2. [12 points] Caches & Branch Prediction

(a) [1 point] What is one advantage of a *write-back* cache over a *write-through* cache?

(b) [1 point] What is one advantage of a *write-through* cache over a *write-back* cache?

(c) [3 points] Describe a cache design (giving the total size, associativity and block size) that has a tag array that is *the same size* as the data array. Assume that addresses are 32 bits in size.

(d) [2 points] What are the two purposes that a BTB serves?

i) Records targets of taken branches

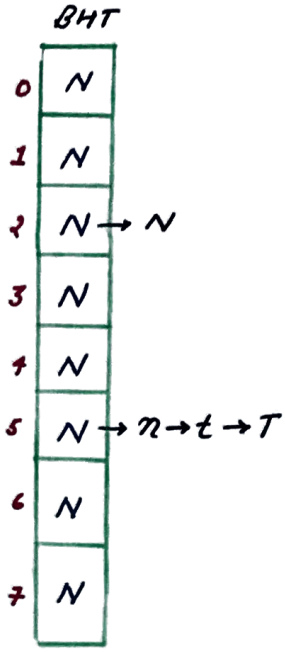
ii) Identifies (predicts) if it is a branch instruction or not.

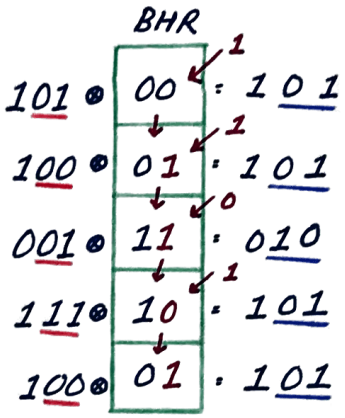
(e) [5 points] Consider a gshare branch direction predictor that has eight 2-bit saturating counters and a 2-bit history register. All counters are initialized to the strongly not-taken state (N), and the history is initialized to 00 (NN).

For each branch PC (given in binary) in the sequence below, give the predicted direction for each branch. Assume that these instructions come from a variable-length ISA where program counters are not necessarily aligned to any boundary.

branch PC	actual direction	predicted direction
1000 1101	T	N
0000 0100	T	N
0101 1001	N	N
1010 1111	T	T
0000 1100	T	T

• solution on next page


BHT
0 N
1 N
2 N → N
3 N
4 N
5 N → n → t → T
6 N
7 N


BHR
101 ⊗ 00 : 101
1
100 ⊗ 01 : 101
1
001 ⊗ 11 : 010
0
111 ⊗ 10 : 101
1
100 ⊗ 01 : 101

3. [8 points] Branch Prediction

(a) [1 points] Name one advantage that a 2-Level Adaptive Branch Predictor provides over a gshare predictor.

2-Level Adaptive Branch Predictors can take advantage of LOCAL BRANCH HISTORY whereas GShare is designed for GLOBAL BRANCH History only.

(b) [1 points] Give a reason why it would be a bad idea to use 5-bit saturating counters in a branch predictor.

Takes too long to train

(c) [3 points] In a gshare predictor, not all history bits are equally useful for predicting branches. How might you identify branches that are less useful so that they can be omitted from the history?

A "chooser" may be used to indicate situations when a branch is highly biased (it triggers the use of a simple Bimodal Branch Predictor, and it turns out to be effective).

(d) [3 points] Consider a bimodal branch predictor that has sixteen 2-bit saturating counters. All counters are initialized to the strongly not-taken state (N).

For each branch PC (given in binary) in the sequence below, give the predicted direction for each branch. Assume that these instructions come from a variable-length ISA where program counters are not necessarily aligned to any boundary.

branch PC	actual direction	predicted direction
1000 1101	T	N
0000 0011	N	N
0101 1101	T	N n
1010 1101	T	T t
0000 1001	T	N
1111 0011	T	N n

(i) Which bits in the PC are used as index bits?

There are 16 elements in the 2-bit saturating counter BHT. They will require 4 bits to represent. The PC's are not aligned. Therefore the 4 LSB are used as index bits.

4. [16 points] Branch Prediction

(a) A gshare branch predictor with a zero-bit history is also called what?

Bimodal Predictor

(b) What are the three questions a branch prediction mechanism must answer on each prediction? For each question, list the hardware structure that provides the answer.

i) Is the instruction a Branch?
BTB + RAS

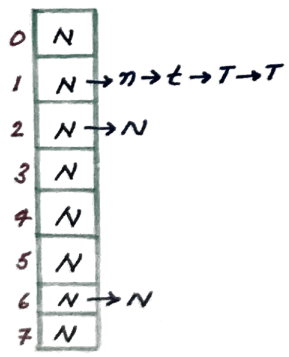
ii) Is it taken or not taken?
BHT, BHR

iii) If it is taken, where is it going?
BTB

(c) Consider a bimodal branch predictor that has eight 2-bit saturating counters. All counters are initialized to the *strongly not-taken* state (N).

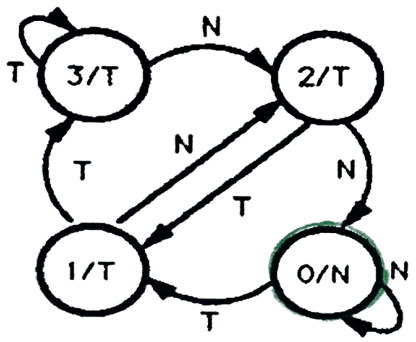
For each branch PC (given in binary) in the sequence below, give the predicted direction for each branch. Assume that these instructions come from a variable-length ISA where program counters are not necessarily aligned to any boundary.

branch PC	actual direction	predicted direction
1100 0001	T	N
0100 1010	N	N
1100 0001	T	N
0010 0110	N	N
1100 0001	T	T
1100 1001	T	T

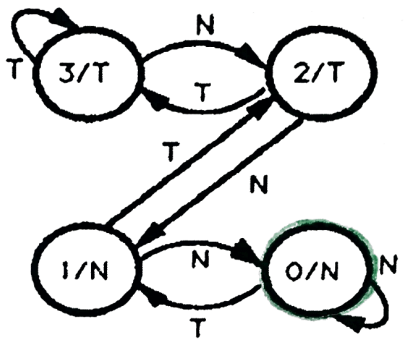


(d) Below are diagrams of two of the automata from the paper "Two-Level Adaptive Training Branch Prediction" paper by Yeh and Patt. Note that Automaton A2 is a standard 2-bit saturating counter.

Branch directions on edges indicate state transitions taken when the labeled direction is seen. Branch directions inside each numbered state indicate the prediction given for an automaton in that state (e.g., "2/T" means Taken is predicted when in state 2). Each automaton is initialized to the 0 state.



Automaton A1



Automaton A2
(2-bit Saturating Up-down Counter)

i) Give a sequence of four branch directions that will result in more correct predictions with Automaton A1 than with A2.

T T N T

- 4 mispredictions with A2
- 2 mispredictions with A1

Give a sequence of four branch directions that will result in more correct predictions with Automaton A2 than with A1.

T N N N

- 1 misprediction with A2
- 3 mispredictions with A1

(i) Are these in the syllabus?
Not on their own, but in a question like this - maybe.

4. [7 points] Pipelining and Branch Prediction

Dr. Chip D. Signer has come up with a new optimization for the classic 5-stage pipelined processor. His proposal is to detect certain kinds of branch mispredictions during Decode instead of during Execute.

Consider the standard execution of two insns, J and K, in the pipeline diagram below. Suppose that, during cycle 2, J is predicted to be a taken branch to location Q, and insn Q is fetched accordingly. In fact, however, J is not a branch insn at all. During cycle 3 the mistake is discovered and the correct insn, K, is fetched in cycle 4.

insn	1	2	3	4	5	6	7	8
J	F	D	X	M	W			
Q		*F*	*D*					
K				F	D	X	M	W

Dr. Signer's proposal is "fast misprediction detection" (FMD) to detect this situation sooner, during cycle 2 when J is decoded and discovered not to be a branch. This allows the correct insn, K, to be fetched one cycle earlier, as shown below.

insn	1	2	3	4	5	6	7	8
J	F	D	X	M	W			
Q		*F*						
K			F	D	X	M	W	

(a) Is FMD a good idea? Why or why not?

(i) Bad idea
(ii) 'J' wouldn't have a BTB entry so the branch would not have been taken

• (i) How does this contrast to Fast Branching where we perform simple comparisons in Decode?
→ Different because here J is the branch

(b) Even assuming FMD is a good idea, what is one potential negative performance impact of implementing FMD as described?

more latency in the decode stage

5. [9 points] Branch Prediction

(a) Branch prediction is a form of speculation, and must provide mechanisms to satisfy the three required criteria of speculation (posed as questions below). For each question, give the answer used in branch prediction.

i) When should speculation occur?

Fetch

ii) How is speculation validated?

During execution stage

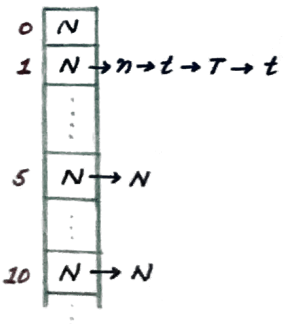
iii) On a mis-speculation, how is correctness recovered?

(i) Flush Fetch & Decode
(ii) Insert no-ops

(b) Consider a bimodal branch direction predictor that has sixteen 2-bit saturating counters. All counters are initialized to the *strongly not-taken* state (N).

For each branch PC (given in binary) in the sequence below, give the predicted direction for each branch. Assume that these instructions come from a variable-length ISA where PCs are not necessarily aligned to any boundary.

branch PC	actual direction	predicted direction
1100 0001	T	N
0100 1010	N	N
1100 0001	T	N
0010 0001	T	T
1101 0101	N	N
1100 0001	N	T



2. [6 points] Branch Prediction

(a) Consider two different branch predictor configurations for an 8-bit processor with 4B insns.

i) [2 points] Give two branch **addresses**, A and B, in binary and their fixed **directions** (e.g., A is always taken, and B is never taken) such that if A and B are executed repeatedly in sequence (ABABAB...) a 100% branch misprediction rate will result. Assume a **bimodal** branch predictor with 8 BHT entries where each BHT entry is a 2-bit saturating counter initialized to the **weakly-not-taken (n)** state.

0000 0000 T
1000 0000 N

- (i) How do you pick index bits?
  - 4B instructions will need $2^2 \Rightarrow$ 2 bit skipped bits
  - 8 BHT entries will need $2^3 \Rightarrow$ 3 bits

ii) [4 points] Give two branch addresses and directions, as above, that exhibit **100%** mispredictions with a **gshare** branch predictor, with 8 BHT entries of 2-bit saturating counters initialized to the **weakly-not-taken (n)** state, and a 1-bit BHR initialized to 0 (not taken).

0000 0000 T
1000 0100 N

- (i) same question as (a)(i)
  same answer as above

5.) An Olive Branch (7 points total)

5.1) (4 points) List three branch prediction prediction approaches, ordered from least accurate (a) to most accurate (c) for a typical workload. If you need to correct the order after you give the answer, **clearly write** (a) next to your least accurate method, (b) next to the middle method, and (c) next to the most accurate method.

(a) Static predictor (always T or always NT)

(b) 1-bit predictor

(c) 2-bit predictor

(d) GShare

(e) Tournament Predictor (Hybrid)

5.2) (3 points) State what each of the following acronyms stand for, and, **in 15 words or less**, describe it. If the acronym has nothing to do with branch prediction, just write "unrelated" (stating what unrelated acronyms stand for is optional).

(a) BBB -

(b) BBS -

(c) BHT Branch History Table

(d) BTB Branch Target Buffer

(e) RAS Return Address Stack

(f) RSA -

6.) Not an Olive Branch (10 points total)

Fill in the following branch prediction table. For the 2-bit saturating counter, assume all branch instructions share a single PC. Assume the previous three outcomes were taken, taken, not taken. The initial states of the predictors have been filled in for you on the first row, but you need to add the predictions.

Branch Outcome	2-bit counter (states are T, t, n, N)		3-bit correlated predictor (history of last three branches)								
	State	Prediction	State								Prediction
			NNN	NNT	NTN	NTT	TNN	TNT	TTN	TTT	
T	t	T	N	T	N	T	N	T	N	T	N
N	T	T									
N	t	T									
N	n	N									
T	N	N									
T	n	N									
N	t	T									
N	n	N									
T	N	N									
T	n	N									
N	t	T									

(i) How to solve this?
problems like this will not be asked

4. [17 points] Branch Prediction

Dr. Pipi Line is considering a simplified branch predictor design that eliminates the BTB, relying on just the branch history table (BHT) instead. Non-branch insns will be encoded as "not-taken" entries, since they have the same behavior as not-taken branches, namely, going to the next sequential PC.

(a) Give one good aspect of Dr. Line's design.

[2 points] By eliminating the BTB, the Fetch stage is simplified and can use less energy. The overall pipeline frequency may increase if Fetch was the slowest stage.

(b) Give one bad aspect of Dr. Line's design.

[2 points] Without the BTB, there will be large amounts of aliasing between branches and non-branches in the BHT, likely leading to almost all of the BHT entries being not-taken and causing poor branch prediction accuracy.

(c) Overall, do you think Dr. Line's proposal is a good or bad idea? Why?

[3 points] The BHT is not very useful without a BTB, and branch prediction is critical for good performance, so the BTB's utility outweighs its complexity.

(d) Consider a *gshare* branch direction predictor that has eight 2-bit saturating counters in the BHT and a 3-bit branch history register. All counters are initialized to the *strongly not-taken* state (N). This predictor is used in a RISC ISA with 4B insns, and PCs are 8 bits.

Fill in a sequence of branch PCs and actual directions that result in the given predictions.
[10 points]

branch PC	actual direction	predicted direction
0000 0000	T	N
0000 0100	T	N
0000 1100	T	T
0001 1100	T	T
0000 0000	T	N

4. [19 points] Branch Prediction

(a) Why does a Branch Target Buffer (BTB) use tags instead of a valid bit for each entry?

(i) valid bits would indicate if it is a branch or not

(ii) but they do not hint at whether the target is associated with the current PC or some other aliased instruction

(b) A chip designer has a bright idea: since 2-bit saturating counters work so great in branch predictors, why not use 3-bit saturating counters instead! Assuming all counters are initialized to the **weakest taken state**: N → nn → nnn → nnnn → tttt → ttt → tt → T

i) Give a short sequence of branches (6 branches or fewer) under which 3-bit counters are **better than** 2-bit counters.

if initialized at "n" "t"

T T N N T T

2-bit will have 3 mispredictions

3-bit will have 2 mispredictions

ii) Give another short sequence under which 3-bit counters are **worse than** 2-bit.

T T N N N

2-bit will have 2 mispredictions

3-bit will have 3 mispredictions

(c) You're a microprocessor designer, and your trace-based simulations of an important workload indicate that branch instructions at the following two 32-bit addresses (in binary) are executed frequently. Assume that insns are variable-length (like x86).

- `Address A: 0010 1100 1111 1010 0111 1001 0101 1101`
- `Address B: 1011 1010 0001 1110 1111 1001 0101 1101`

The above two instructions are likely to conflict (hash to the same entry) in a branch predictor.

i) For a simple bimodal predictor with **two-bit saturating counters**, how many entries must the predictor have to prevent these two branches from interfering (conflicting) with each other?

(i) index bits would need to be at least 16 since first 15-bits conflict

(ii) 2^{16} entries

ii) How much storage does the predictor use (in bits)?

2^{16} entries × 2 bits each

= 2^{17} bits

(d) Consider a **gshare** branch direction predictor that has eight 2-bit saturating counters in the BHT and a 3-bit branch history register. All counters are initialized to the **weakly not-taken** state (n), and the BHR is initialized to all zeroes (**not-taken**). This predictor is used in a RISC ISA with 2B insns, and PCs are 8 bits in size.

Fill in a sequence of branch PCs and actual directions that result in the given predictions.

branch PC	actual direction	predicted direction
0000 0000	T	N
0000 0010	T	T
0000 0000	T	N
0000 1110	T	T
0000 0000	T	N

• (i) How to select index bits?

2B insns = 2^1 ⇒ 1 bit skip

8 BHT entries = 2^3 ⇒ 3 bit index

6.) Some More Arm Twisting (6 points total)

Having hastily replicated Apple's design, incorporated the variant you determined was fastest, and rolled out a massive ad campaign trumpeting the improvements, Qualcomm discovered a second page of notes in the middle of a 600-page iTunes end-user license agreement concerning branch prediction in the different variants. It turns out Apple's final A371 design has no mispredict penalty, so it doesn't even need a branch predictor. But all the variants *do* have mispredict penalties, and include a branch predictor. There is just enough time to add a branch predictor into the design, but not to beat Apple's predictor performance. For each of the variants, calculate the revised CPI including the new information.

6.1) (2 points) Variant 1: 5 cycle mispredict, with 50% predictor accuracy

$CPI = CPI_{variant\,1} + 5 \times 0.5 \times 0.5$
$= 3.30 + 1.25$
$= 4.55$

cycle penalty = 5
branches = 50%
accuracy = 50%.

6.2) (2 points) Variant 2: 10 cycle mispredict, with 90% prediction accuracy:

$CPI = CPI_{variant\,2} + 10 \times 0.1 \times 0.5$
$= 2.80 + 0.50$
$= 3.30$

cycle penalty = 10
branches = 50%.
accuracy = 90%

6.3) (2 points) Variant 3: 100 cycle mispredict, with 100% predictor accuracy

$CPI = CPI_{variant\,3} + 100 \times 0.5 \times 0$
$= 3.20 + 0$
$= 3.20$

cycle penalty = 100
branches = 50%
accuracy = 100%.

5. **[11 points] Branch Prediction**

(a) During what pipeline stage does branch prediction make its speculative decisions?

Fetch Stage

(b) How is the speculation from branch prediction validated?

Execute Stage - all information needed is now available

(c) On a mis-speculation, how is correctness restored?

(i) Flush Fetch Stage and Decode Stage
(ii) Insert no-ops

(d) Consider a **bimodal** branch direction predictor that has eight 2-bit saturating counters in the BHT. All counters are initialized to the **weakly not-taken** state (n). This predictor is used in an LC4 processor.

Fill in a sequence of branch PCs and actual directions that, if fed to the direction predictor without any other inputs, would result in the given predictions.

branch PC	actual direction	predicted direction
0000 1000	T	N (n)
0000 1000	T	T (t)
0000 0101	T	N (n)
0000 1000	T	T
0001 1100	T	N (n)

(i) Which bits of the PC are used to index the BHT?