CS/ECE 752 ADVANCED COMPUTER ARCHITECTURE I Spring 2022
HOMEWORK # 3
(Due by **11:00 AM** on **Wednesday, March 23**; upload PDF to Canvas)

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

1. (**30 points**) Gem5

For this problem you will be implementing a 3-bit branch predictor. A 3-bit predictor has eight states and it is a natural extension of the 2-bit predictor.

Edit *2bit_local.cc* and *BranchPredictor.py* to support the 3-bit prediction scheme with BHT consisting of 512 entries (number of sets in the local predictor table). For each entry in the prediction buffer, the state of the branch predictor entries should be set to an initial value of '**000**' (strongly not taken). The predictor is going to predict dynamic values from '**000**' through '**010**' (3 values) as not taken, and '**011**' through '**111**' (5 values) as taken.

(a) Using out-of-order CPU model (DerivO3), run the 3-bit predictor with gem5's default configuration (*se.py*). Use one of the three benchmarks bfs, bc and sssp, depending upon your name, as in HW2.

Run 50M instructions for your benchmark and mention the benchmark that you use.

Similarly, run the built-in 2-bit and ltage prediction schemes. Compare and comment on your results of the three policies in terms of IPC and the miss rates at the L1 cache.

(b) Include a print-out of the changes to *2bit_local.cc* and *BranchPredictor.py* with your changes highlighted.

(c) Now implement another predictor which selects the entry in the 512-entry prediction table by EXORing bits 17-9 of the branch instruction address with the low-order bits (bits 8-0). Repeat parts (a) and (b) above.

2. (**30 points**)
Consider the following piece of code.

1. ldf Y(r2),f2
2. mulf f2,f3,f0 ; <f0 = f2*f3>
3. ldf X(r1),f1
4. addf f1,f0,f2 ; <f2 = f1 + f0>
5. addi r1,4,r1
6. addi r2,4,r2
7. stf f2,X(r1)
8. slt r1,r3,r4

Assume that the above code is being run on MIPS R10K like out-of-order processors.

(a) Construct and fill out the following tables after 10 cycles and in the last cycle of the program's execution.

1) ROB
2) Reservation Stations
3) Map Table
4) Free List

(b) For how many cycles does this program run?

Assumptions:
1) The processor can dispatch, issue, complete and retire one instruction at a time.
2) Floating point multiplication operations take 4 cycles to execute, while floating point add takes 2 cycles (this is only the number of execute cycles and does not include the “complete” cycle).
3) All load operations take 2 cycles and store operations take 2 cycles to execute (this is only the number of execute cycles and does not include the “complete” cycle).
4) Integer operations take 1 cycle to execute (this is only the number of execute cycles and does not include the “complete” cycle).
5) Functional units are fully pipelined.
6) There are only 16 physical registers and they can be used to store both floating point and integer values.
7) There are only 8 ROB slots available and 5 reservation stations.

The tables' status at the end of cycle 2 is as below (note the column for Retire (R) also):

ROB:

ht	#	Inst	T	Told	S	X	C	R
h	1	ldf Y(r2),f2	PR#9	PR#2	c2			
t	2	mulf f2,f3,f0	PR#10	PR#1				
	3							

Reservation Stations:

#	FU	Busy	op	T	T1	T2
1	ALU	N				
2	L/S1	Y	ldf	PR#9	-	PR#5+
3	L/S2	N				
4	FP1	Y	mulf	PR#10	PR#9	PR#4+
5	FP2	N				

Map Table:

f0	PR#10
f1	PR#2+
f2	PR#9
f3	PR#4+
r1	PR#5+
r2	PR#6+
r3	PR#7+
r4	PR#8+

Free List:

PR#11, PR#12, PR#13, PR#14, PR#15, PR#16