CPS 500 - Lab 3

William Katsak

Due: 11/15/2020 11:59pm ET

Overview

For this assignment, you will dive into MIPS instruction encoding by writing your own instruction *decoder* in C.

Cheating

Please be sure that all work is your own. Looking up the solution on the Internet to any of these problems counts as cheating. If caught, you will receive a 0 for the assignment and will be reported to the academic integrity office.

Background and Setup

For this lab, I recommend that you work in the Ubuntu Virtual Machine that we used for Lab 1. If you happen to have a Mac, Xcode/gcc will work fine as well.

What to turn in

Please submit your `.c` file, as well as compilation instructions.

Assignment

Description

Starting with the below program, you write a program that:

1. Accepts a string of 32 [0/1]'s as a command line argument.
2. Displays the assembly language equivalent.
3. Requirements:
   (a) **Instruction Type**: Your program must support decoding the following instruction types: `add`, `addi`, `sub`, `beq`, `bne`, `slt`, `j`, `jr`, `sll`, `srl`, `lw`, `sw`.
   (b) **Register Operands**: Registers must be displayed in symbolic form, e.g. `$t0`, `$s0`, `$sp`, etc. and not simply as register numbers.
   (c) **Constants**: Constants (including jump offsets) should be displayed in decimal (base-10). Assume that immediate values are in 2's complement.
4. Note that labels are replaced by constants by the assembler. For branches, you can just display the offset that is contained in the instruction. For unconditional jumps, you should figure out the actual destination address (you can use 1111 for the four high order bits).
5. Your output should be correct MIPS assembly (e.g. I should be able to copy it into MARS and reassemble it).

Example

```
$ ./instruction_decoder 00100001010010010000000000000101

addi $t1, $t2, 5
```

Testing

It is possible to use MARS to generate binary strings to use as input to your instruction decoder. To do this, first write (and test) a correct program in MARS and save it to an `.asm` file. Then, you can run the following from a command line:

```
java -jar <path_to_mars_jar> a dump .text BinaryText <output_file> <asm_file>
```

This will result in your program being assembled and written to `<output_file>`, with each instruction as an individual 32-bit binary string. Keep in mind that some assembly pseudo-instructions are actually assembled into more than one actual machine instruction.

Starting Point

```
#include <stdio.h>
#include <string.h>

#define LENGTH 32
#define false 0
#define true 1

// Checks that the char array is 32 bits long
// and only has 0 and 1 in it.
int check(char* bits) {
    if (strlen(bits) != LENGTH) {
        return false;
    }
    for (int i = 0; i < LENGTH; i++) {
        if (bits[i] != '0' && bits[i] != '1') {
        return false;
        }
    }
    return true;
}

// We will take the bitstring as a command line argument.
int main(int argc, char **argv) {
    // Make sure we have at least one argument.
    if (argc < 2) {
        fprintf(stderr, "No argument found\n");
        return -1
    }

    // Get a pointer to the argument.
    char *instruction = argv[1];

    if (!check(instruction)) {
        fprintf(stderr, "Invalid bit string\n);
        return -2;
    }

    // Your code here!
    // You can use regular printf()'s to produce your output.

    return 0;
}
```