Tutorial- Branch prediction and hazards

Q1: Control hazards

- LW R1, 0(R4) ; R1 ← address (0+R4)
- ADDI R2, R1, #8 ; R2 ← R1+8
- BEQ R3, R1, loop ;
- SW R3, 4(R2) ; address(4+R2) ← R3
- ADDI R2, R1, #8 ; R2 ← R1+8
- ADD R2, R1,R1 ;
- Loop: SUB R1, R2, R3

- Assume that BEQ is resolved in EX stage, and the processor is is always fetching instructions PC+4 (not taken branch). Write the pipeline diagram assuming R3=R1 (at the BEQ step). How many instruction flushes are needed?
- Assume forwarding, no code-reordering

Q2

Assume that we have a 6 stage processor pipeline:

IF, ID, EX1, EX2, MEM, WB

Forwarding is enabled.

Branch is resolved in EX2

Processor keeps fetching PC+4 always.

We execute the following set of instructions:

```
ADD t1, t2, t0
BEQ t1, t2, target
LW t2, 0(t0)
SUB t1, t2, t3
ADD t1, t2, t0
..
..
target: SUB t1, t2, t3
```

At some point in time, the processor finds out that t1= t2.

Does the processor need to flush the pipeline? If yes, how many instructions does it need to flush. If no, write 0 as the answer

Q3

The following set of instructions work well on a non-pipelined processor. But, when we create a 5-stage pipeline, register t1 gets the wrong answer. How do you fix this error?

Assume that we CANNOT write and read registers in the same clock cycle. (We cannot write in first half of the cycle, and read in the second half)

ADDI t0 t7 1

ADD t2 t0 t3

SUB t1 t2 t3

Select one or more:

- [ ] a. Insert 2 NOPs (No-operation instruction) after Instruction 1, and after Instruction 2
- [ ] b. Forward from execute to ID stage:Instruction 2 to 3 only
- [ ] c. Insert 1 NOP (No-operation instruction) after Instruction 2 only
- [ ] d. Forward from execute to ID stage: Instruction 1 to 2 only
- [ ] e. Forward from execute to execute stage: Instruction 1 to 2, Instruction 2 to 3
- [ ] f. Insert 3 NOPs (No-operation instruction) after Instruction 1, and after Instruction 2

Q4

Assume that the breakdown of dynamic instructions into various instruction categories is as follows:

	R-Type	BEQ	JMP	LW	SW
a.	40%	25%	5%	25%	5%
b.	60%	8%	2%	20%	10%

Also, assume the following branch predictor accuracies:

	Always-Taken	Always-Not-Taken	2-Bit
a.	45%	55%	85%
b.	65%	35%	98%

a. Stall cycles due to mispredicted branches increase the CPI.
What is the extra CPI due to mispredicted branches with the always-taken predictor? Assume that branch outcomes are determined in the EX stage, that there are no data hazards, and that no delay slots are used.

b. Repeat (a) for the “always-not-taken” predictor.

Predictor accuracy

Q5

- T, T, NT, NT
  - What is the accuracy of always-taken and always-not-taken predictors for this sequence of branch outcomes?
  - What is the accuracy of the two-bit predictor in this pattern, assuming that the predictor starts off in the SNT state

Q6

```
back:
LW R4, 0(R3)
ADDI R3, R3, 4
SUBI R1, R1, 1
branch1:
BEQZ R4, branch2 //branch if R4==0
ADDI R2, R2, 1
branch2:
BNEZ R1, back //branch on not equal to zero
```

- Let the initial value of R1 be n=8
- Let the initial value of R2 be 0 (R2 holds the result of the program).
- Let the initial value of R3 be p (p is a pointer to the beginning of an array of 32-bit data in memory).
- The processor uses a 2 bit counter based branch prediction which is initiated to the Weakly not taken state.

...cont on next page

We provide the following inputs to the program:
N=8, p[0] = 1, p[1] = 0, p[2] = 1, p[3] = 0,... etc. So, the array elements exhibit an alternating pattern of 1's and 0's.

Fill out the following table.

What is the prediction accuracy of the predictor for branch 1, for this sequence? Assume that branch2 always gets predicted correctly.

R1 or n	R3	R4	R2	Prediction for branch1	Actual branch behavior for branch1	Misprediction?
				WNT		

Q7

Q4: Consider the following instructions. Assume that the initial values for R1, R2, and R3 are all 0.

loop: SUBI R2, R1, 2

BNEZ1 R2, target1

ADDI R3, R3, 1

target1: ADDI R1, R1, 1

SUBI R4, R1, 3

BNEZ2 R4, loop

a. Assume that we have a 1-bit branch predictor for each branch instruction. The predictor that uses the result of the previous branch and makes the prediction based on the result, i.e., if the BNEZ1 was taken, then the predictor for BNEZ2 predicts “taken”, and similarly for “not-taken”.

Also, if the BNEZ2 was taken, then the predictor for BNEZ1 predicts “taken”, and similarly for “not-taken”. Show the results of all predictions for 3 iterations of the entire loop. Assume an initial state of “Not taken for the predictor” (6 marks)

b. Assume that we have a 2-bit branch predictor for each branch. Show the results of all predictions for three iterations of the loop. Assume we start with a “Weakly not taken state”. (6 marks)