CS 2410 Computer Architecture

Spring 2023

Course Project

Distributed: Feb 19th, 2023

Due: 11:59pm April 25th, 2023

Introduction:

This is a single-person project.

You are allowed and encouraged to discuss the project with your classmates, but **no sharing of the project source code and report**. Please list your discussion peers, if any, in your report submission.

One benefit of a dynamically scheduled processor is its ability to tolerate changes in latency or issue capability in out of order speculative processors.

The purpose of this project is to evaluate this effect of different architecture parameters on a CPU design by simulating a modified (and simplified) version of the PowerPc 604 and 620 architectures. We will assume a 32-bit architecture that executes a subset of the RISC V ISA which consists of the following 10 instructions: fld, fsd, add, addi, slt, fadd, fsub, fmul, fdiv, bne. See Appendix A in the textbook for instructions' syntax and semantics.

Your simulator should take an input file as a command line input. This input file, for example, prog.dat, will contain a RISC V assembly language program (code segment). Each line in the input file is a RISC V instruction from the aforementioned nine instructions. Your simulator should read this input file, recognize the instructions, recognize the different fields of the instructions, and simulate their execution on the architecture described below in this handout. Your will have to implement the **functional+timing** simulator.

Please read the following a-g carefully before you start constructing your simulator.

The simulated architecture is a **speculative, multi-issue, out of order** CPU where:

(Assuming your first instruction resides in the memory location (byte address) $0x00000_{hex}$. That is, the address for the first instruction is $0x00000_{hex}$. PC+4 points to next instruction).

a. The fetch unit fetches up to NF=4 instructions every cycle (i.e., issue width is 4).

b. A 2-bit dynamic branch predictor (initialized to predict weakly taken(t)) with 16-entry branch target buffer (BTB) is used. It hashes the address of a branch, L, to an entry in the BTB using bits 7-4 of L.

c. The decode unit decodes (in a separate cycle) the instructions fetched by the fetch unit and stores the decoded instructions in an instruction queue which can hold up to NI=16 instructions.

d. Up to NW=4 instructions can be issued every clock cycle to reservation stations. The architecture has the following functional units with the shown latencies and number of reservation stations.

Unit	**Latency (cycles) for operation**	**Reservation stations**	**Instructions executing on the unit**
INT	1 (integer and logic operations)	4	*add, addi,slt*
Load/Store	1 for address calculation	2 load buffer + 2 store buffer	*fld* *fsd*
FPadd	3 (**pipelined** FP add)	3	*fadd, fsub*
FPmult	4 (**pipelined** FP multiply)	3	*fmul*
FPdiv	8 (**non-pipelined** divide)	2	*fdiv*
BU	1 (condition and target evaluation)	2	*bne*

e. A circular reorder buffer (ROB) with NR=16 entries is used with NB=4 Common Data Busses (CDB) connecting the WB stage and the ROB to the reservation stations and the register file. You have to design the policy to resolve contention between the ROB and the WB stage on the CDB busses.

f. You need to perform register renaming to eliminate the false dependences in the decode stage. Assuming we have a total of 32 physical registers (p0, p1, p2, …p31). You will need to implement a mapping table and a free list of the physical register as we discussed in class. Also, assuming that all of the physical registers can be used by either integer or floating point instructions.

g. A dedicated/separate ALU is used for the effective address calculation in the branch unit (BU) and simultaneously, a special hardware is used to evaluate the branch condition. Also, a dedicated/separate ALU is used for the effective address calculation in the load/store unit. You will also need to implement forwarding in your simulation design.

The simulator should be parameterized so that one can experiment with different values of NF, NI, NW, NR and NB (either through command line arguments or reading a configuration file). To simplify the simulation, we will assume that the instruction cache line contains NF instructions and that the entire program fits in the instruction cache (i.e., it always takes one cycle to read a cache line). Also, the data cache (single ported) is very large so that writing or reading a word into the data cache always takes one cycle (i.e., eliminating the cache effect in memory accesses).

Your simulation should keep statistics about the number of execution cycles, the number of times computations has stalled because 1) the reservation stations of a given unit are occupied, 2) the reorder buffers are full. You should also keep track of the utilization of the CDB busses. This may help identify the bottlenecks of the architecture.

You simulation should be both functional and timing correct. For functional, we check the register and memory contents. For timing, we check the execution cycles.

Comparative analysis:

After running the benchmarks with the parameters specified above, perform the following analysis:

1) Study the effect of changing the issue and commit width to 2. That is setting NW=NB=2 rather than 4.
2) Study the effect of changing the fetch/decode width. That is setting NF = 2 rather than 4.
3) Study the effect of changing the NI to 4 instead of 16.
4) Study the effect of changing the number of reorder buffer entries. That is setting NR = 4, 8, and 32

You need to provide the results and analysis in your project report.

Project language:

You can **ONLY** choose C/C++ (highly recommended) or Python to implement your project. No other languages.

Test benchmark

Use the following as an initial benchmark (i.e. content of the input file prog.dat).

%All the registers have the initial value of 0.
%memory content in the form of address, value.
0, 111
8, 14
16, 5
24, 10
100, 2
108, 27
116, 3
124, 8
200, 12

```
addi    R1, R0, 24
addi    R2, R0, 124
fld     F2, 200(R0)
```

```
loop:  fld    F0, 0(R1)
       fmul   F0, F0, F2
       fld    F4, 0(R2)
       fadd   F0, F0, F4
       fsd    F0, 0(R2)
       addi   R1, R1, -8
       addi   R2, R2, -8
       bne    R1,$0, loop
```

(Note that this is just a testbench for you to verify your design. Your submission should support **ALL** the instructions listed in the table and you should verify and ensure the simulation correctness for different programs that use those nine instructions. When you submit your code, we will use more complicated programs (with multiple branches and all instructions in the table) to test your submission).

Project submission:

You submission will include two parts: i) code package and ii) project report

1. Code package:
   a. include all the source code files with **code comments**.
   b. have a README file 1) with the instructions to compile your source code and 2) with a description of your command line parameters/configurations and instructions of how to run your simulator.
2. Project report
   a. A figure with detailed text to describe the module design of your code. In your report, you also need to mark and list the key data structures used in your code.
   b. The results and analysis of Comparative analysis above
   c. Your discussion peers and a brief summary of your discussion if any.

Project grading:

1. We will test the **timing** and **function** of your simulator using more complicated programs consisting of the nine RISC V instructions.
2. We will ask you later to setup a demo to test your code correctness in a 1-on-1 fashion.
3. We will check your code design and credits are given to code structure, module design, and code comments.
4. We will check your report for the design details and comparative analysis.
5. Refer to syllabus for Academic Integrity violation penalties.

Note that, any violation to the course integrity and any form of cheating and copying of codes/report from the public will be reported to the department and integrity office.

Additional Note

For those who need to access departmental linux machines for the project, here is the information

log on into any of the linux machines

ritchie.cs.pitt.edu
kernighan.cs.pitt.edu
thompson.cs.pitt.edu

For example, the command: ssh <username>@ritchie.cs.pitt.edu
Note that you need first connect VPN in order to use these machines.