CS/ECE 752 Spring 2024: HOMEWORK # 3
(Due by **11:00 AM** on **Monday, March 18**, via upload of PDF to Canvas)

Contact Rajesh Srivatsav (rsuresh6@wisc.edu)

1. Problem 1 (30 points)

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 (implemented as the bimodal predictor in ChampSim).

Edit ***bimodal.cc*** to support the 3-bit prediction scheme with BHT consisting of 16K entries (number of sets in the local predictor table, 16K is already the default). 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 '**011**' (4 values) as not taken, and '**100**' through '**111**' (4 values) as taken.

(a) Run the 3-bit predictor with ChampSim's default configuration (*champsim.json*). Use all of the three benchmarks: 483.xalancbmk-127B.champsimtrace.xz, 459.GemsFDTD-1169B.champsimtrace.xz and 434.zeusmp-10B.champsimtrace.xz

Run 50M instructions for each benchmark (after a warmup of 1M instructions as in HW2).

Similarly, run the built-in 2-bit and hashed perceptron branch prediction schemes. Compare and comment on your results of the three policies in terms of IPC and the branch MPKI.

(b) Include a print-out of the changes to ***bimodal.cc*** and ***champsim.json*** with your changes highlighted.

(c) Now modify the bimodal predictor implementing a 2-bit prediction scheme **with the BHT consisting of 512 entries (number of sets in the local predictor table)** and 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. Problem 2 (30 points)
Consider the following piece of code.

```
loop:
        ldf X(r1),f1
        addf f1,f3, f0 ;   <f0 = f1 + f3>
        ldf X(r2),f1
        mulf f1, f0, f2    <f2 = f0*f1>
        addi r1,4,r1
        stf f2,X(r1)
        slt r1,r3,r4
        BNQZ r4,loop
```

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 12 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 5 cycles to execute, while floating point add takes 3 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 status of the tables 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 X(r1),f1	PR#9	PR#2	c2			
t	2	addf f1,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	addf	PR#10	PR#9	PR#4+
5	FP2	N				

Map Table:

f0	PR#10
f1	PR#9
f2	PR#3+
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