaces
  - .BLKW **does not** initialize those spaces to 0, so there may be junk values there.
  - Useful to create *arrays* (contiguous blocks of memory)



Example

```
.ORIG x3000

x3000 LEA R1, ARRAY_1

      How to load / store an element from
      the array?

x3001 HALT
      ; Local data
x3002 ARRAY_1 .BLKW    #10
      .END
```

- R1 <- x3002
  - x3002 is the address of the first element!
- Reserves 10 spaces
  - From x3002 to x300B

Sentinel-Controlled Loops

- Loops that repeat until the sentinel character is typed in.
  - E.g. a while loop that repeats until the user types in a ‘q’.

- Implementation:
  - Read in a character (GETC).
  - Compare the character against a specified character.
  - If comparison is true, then do not repeat the loop OR break out of the loop.

- Comparison:
  - Main Question: How to check if a register equals another register?
  - Hint: Subtraction!

Exercise 1

- Use one pointer (located in local data) to access two remote data values.
- Use Lab 2 Exercise 3 code as the starting point.

- Use LDR!
  - LDR uses a base register that contains a memory address.
  - How can we modify a register to change its value by 1, 2, etc?

Exercise 2

- Create an array (.BLKW) of 10 locations in local data.
- Prompt user to enter 10 characters and store characters in the array.

- Use LEA to get the address of the array from local data.
  - LEA Format: LEA Rn, LABEL
  - Gets the memory address represented by LABEL and stores it in register n.

- Use the same technique from exercise 1 to traverse the array.

- Use a counter-controlled loop.
  - Use a register as a counter.
  - Subtract 1 from the register inside the loop.
  - Repeat the loop while that register is greater than 0.

Exercise 3

- Copy the exercise 2 code into your exercise 3 file.

- Add another counter-controlled loop that iterates through the array.
  - Read the character at that position in the array.
  - Print out the character *(what register should the character be in?)*.
  - Print out a newline after the character is printed out.

Exercise 4

- Create an array of large size (100 elements) in a REMOTE location (e.g. x4000).
- Copy the exercise 3 code into your exercise 4 file.
  - Remove the local array.
- Use a sentinel controlled loop this time:
  - The first loop stops when it reads a specific character (*called the sentinel character*)
  - Common sentinel character: newline (ENTER key)
  - Repeat the loop until you see the sentinel character.
  - Second loop stops when it reads a specific character from the array (the array will need a sentinel character too).

Demo Info

- **Please sign up to demo only when you have completed all exercises and fully understand your code.**

- Lab Grade Breakdown:
  - 3 points for attendance.
  - 7 points for demoing (+1 bonus point).

- 1 bonus point in the demo category if lab is demo'd *before/during* Friday.
- 3 point penalty if lab is demo'd during the next lab session.