CSE 140 Project Report

Team 7: Nicholas Manzano and Deanicia Delizo

Date: May 5, 2021

1. Single-cycle MIPS CPU

1.1 Overview

We implemented our single-cycle program in C++ language. Each stage has their own method, such as taking bits in a form of a string and transferring them in the correct binary. Our code starts off by declaring tuples. These tuples return two parameters like variables and strings. Our next portion includes all the functions and declarations for the required parts in the project. When seen there are approximately 18 added functions starting near around line 20 and the remaining required functions. Our code then calls upon each declaration and performs each block to register the output. Lastly, our main function includes all registers and are given a variable, in order for us to easily call upon them. The main function then analyses the given binary in the separate text file and gives us our needed output.

1.2 Code Structure

1.2.1 Functions

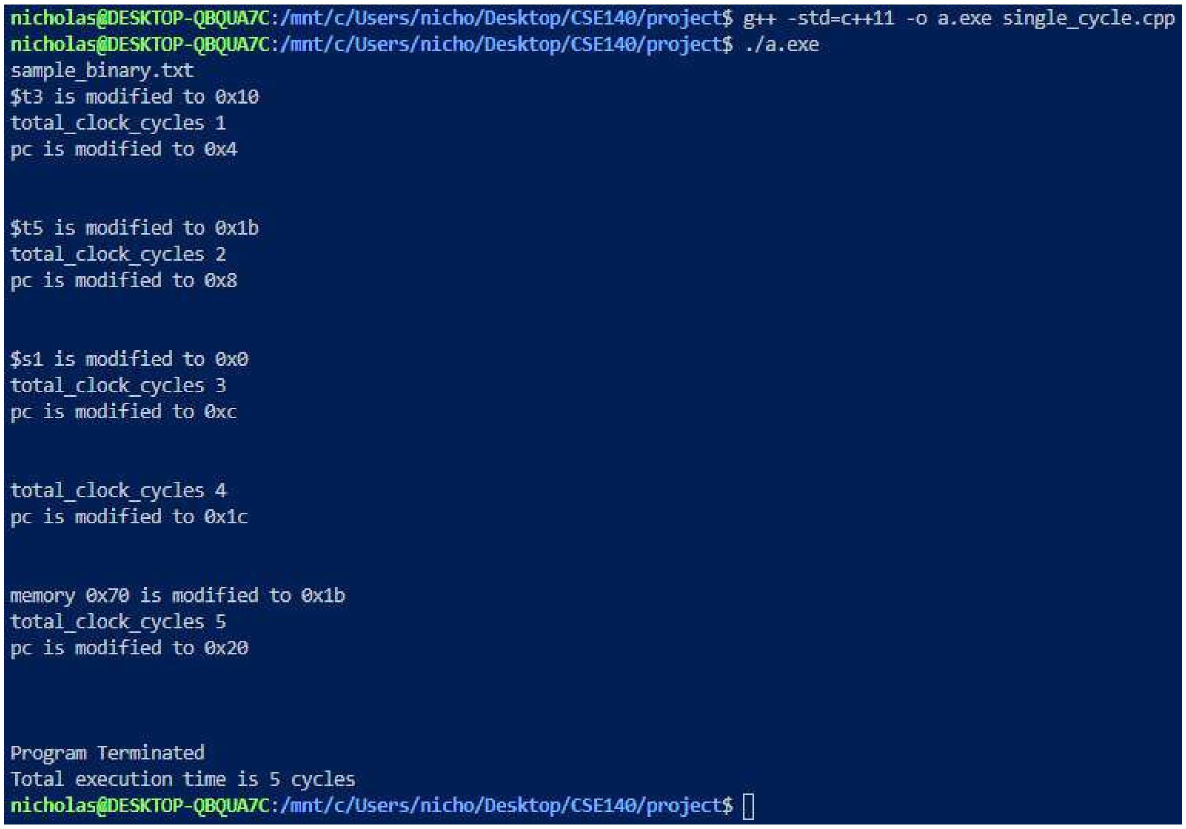
The required functions of our code are: main(), fetch(), decode(), execute(), mem(), and writeback(). Excluding the required functions for this project we have included around 18 more functions. They are located in the top portion of our code. We choose to add these additional functions due to the fact that it was easier and more concise to further compartmentalize individual conversions and calculations rather than having a large block of code. For example rather than writing one long section to do the branch target calculation we can break it down into several methods that, sign extend, shift left 2, and add pc +4. Additionally, for the purposes of how we dealt with inputting the sample binary text and handling our current instruction it became easier to write methods that were capable of converting bit values to decimal integers. An example of that is all of our get functions, such as getOpcode(), that retrieves the correct separate fields for our current instruction and stores them in a string array so that we can later use them.

1.2.2 Variables

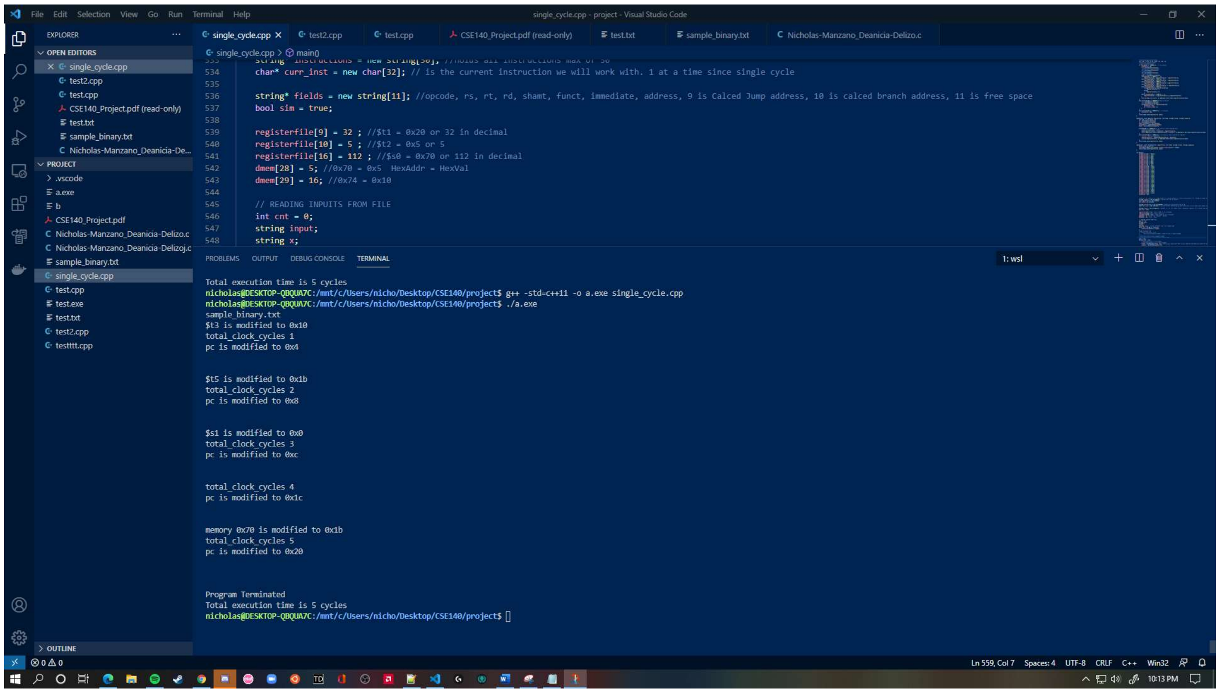
We used multiple variables in our code. More precisely, multiple variations of arrays: string arrays, arrays that hold all instructions, character arrays that hold int values. Our code includes variables of fields that hold separate branches and all the required ones that were given to us in the project files. Such examples are: opcode, rs, rt, rd, shamt, funct, immediate, address, calced jump address, calced branch address, and a free space variable. Each variable allows us to store and analyze the binary input files. Once done and processed we are given our output.

1.3 Execution Results

Our execution results were based on the given output in the project pdf file. However, it is not exactly as the provided output example as our code switches lines. All output is still correct, just not in the proper order. As shown below this is an example of our programs output:



```
nicholas@DESKTOP-QBQUA7C:/mnt/c/Users/nicho/Desktop/CSE140/project$ g++ -std=c++11 -o a.exe single_cycle.cpp
nicholas@DESKTOP-QBQUA7C:/mnt/c/Users/nicho/Desktop/CSE140/project$ ./a.exe
sample_binary.txt
$t3 is modified to 0x10
total_clock_cycles 1
pc is modified to 0x4


$t5 is modified to 0x1b
total_clock_cycles 2
pc is modified to 0x8


$s1 is modified to 0x0
total_clock_cycles 3
pc is modified to 0xc


total_clock_cycles 4
pc is modified to 0x1c


memory 0x70 is modified to 0x1b
total_clock_cycles 5
pc is modified to 0x20



Program Terminated
Total execution time is 5 cycles
nicholas@DESKTOP-QBQUA7C:/mnt/c/Users/nicho/Desktop/CSE140/project$
```



```cpp
    char* curr_inst = new char[32]; // is the current instruction we will work with. 1 at a time since single cycle

    string* fields = new string[11]; //opcode, rs, rt, rd, shamt, funct, immediate, address, 9 is Calced Jump address, 10 is calced branch address, 11 is free space
    bool sim = true;

    registerfile[9] = 32 ; //$t1 = 0x20 or 32 in decimal
    registerfile[10] = 5 ; //$t2 = 0x5 or 5
    registerfile[16] = 112 ; //$s0 = 0x70 or 112 in decimal
    dmem[28] = 5; //0x70 = 0x5  HexAddr = HexVal
    dmem[29] = 16; //0x74 = 0x10

    // READING INPUITS FROM FILE
    int cnt = 0;
    string input;
    string x;
```

```
Total execution time is 5 cycles
nicholas@DESKTOP-QBQUA7C:/mnt/c/Users/nicho/Desktop/CSE140/project$ g++ -std=c++11 -o a.exe single_cycle.cpp
nicholas@DESKTOP-QBQUA7C:/mnt/c/Users/nicho/Desktop/CSE140/project$ ./a.exe
sample_binary.txt
$t3 is modified to 0x10
total_clock_cycles 1
pc is modified to 0x4


$t5 is modified to 0x1b
total_clock_cycles 2
pc is modified to 0x8


$s1 is modified to 0x0
total_clock_cycles 3
pc is modified to 0xc


total_clock_cycles 4
pc is modified to 0x1c


memory 0x70 is modified to 0x1b
total_clock_cycles 5
pc is modified to 0x20



Program Terminated
Total execution time is 5 cycles
nicholas@DESKTOP-QBQUA7C:/mnt/c/Users/nicho/Desktop/CSE140/project$
```

1.4 Challenges and Limitations

Some challenges and limitations we encountered while implementing our code were human interactions. Due to the covid pandemic we as a group had issues with working on the code together as we both tried to implement different suggestions on a single computer over call. Our main challenge was towards the end when we executed our code we saw that our 4th cycle showed an incorrect output. We faced this challenge due to the fact that we did not understand why the output was the way it was (0x1c). However we fixed this in our code, specifically line 456 sets the pc value to branch target-4 which then sets up the next pc update to being our branch target address.

Our code does have some limitations such as during the 4th cycle we noticed that the output differs from the given output provided for us. We were contemplating to print the pc value of the branch instruction line itself or printing the value just before our branch target. Ultimately, we opted for the 2nd option since that output matched the given sample which we assume is correct. We collaborated with each other and both of us agreed that the result of 0x1c may perhaps be wrong.