CS/ECE 752 ADVANCED COMPUTER ARCHITECTURE I Spring 2022
HOMEWORK # 2
(Due by **11:00 AM** on **Wednesday, Feb 23**; upload PDF to Canvas)

Contact Sagnik Basu (SBASU24@wisc.edu) for questions

1. (6 points)
Assume that in a particular program, 18% of the dynamic instructions are branches. The remaining 82% of the instructions have a CPI of 1.4. You are considering three microarchitecture designs for handling branches as below:

Design A does not have a branch predictor and has to expend 2 cycles for every branch instruction. Determine the CPI of design A.

Design B has a branch predictor with an accuracy of 92%. A correctly predicted branch requires only 1 cycle, but a mispredicted branch requires 4 cycles to execute. Determine the CPI of design B.

Design C has a branch predictor with an accuracy of 70%, and a branch execution time of 1 cycle for correct prediction and 3 cycles for misprediction. Determine the CPI of design C.

Rank the three designs in order from best to worst.

2. (12 points, 4+4+4)
Answer the following questions based on the code sequence below.

```
loop:  LD    F2, 0(R1)
       LD    F4, 0(R2)
       MULD  F2, F0, F6        # F6 is dest reg
       ADDD  F2, F4, F7        # F7 is dest reg
       ADDD  F6, F7, F2        # F2 is dest reg
       SD    F2, 20000(R1)
       ADD   R1, #8, R1        # R1 is dest reg
       ADD   R2, #8, R2        # R2 is dest reg
       BLT   R1, R4, loop
```

The latencies of the different operations are: Integer ADD, SUB = 1 cycle, Memory LD = 3 cycles, Memory SD = 2 cycles, Branches = 2 cycles, ADDD = 2 cycles, and MULTD = 4 cycles.

(a) What would be the baseline performance (in cycles, per loop iteration) of the code sequence if no new instruction execution could be initiated until the previous instruction execution had completed? Ignore front-end fetch and decode. Assume for now that execution does not stall for lack of the next

instruction, but only one instruction/cycle can be issued. Assume the branch is taken.

(b) Consider a multiple-issue design. Suppose you have two execution pipelines, each capable of beginning execution of one instruction per cycle, and enough fetch/decode bandwidth in the front end so that it will not stall your execution. Assume results can be immediately forwarded from one execution unit to another, or to itself. Further assume that the only reason an execution pipeline would stall is to observe a true data dependence. Now how many cycles does the loop require?

(c) In the multiple-issue design of (b), you may have recognized some subtle issues. Even though the two pipelines have the exact same instruction repertoire, they are not identical nor interchangeable, because there is an implicit ordering between them that must reflect the ordering of the instructions in the original program. If instruction N + 1 begins execution in Execution Pipe 1 at the same time that instruction N begins in Pipe 0, and N + 1 happens to require a shorter execution latency than N, then N + 1 will complete before N(even though program ordering would have implied otherwise). Recite at least two reasons why that could be problematic and will require special considerations in the microarchitecture. Give an example of two instructions from the above code that demonstrate this problem.

3. (12 points) Introduction to ***gem5***
Learning gem5 book **-** ***http://learning.gem5.org/book/***

Step 1: Compile gem5

Go through the *Introduction* and *Building gem5* pages of the Learning gem5 book. Make sure to get your gem5 install working before moving onto the next step. It is advised to use Linux for this assignment.

NOTE: If you are using CSL machines (highly recommended), then the necessary dependencies are installed by default. No need to use command like “sudo apt install xxx”. You can clone the gem5 source code and use the *scons* command below to build gem5. (“X” in the below command is the number of CPU cores in the machine.)

NOTE: It is recommended to clone a stable release version of gem5.

NOTE: There is one small issue with the compilation command in the Learning gem5 book: it will not compile the *MinorCPU* model by default. Use the following command instead:

scons build/X86/gem5.opt -jX
CPU_MODELS=AtomicSimpleCPU,TimingSimpleCPU,O3CPU,MinorCPU

Step 2: gem5 Book, Part I

For this assignment, the most important parts of the Learning gem5 book are:

- downloading and building gem5,
- creating a simple configuration script,
- how to run gem5. [Note you will be using a simple “hello” executable]
- adding some complexity to your first script by adding a two-level cache hierarchy,
- how to parse the gem5 output and understand the statistics, and
- using the default configuration scripts (**se.py**).

NOTE: You are also expected to explore the following folders of gem5 :

- m5out [analyze stats after running a workload]
- configs/learning_gem5/ [sample gem5 configuration scripts]

Step 3: Exercise

In this part, we will run applications from the GAP (Graph Algorithms Platform) benchmark suite on gem5 using ***se.py*** config file.

- Change directories to your gem5 directory.
- Then, clone the gapbs repository from *https://github.com/sbeamer/gapbs* using the *git clone* command.
- Go to *gapbs/* directory and run “*make”* command. This should generate the binaries for all applications.
- You can run “*make test”* to verify if the tests pass on your build.

You will perform a simple characterization of one of
(a) bfs, with arguments “-g 15 -n 1” [if your last name begins with A-I]
(b) bc, with arguments “-g 20 -n 1” [if your last name begins with J-Q]
(c) sssp, with arguments “-g 17 -n 1” [if your last name begins with R-Z]

NOTE: you can pass arguments to the test programs from your gem5 configuration scripts! See the examples in gem5/configs/learning_gem5/ and go through the comments on each script.

NOTE: **se.py** has lots of command line arguments which needs to be explored. To get the list, use : *build/X86/gem5.opt configs/example/se.py -h*

Submit the following.
Report the IPC and the cache hierarchy miss rates for:

Cache hierarchy for parts (a) and (b):
L1 instruction cache: 16KB 2-way set associative with 64B lines
L1 data cache: 16KB 2-way set associative with 64B lines
L2 cache: 128KB 8-way set associative L2 cache with 64B lines

(a) Simulate using the in-order CPU (**MinorCPU**) for the cache hierarchy
(b) Simulate using the out-of-order CPU (**DerivO3CPU**) for the cache hierarchy
(c) Simulate using the out-of-order CPU (**DerivO3CPU**), and changing the cache hierarchy to the following setting:

Two Level Cache:
L1 instruction cache: 32 KB 4-way set associative with 64B lines
L1 data cache: 32 KB 4-way set associative with 64B lines
L2 cache: 256 KB 8-way set associative L2 cache with 64B lines

(d) How much time (roughly) did the simulations take for (a), (b) and (c)?

For all of the above tests, fast-forward 100M instructions, and simulate for next 50M instructions. The relevant config parameters can be found in configs/common/Options.py

NOTE(Optional) : you can also refer to the *two_level.py* script in "learning_gem5/part1/" folder on how to add caches. To play with cache associativity and size, see the *caches.py* script in the same path.

Optional resources :

1) Learning gem5 - Part 1
2) CPU models in gem5