MTH410E – RISC-V Architecture and Processor Design

Assignment 3: Pipelined RiscV Core in SystemVerilog

Deniz Bashgoren 040180902

1. Introduction

This project is the continuation of the singlecycle RiscV processor submitted in the 2nd assignment. Now the processor implements a 5-stage pipeline. It uses bypassing, stalling and flushing techniques to achieve a CPI ~ 1.5.

2. Source Code Organization

The main folder contains the following:

- **src/** : The folder with the source code. The top module is riscv_pipelined.sv.
- **tb/** : The testbench modules. riscv_pipelined_tb.sv is the module controlling the entire core. The other testbenches are there to test the various parts of the core. Running each of them should print "No errors" on the screen.
- **samples/** : The folder containing simple C programs to test the core. Each sample consists of a C source file (like prime.c), an elf executable file, compiled by GCC (prime.c.elf), an disassembler output file (prime.c.dump), and hex files ready to be imported by SystemVerilog, extracted from the elf file (prime.c.elf.imem and prime.c.elf.dmem).
- **utils/** : A script written in javascript that generates *.imem and *.dmem files from a given *.elf file. To parse the elf file it uses the library node-elf-file [1].
- **makefile** : The intended way to build the project.
- **compilation.ld** : A linker script used to reorder the memory mapping of the sections.
- **Report.pdf** : This file.

2. Required Software

Here it's assumed that the user runs a GNU/Linux machine that has access to basic tools like make. The software needed to run the core:

- **GNU Compiler Collection for RiscV** : This is used to compile the sample C files to RiscV binary files. Run the **`riscv64-linux-gnu-gcc`** command to make sure it's installed.
- **NodeJS** : This is a javascript runtime that is used to run the scripts to prepare the *.dmem and *.imem files. Run the **`node`** command to make sure it's installed.
- **Icarus Verilog** : The SystemVerilog simulator used for this project.

3. How to Run

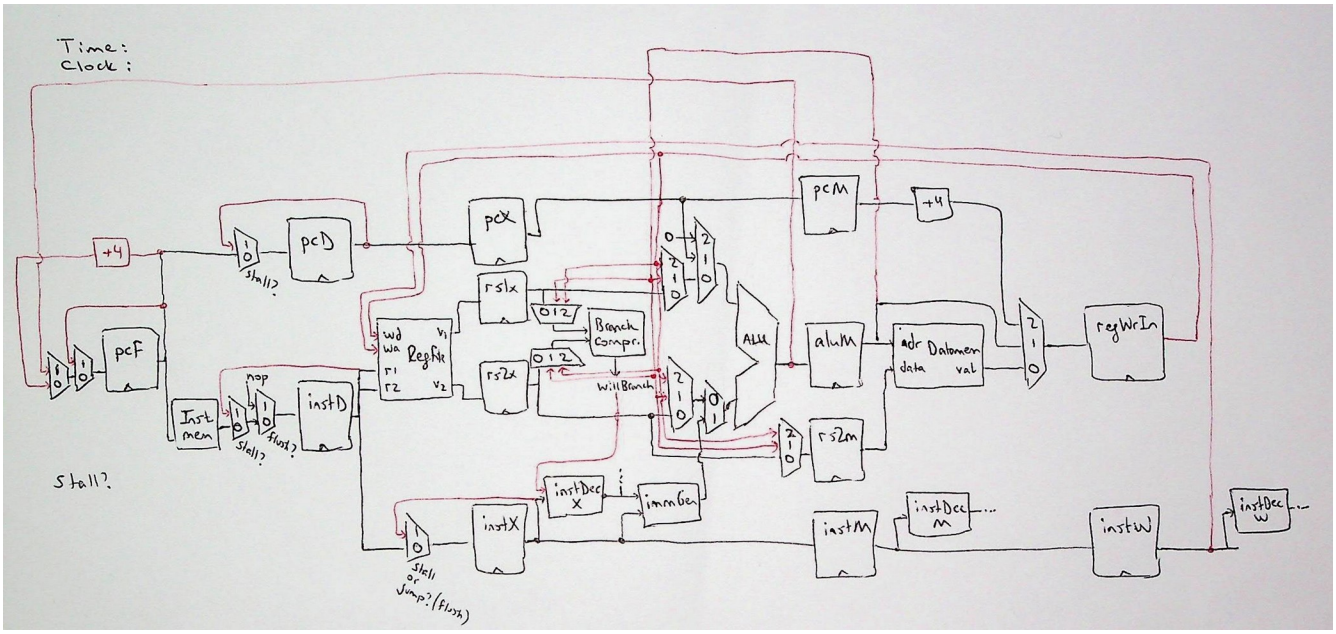
Refer to the report submitted together with the 2nd assignment. The make command to compile source files has changed from

`make prepare SRC=…`
to
`make prepare_c SRC=…`
and
`make prepare_asm SRC=…`

Note that for the bubblesort test program INITIAL_PC=112 and for the other three INITIAL_PC=36. Also note that in order to run a test, first the test should be compiled by using make **`prepare_*`**, then the source code of the top module should be edited to change the parameters, and then **`make cpu`** should be executed to run the code.

4. Changes from the SingleCycle Processor

- Regfile now updates on negedge clock. This is done to avoid structural hazards.
- 4 sets of intermediate registers are placed between the 5 stages. PC and instructions are propagated along the pipeline, together with the main datapath. The instruction decoder in assignment 2 is replaced with 3 seperate decoders, one for each of X, M and W stages. The naming convention uses the letters F (fetch), D (decode), X (execute), M (memory), W (writeback).
- A forwarding unit determines whether to forward values from M and W stages. These measures prevent data hazards. Branch comparator inputs are also now behind the forwarding muxes. Forwarding unit is placed in the top module, and not on its own module.
- To prevent load-use hazards, 1 cycle stall (bubble) is applied.
- Jumps and branches are decided on X stage. 2 cycles of stalling is applied and the 2 instructions after the jump are flushed. Prediction unit is not implemented.



The schematic above illustrates the entire pipelined processor. Black colored parts are forward path, and red colored ones are backward paths (feedbacks). Control words and inputs are not shown.

5. Performance

There are 4 test programs provided in the samples folder, all written in C. All of the test programs are tested and give correct results. Special assembly instruction combinations that would result in hazards

are also tested and all give correct results. These are not provided in the samples folder. The results are as following. The same program is fed to singlecycle (of assignment 2) and pipelined processors. Assuming that the singlecycle processor has CPI=1, the CPI of the pipelined one is calculated by scaling the completion time.

Program and inputs	**Completion time of the Singlecycle CPU**	**Completion time of the Pipelined CPU**	**CPI of the Pipelined CPU**
Bubblesort: 4 numbers	t=709	t=1057	1.49
Prime: 6^{th} prime	t=3763	t=5645	1.5
Findmax: 17 numbers	t=595	t=859	1.44
Strlen: 12 characters	t=243	t=377	1.55

The pipelined CPU has a higher CPI, which might make it look like a worse-performing processor, however it shouldn't be forgotten that performance doesn't only take CPI into account; it also considers the time it takes to pass the critical path. We can roughly estimate the critical path to be 1/5 that of the singlecycle CPU. So, overall a performance gain of 3x should be achieved.

6. References

[1] https://github.com/k13-engineering/node-elf-file