LEARNINGNERD

HOME › **CATEGORIES:** DAILY LEARNING NOTES

Daily Learning Notes for January 27, 2016

More NAND2Tetris, yay! I learned a bit about machine code and assembly language yesterday, and today I wrote my first assembly program! And it works! I was surprised at how much fun I had with this!

Overview

First thoughts on the assembler and CPU emulator

Before starting on the project for this week, I reviewed the Assembler tutorial (PDF) and CPU emulator tutorial (PDF).

The software tools for this course have an easy-to-use GUI, so they're very beginner-friendly! I love that the provided assembler can show its process step-by-step to help me visualize how assembly language gets translated into machine code. (Later I'll be building my own assembler so I better understand how it works!)

The CPU emulator has a lot more going on compared to the other tools I've used so far, but it's pretty self-explanatory. But I'll have to read the section on debugging in detail later. It supports breakpoints and test scripts and has its own Test Descri Language!

There's also an interesting quote from Maurice Wilkes in 1949 at the end about when he realized the debugging would be a big part of his life. I want to read more about him sometime. Oh, and there was a good quote by Richard Feynman about how he came to understand the difference between knowing the *name* of something and actually *knowing* something. Important distinction!

Writing my first assembly program

My challenge for this fourth project is to write two programs in assembly language: one that multiplies two numbers and one that displays or clears pixels on the screen based on keyboard input.

Where do I start?! OK, deep breath. Start at the beginning. The multiplication program looks easier so I'll start there.

Multiplication program specs:

- The inputs are stored in `R0` and `R1`, the first two RAM locations.
- The program should multiply the two inputs and store the result in `R2`.
- I should assume that both inputs are zero or greater, and that their product will always be less than 32768. (I assume the RAM location can't store any number larger than that.)
- I should use the supplied test script and comparison file to check if my program is working properly.

OK, so I know that multiplication is repeated addition (good thing December was my math month!), and I know that I need some sort of loop to add a number to itself a certain number of times. I'll need to store all this stuff in memory somewhere....

What I *don't* know: How do I a create a loop in assembly? How do I add a number to itself? In other words, I need to figure out which commands I need to use and I need to get familiar with the syntax of this new language. Back to the book!

How to load inputs with assembly language

I learned yesterday that I need to specify a memory location and run an operation in two steps. Looking at the examples in chapter 4 (PDF), I think I would need to do `@R0` to select the location of the first input according to the specs, and then I would need to load that input into the D register with the command `D=M`.

How to add two numbers in assembly language

I found the following commands for adding two variables: `D+A` and `D+M`. I'll need to store the running total in one memory location (may as well use `R2`, which the specs say should store the final result) and add the input to that sum.

So far, here's what I imagine my program looks like if it were to just add a number to itself one time:

```
// set the A register to point to R0
@R0
// load the value of R0 into the D register
D=M
// set the D register to the sum of itself plus the value of R0
D=D+M
```

I think I'll test that out in the assembler and the CPU emulator first just to check my understanding. I'm a little scared to use these for the first time, but let's just take a stab at it…

Alright, so I opened the assembler and CPU emulator. I opened the `mult.asm` file with my text editor, pasted my code in there, and opened the file in the assembler. Now I'll run it… Cool, machine code! It's fun to watch it do the translation. OK, I saved the file it generates, which is named `mult.hack`. Now I just open that new file in the CPU emulator, type a number into the first RAM location, and run it… Awesome, it works! I entered 5 and it shows the D register now contains the value for 10.

Now I just need to figure out how to turn this into a loop!

How to write a loop in assembly language

To multiple my two inputs, I want my program to take `R0` and add it to itself `R1` times, storing the result in `R2`. How do I create a loop based on a variable?

First I need a condition for when the loop should stop. I guess I could subtract 1 from `R1` after each addition of `R0` and then stop my loop when `R1` becomes 0. But how do I create the loop?

Luckily, the book includes an example of a loop, so I can just steal from there! I just need to understand it first…

I guess I need to use one of the jump commands. I can use `D;JEQ` to jump to a specified address if the value of the D register is zero. OK, that makes sense to me now. But how do I know which address it should jump to in order to create the loop?

Ugh, my brain hurts!

Oh, here's what I need! **Label symbols**, user-defined symbols that serve as labels for locations in the instruction memory. Here's how the book explains them:

> "This directive [`(XXX)`] defines the symbol `Xxx` to refer to the instruction memory location holding the next command in the program."

Woohoo, now I think I finally understand the loop example in the book! So I should be able to write `(LOOP)` and `(END)` to mark positions in my code where I want to jump back to later, and then I can jump to those locations with commands like `@LOOP` followed by `0;JMP`.

Writing the complete program

I tried a couple ideas out on paper first, and I finished my first complete assembly program! Check it out:

```
// Create a label symbol to jump back to later
(LOOP)
// set the A register to point to R0
@R0
// load the value of R0 into the D register
D=M
// set the D register to the sum of itself plus the value of R0
D=D+M
// set the A register to point to R2
@R2
// Store the running total in R2
M=D
// set the A register to point to R1
@R1
// set the D register to R1 - 1 (decrement the loop counter)
D=M-1
// set the A register to point to location of (END) label in instruction memory
@END
// if value of D register is zero or less, jump to (END)
D;JLE
// set the A register to point to location of (LOOP) label in instruction memory
@LOOP
// Unconditionally jump to (LOOP), since loop condition is still true if this code executes
0;JMP
(END)
@END
0;JMP // Infinite loop to end the program
```

Now let's try running it with the test script and see what happens! Fingers crossed…

Comparison failure at line 3.

Aw, boo! Well, that's OK. My programs rarely work on the first try, especially if it's my first time using a new language. Let's inspect the comparison file….

Oh! Duh! I'm stupid. I forgot about a very important case: multiplying a number by 0! I'll have to change my program a little bit to check the loop condition *before* running any code. That should fix it.

Let's try this again… Ugh, more errors. Every time I fix one, another one pops up! I'm just doing things in the wrong order, I guess. But I need to stop for now becaus a social outing to go to. I'd rather stay here and play with assembly, but I should cancel at the last minute, so I'll go. Bleh!

A few hours later: I'm back! Well, that socializing wasn't so bad, but I'm happier now that I'm home alone with my laptop and my books.

I took another look at the output file and apparently I forgot about *another* very important case: multiplying a number by 1! Oopsies! But the solution is pretty clear: I should start by adding the number to zero and then keep adding to the total, instead of first adding a number to itself. That was a very silly mistake. Let's fix it and try running the test script again…

End of script - Comparison ended successfully

Success!!! I'm not going to post my final assembly program (I don't want to give away the answer!) but trust me, it is a thing of beauty!

And now, time to sleep.

Questions

- Who is Maurice Wilkes?
- What's the binary representaion of 32768 and what does that tell me about the design of this computer?
  - **Answer:** 32768 is 2 to the power of 15, which tells me that the Hack computer can store a 15-bit number in each of its registers. The registers are 16 bits wide, so I guess one of the bits is reserved for negative numbers via the two's complement method.

Learning summary (#TIL)

- I learned how to use the NAND2Tetris assembler and CPU emulator.
- I wrote my first assembly program!

Next steps

- Watch the video lectures on Coursera
- Complete the project for week 4

Time breakdown

- Study time: 3 hours 48 min
    - Active reading: 1 hour 52 min
    - Writing my first assembly program: 1 hour 56 min

Written by Liz Krane
Liz is a full-time nerd, sharing everything she learns as she tries to learn everything!

Share on Twitter Share on Facebook Share on Google+

Updated January 27, 2016

0 Comments LearningNerd 1 Login

Recommend Share Sort

Start the discussion…

LOG IN WITH

OR SIGN UP WITH DISQUS ?

Name

Be the first to comment.

ALSO ON LEARNINGNERD

English Grammar: Basic Sentence Elements

1 comment • 7 months ago

case study help online — Thanks posting these important terms of the basic sentence elements for English grammar. These are all important

English Parts of Speech: Adjectives, Determiners, and Adverbs

1 comment • 7 months ago

stevej101 — Nice tips!

A moment with my dad and the Pythagorean theorem (Math Immersion

1 comment • 3 months ago

Learn Mathematics — https://youtu.be/aF9xR1rD3Ak

English Punctuation Overview

1 comment • 3 months ago

essay writing services — This is very much in demand to fix the punctuation in English learning approaches. In addition this is making

Subscribe Add Disqus to your siteAdd DisqusAdd Privacy

Advertisement