Sprint Challenge: Computer Architecture - Conditional Jumps

This challenge allows you to practice the concepts and techniques learned over the past week and apply them in a concrete project. During this Sprint, you studied CPU components, number bases, bitwise operations, CPU stack, interrupts, and subroutines.

In your challenge this week, you will demonstrate proficiency by adding *conditional jumps* to your LS-8 emulator as well as answering a few questions during a one-on-one session with a PM.

Instructions

Read these instructions carefully. Understand exactly what is expected *before* starting this Sprint Challenge.

This is an individual assessment. All work must be your own. Your challenge score is a measure of your ability to work independently using the material covered through this sprint. You need to demonstrate proficiency in the concepts and objectives introduced and practiced in preceding days.

You are not allowed to collaborate during the Sprint Challenge. However, you are encouraged to follow the twenty-minute rule and seek support from your PM and Instructor in your cohort help channel on Slack. Your work reflects your proficiency with Computer Architecture and your command of the related concepts and techniques.

You have three hours to complete this challenge. Plan your time accordingly.

Commits

Commit your code regularly and meaningfully. This helps both you (in case you ever need to return to old code for any number of reasons and your project manager.

Description

In this code challenge, you will add *conditional jumps* (AKA *conditional branching*) to your LS-8 emulator.

Project Set Up

Options for submission, whichever is easier for you:

- Copy your source into this repo, or...
- Submit a PR for the Sprint Challenge from the `Computer-Architecture` repo you've been using all along.

Minimum Viable Product

Your finished project must include all of the following requirements:

- [ ] Add the `CMP` instruction and `equal` flag to your LS-8.

- [ ] Add the `JMP` instruction.
- [ ] Add the `JEQ` and `JNE` instructions.

See the LS-8 spec for details

In your solution, it is essential that you follow best practices and produce clean and professional results. Schedule time to review, refine, and assess your work and perform basic professional polishing including spell-checking and grammar-checking on your work. It is better to submit a challenge that meets MVP than one that attempts too much and does not.

Validate your work through testing and ensure that your code operates as designed.

Here is some code that exercises the above instructions. It should print:

```
1
4
5
```

```
# Code to test the Sprint Challenge
#
# Expected output:
# 1
# 4
# 5

10000010 # LDI R0,10
00000000
00001010
10000010 # LDI R1,20
00000001
00010100
10000010 # LDI R2,TEST1
00000010
00010011
10100111 # CMP R0,R1
00000000
00000001
01010101 # JEQ R2
00000010
10000010 # LDI R3,1
00000011
00000001
01000111 # PRN R3
00000011
# TEST1 (address 19):
10000010 # LDI R2,TEST2
00000010
00100000
10100111 # CMP R0,R1
```

```
00000000
00000001
01010110 # JNE R2
00000010
10000010 # LDI R3,2
00000011
00000010
01000111 # PRN R3
00000011
# TEST2 (address 32):
10000010 # LDI R1,10
00000001
00001010
10000010 # LDI R2,TEST3
00000010
00110000
10100111 # CMP R0,R1
00000000
00000001
01010101 # JEQ R2
00000010
10000010 # LDI R3,3
00000011
00000011
01000111 # PRN R3
00000011
# TEST3 (address 48):
10000010 # LDI R2,TEST4
00000010
00111101
10100111 # CMP R0,R1
00000000
00000001
01010110 # JNE R2
00000010
10000010 # LDI R3,4
00000011
00000100
01000111 # PRN R3
00000011
# TEST4 (address 61):
10000010 # LDI R3,5
00000011
00000101
01000111 # PRN R3
00000011
10000010 # LDI R2,TEST5
00000010
01001001
01010100 # JMP R2
00000010
01000111 # PRN R3
00000011
```

```
# TEST5 (address 73):
00000001 # HLT
```

Stretch Problems

After finishing your required elements, you can push your work further. These goals may or may not be things you have learned in this module but they build on the material you just studied. Time allowing, stretch your limits and see if you can deliver on the following optional goals:

- [ ] Add the ALU operations: `AND` `OR` `XOR` `NOT` `SHL` `SHR` `MOD`
- [ ] Add an `ADDI` extension instruction to add an immediate value to a register
- [ ] Add timer interrupts
- [ ] Add keyboard interrupts