

CSC3420 Computer System Architectures

Project Phase 2

Deadline: 23:59 14th March, 2009

1. Introduction

This course project is intended to let you have a more in depth understanding of CPU architecture, by writing an assembler of a simplified assembly language, a single-cycle simulator and a multi-cycle pipelined simulator of a simplified CPU. The project is divided into 3 related phases. In this phase, you have to write a single-cycle simulator.

2. Project Settings

In order to encourage real understanding, each of you is required to finish a slightly different task. The variations and the hashcodes are exactly equal to the settings of phase 1:

- 4 sets of opcodes (set 1 to 4)
- 4 kinds of conditional branches – 0: beq, 1: bne, 2: blt, 3: bge
- Either 16 registers (hash code 0) or 32 registers (hash code 1)

3. Architecture

The architecture (instruction set) you are simulating is exactly the same with phase 1. Figure 2 in this specification shows the datapath you need to simulate.

4. Simulator

You are required to write a single-cycle simulator in standard C (and CANNOT use extra libraries) which simulates what the MIPS does **cycle-by-cycle**.

4.1 Input

Your simulator should be able to accept two arguments, the first one is your student ID and the second one is the filename of your binary assembly program. The main function is provided for you in the provided resources, and you are highly NOT suggested to edit any provided code segments in the skeleton program.

4.2 Tasks

You are required to complete the first four methods in the provided `phase2.c` (task 1 to task 4) and to implement methods for every component in the datapath (task 5). It is **FORBITTEN** to declare any extra global variables in any files, and it is **NOT allowed** to modify any methods except the following **FOUR** methods in `phase2.c`

4.2.1 Read Binary File

In `read_machine_code_file` method, you have to read the instructions from the binary file and store them into the memory correctly. Please note that the instructions should be stored at the memory address starting from `0x0000` and the input binary file is in *big-endian*.

4.2.2 Connect Component

In `connect_components` method, you have to connect the components according to the datapath properly by constructing linkage from every input to its corresponding value.

4.2.3 Initialization

In `initialize_register_states` method, you have to initialize all the output states of all the components to 0, except Mux 3.

4.2.4 Simulation Sequence

In `simulate_cycle` method, you have to put down the correct working sequence of the components in one single cycle by calling the component methods sequentially. Program Counter is the first working component in a cycle, therefore its corresponding component method `program_count` is called first.

4.2.5 Component Methods

In this task, you have to implement all the component methods. Please note that usually each component should have ONE component method, but sometime it can have more than one. Such as the Registers Unit, it has more than one different working task in one single cycle which can be reading, writing... etc., therefore it can have more than one component method responding for these tasks.

Inside the component method, you have to simulate what it does according to the control signals.

Special Requirement

For Instruction Memory Unit, if the current fetched instruction is ZERO (no more instructions), it should return a termination signal to the provided method `simulate_program`.

4.3 Control Signals and Datapath

For any control signal lines (Orange lines on Figure 1), they allow **ONLY ONE** bit signal (either 0 or 1) to be passed.

5. Marking Scheme

The distribution of marks of phase 2 is as follows:

Part	Marks
Simulator	
Read the binary assembly file correctly	10%
Connect the components properly	10%
Initialize the components appropriately	10%
Correct simulation sequence	30%
Complete the component methods	40%

Differ from phase 1, the simulating environment of phase2 is Linux, Please make sure that your submitted program would be compiled on the server *linux2*, using *gcc*, so please make sure your simulator can be compiled and runs correctly under this setting. A sample simulator has been provided for you in phase 1 to test the behaviors of your simulator.

Please refer to the [Appendix for details](#).

6. Submission Guidelines

You should tar and gzip the following files as `tmchan.tar.gz` and send the file to csc3420@cse.cuhk.edu.hk with `uuencode` and `mailx` command

- For the simulator
 - All `.c` and `.h` files of your simulator
 - A makefile, which when `make` is executed, should compile your simulator properly

Note the followings:

- there will be **ZERO tolerance for plagiarism**
- the deadline will **NOT** be postponed, so you are advised to start doing phase 2 early.

CSC3420 Project Phase 2

Appendix

Detailed Marking Scheme of Project Phase 2

The total marks will be distributed into 2 areas.

1. implementation of the simulator, it contributes 70%
2. individual performance in the demonstration, it contributes the rest 30%

A. Simulator Implementation (70%)

1. Read the binary file generated by phase 1 assembler correctly (10%)

Note: NO appeal will be accepted on ANY (minor) mistakes

2. Connect all the components (10%)

Note: NO appeal will be accepted on ANY mis-connection / wrong-connection

3. Initialize the component states (10%)

Note: NO appeal will be accepted on anything

4. Correct Simulation Sequence (30%)

10 marks deduction for each failing of the six component method testcases

Note: NO appeal will be accepted on ANY mis-implemented single pseudo-instructions

5. Component Method (40%)

Six testcases will be run on your simulator and compare the behaviors with our sample simulator, the six testcases include different sets of similar pseudo-instructions. Please make sure your simulator well performs for every pseudo-instruction.

Note: NO appeal will be accepted on ANY mis-implemented single pseudo-instructions, and NO customization on any testcases will be allowed

- i. R-type (6%)
- ii. I-type (6%)
- iii. Memory (8%)
- iv. Branch (8%)
- v. Jump (6%)
- vi. Procedural Call (6%)

B. Demonstration (30%)

Different questions will be asked to each individual on the following two areas. Different marks will be awarded to each individual depending on his/her understandings.

1. Understanding the datapath
2. Implementation

Marksheet Excel header for Project Phase 2

Simulator Implementation (70%)										Demo (30%)
Read Binary (10 marks)	Connect Components (10 marks)	Initial Components (10 marks)	Correct Flow (30 marks)	Complete Component methods (40%)						
				R-type (6 marks)	I-type (6 marks)	Memory (8 marks)	Branch (8 marks)	Jump (6 marks)	Procedural Call (6 marks)	

Remarks

1. 50% penalty for not following any instructions on the specification, please read specification clearly.
2. 100% penalty for late submission or any kind of resubmission (i.e. change of code) after deadline, please decompile the acknowledge e-mail and examine your submitted file