Thoughts on the Y86 Emulator

Adeeb Kabir

August 14, 2017

Abstract

This file will detail the process of producing and testing code for the Y86 Emulator (PA3).

1 Challenges while writing code

This project details a massive checklist around running a *fetch-decode-execute* loop, and as such, it was incredibly intimidating to start from scratch. Luckly, Professor Russell's instructions were very thorough. I chose to make two main functions where the first would store the input file's directives and the second would execute the instructions as specified in the `.text` directive.

IMPORTANT: I did create a `y86dis` and a custom `.y86` file. However, in regards to my custom `.y86` file, I do not think it fits on my screen. It prints out an ASCII rendition of an enclosed picture of myself.

The ASCII image I loaded into memory is NOT original; I generated it using `http://www.text-image.com/convert/ascii.html`. Also, Professor Russell logged me down for 10 extra credit points concerning shouting what was contained in `prog1.y86`, so I would like to know if that was accounted for.

1.1 Loading the memory image

The most difficult part of the former half came when storing input that was of multiple bytes. However, upon examining the rules C imposes on storing bytes into memory as well as looking at some of the professor's old code for PA1 (where he addressed an array directly as an address with an offset), I found an elegant solution to store these values with relatively minimal lines. The only time my progam's runtime grows with size here is with the length of the largest input string.

1.2 Executing the given instructions

The challenges with the latter half really came down to organization. I chose to use a `switch` statement within a `do...while()` loop which would exit if a halt instruction was encountered. I stuffed most of the functions that fetched from and wrote to memory in `helper.c`. Other than that, most of my loop's operations were facilitated by C, espsecially in the case of`movsbl`, where I just used the `int32_t` type to sign extend when promoting a `char` (1 byte) into a `long` (4 bytes).

2 Challenges while testing code

There is no doubt in my mind that testing and editing my code was far more difficult and time consuming than writing it in the first place, scary as that seemed at the time. The littlest syntax errors (for example, `cpu->counter =+ 5` instead of `cpu->counter += 5` messed up my program counter) completely messed up my stack. Small semantic errors (casting to type `unsigned char` completely erased some of my shifted bytes, so my addresses came out as too small and I read instructions from the byte directives). However, if there is one thing that this project has taught me, it is that `gdb` is my best friend. This also gave me incentive to use only as many variables as I needed, as too many variables on the terminal screen became messy within `gdb`.

My test cases were:

```
./y86emul prog1.y86
./y86emul prog1_2016.y86
./y86emul prog2.y86
./y86dis prog1.y86
./y86dis prog2.y86
```

All disassembled files are written to the original filename with a `.ysm` file extension.

Everything in my execution loop runs in constant $\mathcal{O}(1)$ time. The fetching in my loading functions runs in time dependent the number of directives given n and the length of the largest string s. So my function is bound on top by no more than $\mathcal{O}(ns)$.

3 Conclusion

All in all, I did have a ton of fun with the project, and by extension, this class. While my main interest is the abstract, mathematical side of computer science, CS 211 has enamored me with low-level coding and organization. C has become by far my favorite language to work in, and I have now attained a notable level of comfort in regards operations not too far away from the machine's understanding and am looking forward to CS 214 when this is elucidated on further.

Also, learning how to use and implement LaTeX has been a blast.