Project Report:
Phase I

Mitchell Moran • Noah Perryman
ECE 2162

11/7/2018

Instructor: Prof. Jun Yang

I. FILE SUBMISSION AND INSTRUCTIONS

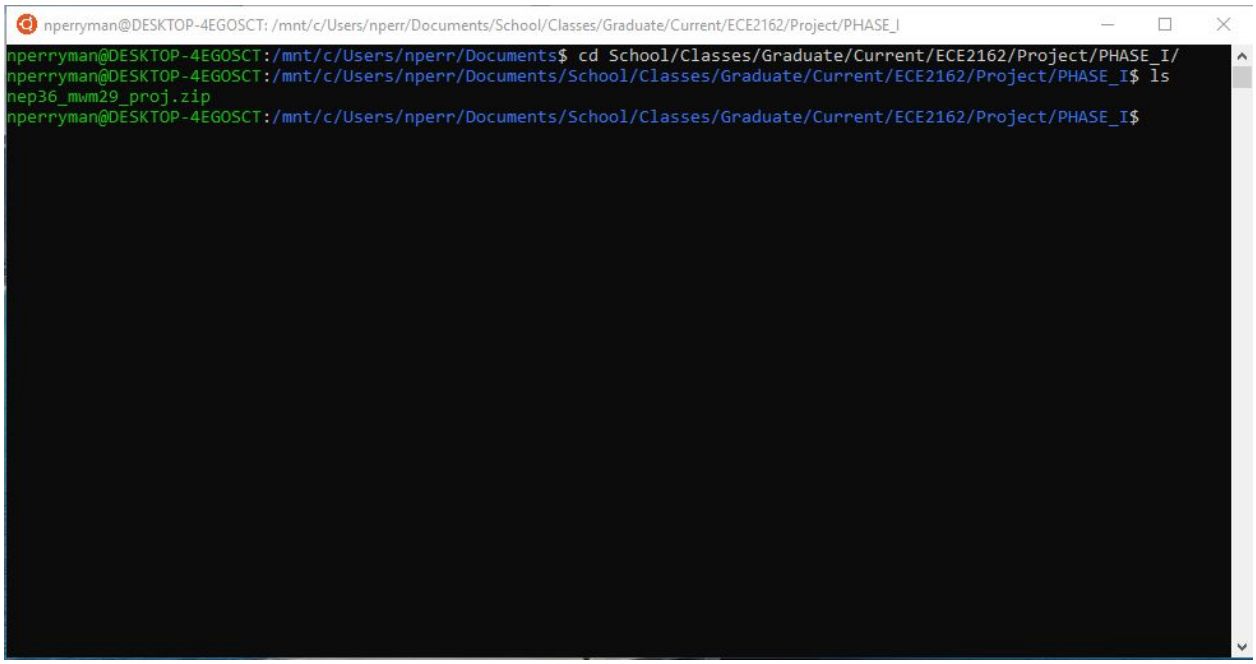
The contents of the project file can be found in the zipped folder ‘nep36_mwm29_proj.zip’. This zipped file contains the main code (nep36_mwm29_proj.c), a header file (struct_defs.h), a benchmarks folder, and this report (nep36_mwm29_report.pdf). The benchmarks folder contains the input text files for the main program as well as the solutions (correct output of input files) to those benchmarks. The files are labeled as benchmarki_j.txt where i stands for the benchmark category and j represents a test case within that benchmark. The benchmark categories are defined in ‘bencchmark_categories.txt’.

I.1 SYSTEM REQUIREMENTS

In order to compile and execute, and therefore utilize the input files, a minimum set of system requirements are needed. The user needs to have a computer capable of compiling and executing c code as well as a text editor to view and edit the input files. A command line interface is preferred as this is what was used to test and execute the program files. GCC, the GNU Compiler Collection, is required for the compilation of this program. For the purposes of this example we will be giving instructions to compile, execute, and read the output of the file assuming the user is using a command line interface (we used the Ubuntu linux subsystem for Windows interface, but any command line interface such as a MAC terminal or a linux terminal will work). A file unzipping command is also needed to unzip the zipped file (we used unzip from the command line).

I.2 COMPILE INSTRUCTIONS

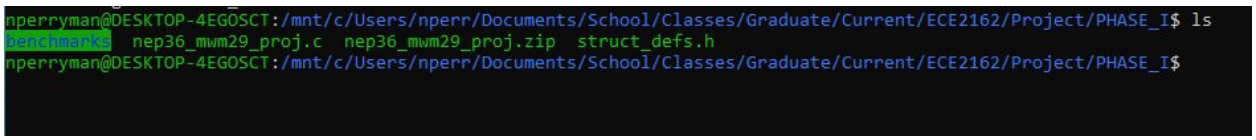
Given the assumption in section I.1, the user first needs to change their directory to the directory containing the zipped folder described above. To confirm that you are in the right folder, an ‘ls’ command should list the zip folder as shown in the figure below.



Next, the zipped file needs to be unzipped. This can be done in any way that the user chooses to, but we used the unzip command on the command line:

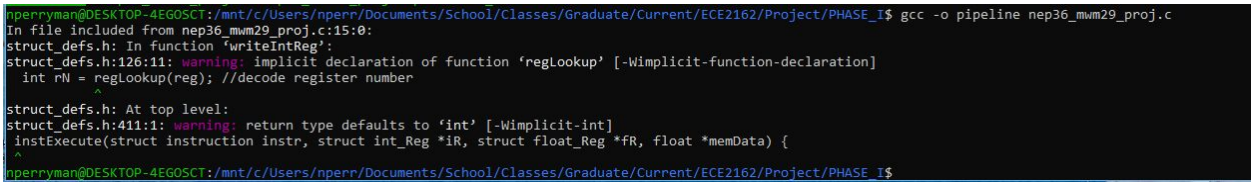
unzip nep36_mwm29_proj.zip -d .

Using ‘ls’ to show the contents of the folder should show the files as shown in the figure below after the unzip command.



To compile the program now, simply use the gcc command on the command line as follows. A few warnings will pop up (these can be ignored), but the contents should look like the figure below. Here, ‘pipeline’ is used as the name of the executable file, but the user can choose whatever name they prefer (given that this change is made throughout the rest of the instructions).

gcc -o pipeline nep36_mwm29_proj.c



I.3 EXECUTABLE

As per the gcc compilation shown in section, the executable will be called 'pipeline'. To run this command simply do the following:

./pipeline input_file

The following section will cover what the input_file field is.

I.4 INPUT INSTRUCTIONS

To enter any input for the program, simply edit a text file following the format given below.

Line 1: 'Integer Adder rs' 'Integer Adder EX Cycles'' Integer Adder Mem Cycles' Integer Adder FUs'
Line 2: 'FP Adder rs' 'FP Adder EX Cycles' 'FP Adder MEM Cycles' 'FP Adder FUs'
Line 3: 'FP Multiplier rs' 'FP Multiplier EX Cycles' 'FP Multiplier MEM Cycles' 'FP Multiplier FUs'
Line 4: 'Load/Store rs' 'Load/Store EX Cycles' 'Load/Store MEM Cycles' 'Load/Store FUs'
Line 5: 'ROB Entries' 'CDB Buffer Entries'
Line 6: 'Number of Integer Register Inits' 'Number of FP Register Inits' 'Number of MEM Inits'
Line 7: 'Number of Instructions'
Line 8 (and so on until FP Register Inits): 'Integer Register' 'Value to Initialize'
Line 9 (and so on until MEM Inits): 'Floating Point Register' 'Value to Initialize'
Line 10 (and so on until instructions): 'Mem' 'Memory Position' 'Value to Initialize'
Line 11 (and so on until no more instructions): 'Instruction'

For example, the following would be an example input .txt file.

2 1 0 1
3 3 0 1
2 20 0 1
3 1 4 1
128 1
10 0 0
5

R1 1
R2 2
R3 3
R4 4
R5 5
R6 6
R7 7
R8 8
R9 9
R10 10
Add R11, R1, R2
Add R12, R3, R4
Add R13, R5, R6
Add R14, R7, R8
Add R15, R9, R10

In order to run the benchmark files, the following command will be given.

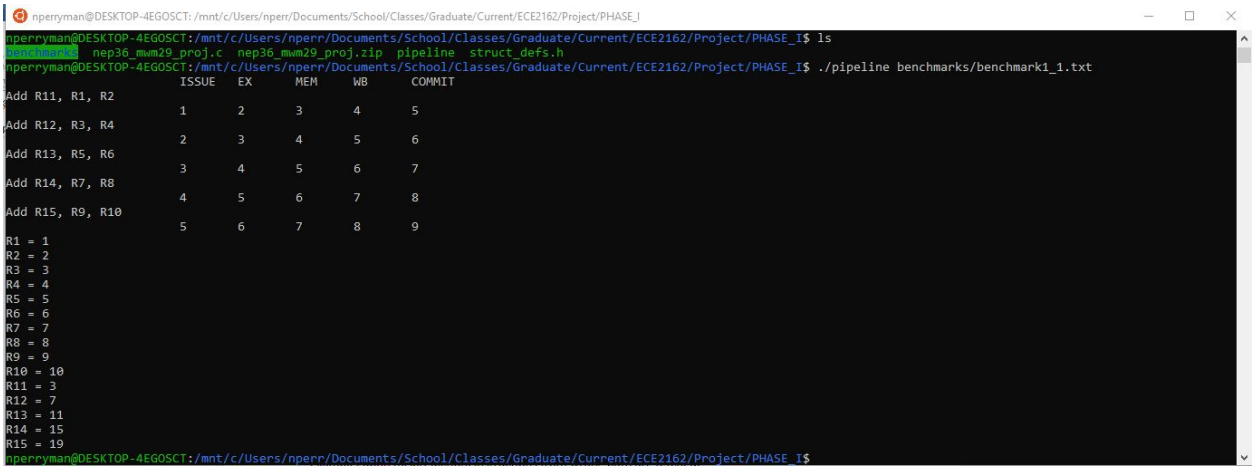
./pipeline benchmarks/benchmarki_j.txt

For example, if you wanted to run the 1_1 benchmark:

./pipeline benchmarks/benchmark1_1.txt

I.5 OUTPUT

The output will be given on the command line screen for the user to see. For example, if the above command (./pipeline benchmarks/benchmark1_1.txt) was run, the following output on the command line would be seen.



The output shows when each instruction entered each stage of the pipeline and when it was committed. It also shows all non-zero values of the integer register files, the floating point register files, and memory. If the command line screen is not sufficient and the user would like to store the output in a .txt file, the user can use the following command line command.

./pipeline benchmarks/benchmarki_j.txt > output.txt

This will store the output into the file output.txt in the directory that the user is in. The user can then view this file instead to see the output contents.

II. TEST BENCHMARKS

The benchmarks created for testing functionality of the code are broken into 6 different categories based on what they are testing. In general, the higher the category number, the more difficult the test. 2-3 benchmarks were made for each category. The descriptions of each benchmark are given below.

Category 1 - no dependencies
1. All integer adds with no dependencies
2. 1 instruction per FU no dependencies

Category 2 - data dependencies (true and false)
1. False dependencies only
2. True and false dependencies

Category 3 - load/store instructions
1. Loads and stores without forwarding
2. Loads and stores with forwarding

Category 4 - structure hazards

1. Structure hazards in reservation stations
2. Structure hazards in CDB
3. Structure hazards in ROB

Category 5 - loop on earlier categories

1. Loop with true and false data dependencies
2. If statements and loop together

Category 6 - branch prediction

1. Benchmark 5.1 with branch prediction
2. Benchmark 5.2 with branch prediction

III. GROUP RESPONSIBILITIES

For the first phase of the project, the following responsibilities were given. Noah Perryman wrote the majority of the c files (nep36_mwm29_proj.c and struct_defs.h) and determined the input file format as well as compilation/execution instructions. Mitchell Moran handled some struct fields in the struct_defs.h (lines 46 - 108) file as well as wrote and tested all of the benchmark files in the benchmarks folder. Noah was in charge of testing and debugging the code while Mitchell made sure that each of the benchmark files were correct and well written. During the project meetings both members worked on their respective tasks and equally engaged in discussion about the scope and responsibilities for the project. Both members communicate often and were involved in creating a github repository for the project.

Moving forward, Noah will be responsible implementing and debugging benchmark1_1 through benchmark3_2 while Mitchell will be responsible for implementing and debugging benchmark4_1 through benchmark6_2. In order to maintain the code, both members will be responsible for version control via meetings and the github repository.