Last (family) name: ____Solution_____

First (given) name: _____

Student I.D. #: _____

Department of Electrical and Computer Engineering
University of Wisconsin - Madison

ECE/CS 752 Advanced Computer Architecture I

Midterm Exam 1

Wednesday, October 25, 2017

Instructions:

1. Open book/open notes.
2. You may use a calculator, but no phones or laptops.
3. **You must show your complete work**. Points will be awarded only based on what appears in your answers.
4. Failure to follow instructions may result in forfeiture of your exam and will be handled according to UWS 14 Academic misconduct procedures.

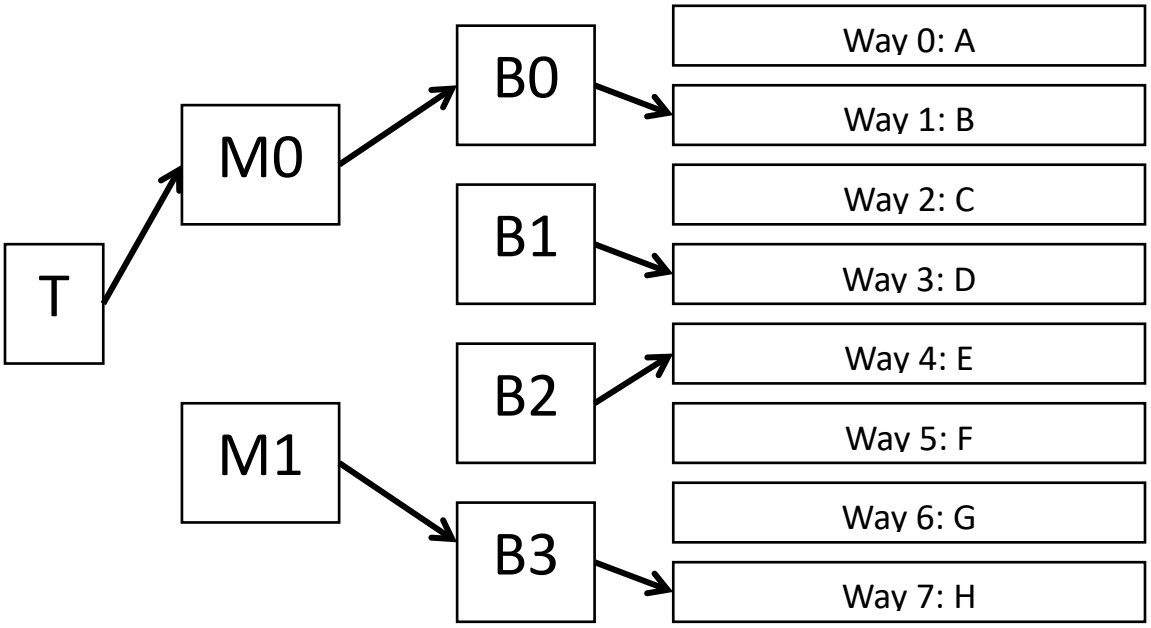
Problem	*Type*	*Points*	*Score*
1-20	Buzzword Bingo, 2 pts each	40	
21	Replacement Policies	20	
22	TAGE vs Neural Branch Prediction	20	
23	Register Renaming	20	
Total		100	

<u>Problems 1-20: (40 pts)</u>: Buzzword Bingo; match each definition to its buzzword

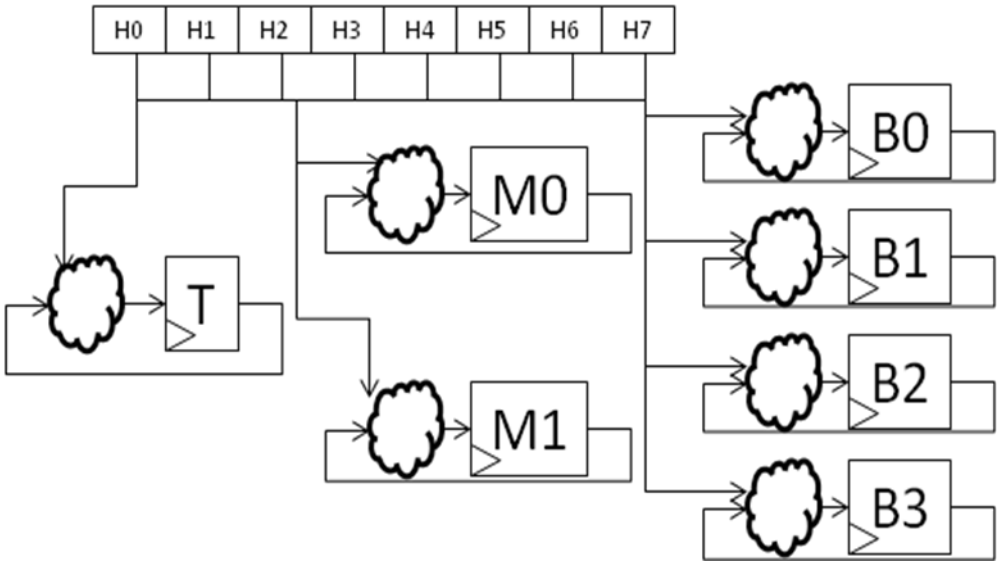
Definition		**Buzzword**	
1. _V_	Keeps prefetches from polluting the cache	A.	MSHR
2. _I_	Very accurate for a branch that is always taken	B.	HPS
3. _S_	Tells the front end to fetch instructions from the call site	C.	Stride Prefetching
4. _M_	These enable fast recovery from branch mispredictions	D.	Future File
5. _C_	Hardware technique that reads blocks from memory before they are needed	E.	Physical Register File
6. _X_	Enables precise exceptions	F.	Wakeup logic
7. L,Y_	Prevents the instruction cache from delivering the peak number of instructions per access	G.	Select logic
8. _G_	Picks which of multiple ready instructions can issue next	H.	Local branch history
9. _N_	Replacement policy that evicts block with next reference furthest in the future	I.	Bimodal predictor
10. _A_	Keeps track of a pending cache miss	J.	Global branch history
11. _E,R	Used in the RS/6000 (RIOS-I) floating point unit	K.	Bypass network
12. _J_	Useful for predicting branches that correlate with other branches	L.	Fetch address alignment
13. _P_	Keeps track of loads that have not yet committed	M.	Checkpoints
14. _W_	Improvements from integrating RF blocks and accelerators on the die	N.	Belady's optimal
15. _R_	Tracks the current location of registers	O.	Store Queue
16. _D_	Another name for rename registers	P.	Load Queue
17. _B_	Decodes complex VAX instructions into equivalent sequences of micro-ops	Q.	Pseudo-LRU
18. U,O	Needed to match an aliased load with the correct store	R.	RAM Map Table
19. _F_	Informs dependent instructions that their operands are ready	S.	Return address stack
20. _Z_	Allows loads to issue before prior store addresses are known	T.	Branch color
		U.	Store color
		V.	Stream buffer
		W.	More than Moore
		X.	Reorder buffer
		Y.	Flynn's bottleneck
		Z.	Speculative disambiguation

21. Replacement Policies [20 pts]

a) (10 pts) Assume an 8-way set-associative cache that implements the tree-based pseudo-LRU replacement policy described in lecture. Given the initial pseudo-LRU state illustrated below, simulate the trace of references given in the table, recording the PLRU block after each reference. Record the PLRU bit for each node in the tree (0 is up, 1 is down).



Reference	T	M0	M1	B0	B1	B2	B3	PLRU Block
Initial	**0**	**0**	**1**	**1**	**1**	**0**	**1**	**Way 1: B**
G	**0**	**0**	**0**	**1**	**1**	**0**	**1**	**Way 1: B**
F	**0**	**0**	**1**	**1**	**1**	**0**	**1**	**Way 1: B**
C	**1**	**0**	**1**	**1**	**1**	**0**	**1**	**Way 7: H**
E	**0**	**0**	**1**	**1**	**1**	**1**	**1**	**Way 1: B**
H	**0**	**0**	**0**	**1**	**1**	**1**	**0**	**Way 1: B**
G	**0**	**0**	**0**	**1**	**1**	**1**	**1**	**Way 1: B**
D	**1**	**0**	**0**	**1**	**0**	**1**	**1**	**Way 5: F**
E	**0**	**0**	**1**	**1**	**0**	**1**	**1**	**Way 1: B**



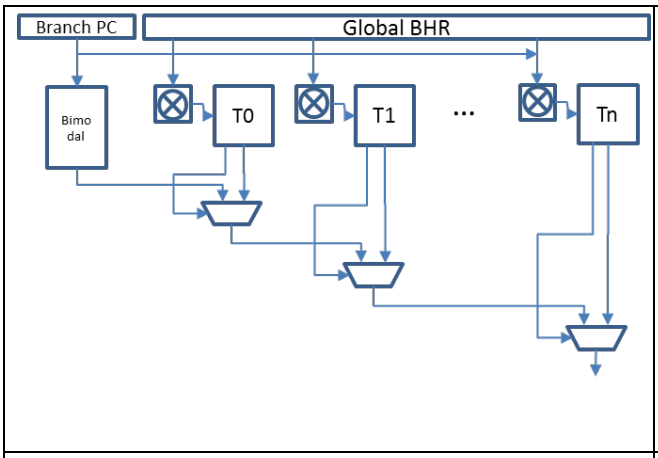
b) (10 pts) Given the logic diagram above, with inputs H0…H7 that indicate a hit in Way 0…Way 7 of this set, fill out the truth table for the next-state logic for each bit in the pseudo-LRU tree. As shown, if H0…H7 are all zero, there is no reference to this set, and no update to the PLRU state.

Input								**Next State**						
H0	**H1**	**H2**	**H3**	**H4**	**H5**	**H6**	**H7**	**T'**	**M0'**	**M1'**	**B0'**	**B1'**	**B2'**	**B3'**
0	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**T**	**M0**	**M1**	**B0**	**B1**	**B2**	**B3**
1	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**1**	**1**	**M1**	**1**	**B1**	**B2**	**B3**
0	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**1**	**1**	**M1**	**0**	**B1**	**B2**	**B3**
0	**0**	**1**	**0**	**0**	**0**	**0**	**0**	**1**	**0**	**M1**	**B0**	**1**	**B2**	**B3**
0	**0**	**0**	**1**	**0**	**0**	**0**	**0**	**1**	**0**	**M1**	**B0**	**0**	**B2**	**B3**
0	**0**	**0**	**0**	**1**	**0**	**0**	**0**	**0**	**M0**	**1**	**B0**	**B1**	**1**	**B3**
0	**0**	**0**	**0**	**0**	**1**	**0**	**0**	**0**	**M0**	**1**	**B0**	**B1**	**0**	**B3**
0	**0**	**0**	**0**	**0**	**0**	**1**	**0**	**0**	**M0**	**0**	**B0**	**B1**	**B2**	**1**
0	**0**	**0**	**0**	**0**	**0**	**0**	**1**	**0**	**M0**	**0**	**B0**	**B1**	**B2**	**0**

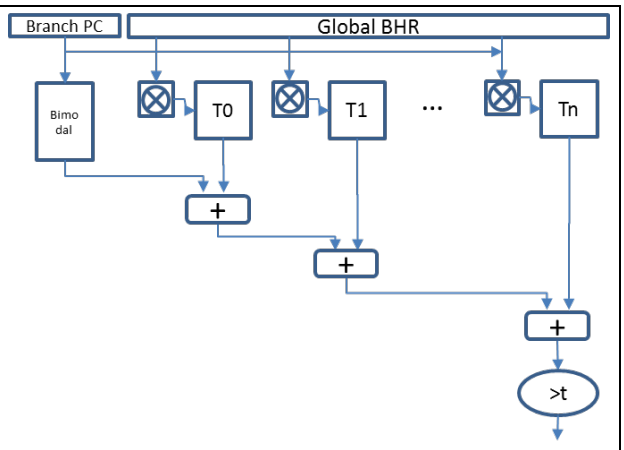
22. TAGE vs. Neural Branch Prediction [20 pts]

The two competing state-of-the-art branch predictors today are the TAGE and Neural approaches described in lecture. In this problem, you will be comparing the two approaches for prediction in terms of implementation cost and delay. Here, both predictors use multiple identical predictor elements, including a base bimodal (Smith) predictor indexed by branch address, and multiple history tables (T0-Tn) that generate predictions based on varying global branch history length. The tables T0-Tn are indexed with a hash of the global BHR and the branch address.

The key difference between these predictors is how the final prediction is determined. As shown below, the TAGE predictor relies on partial tag matches in each table and uses a priority mux to select the prediction from the longest predictor with a matching history. In contrast, the Neural predictor stores a trained multibit *response* in each predictor entry, and uses a series of adders to generate a final sum, which is compared to a learned threshold. The final prediction (taken or not-taken) shows up on the output at the bottom of each diagram below.



TAGE Predictor



Neural Predictor

To characterize these two predictors, consider the following assumptions and parameters:

- **E_b:** The number of entries **E_b** in the bimodal table is fixed for both predictors
- **E_t:** The number of entries **E_t** in each of the tables (T0-Tn) is fixed for both predictors
- **n**: There are **n** tables T0-Tn in both designs
- The TAGE bimodal table entries are two-bit saturating counters, and are not tagged
- The TAGE T0-Tn table entries also contain two-bit saturating counters, plus a 10-bit partial tag and a 3-bit confidence counter
- The Neural bimodal and T0-Tn table entries contain 5-bit learned weights, and no tags
- The size/area of each table is determined by the total number of bits it stores (capacity)
- The access delay for each table is twice the square root of its capacity (total #bits)
- **D_{hash}**: the delay for each hash function used to access the table
- **D_{tag}**: the delay of the tag matching logic in each TAGE T0-Tn table
- **D_{mux}**: the delay of each mux in the TAGE design
- **D_{add}**: the delay of each adder in the Neural design. The threshold comparator also has the same delay, since it uses an arithmetic adder/subtractor to compare the final sum to the threshold value.

(a) Area Analysis [4 pts]: in terms of the parameters described above, write an equation for each predictor that describes the total size of its tables.

TAGE: $2xE_b + (n+1) \times E_t \times (2+10+3)$

Neural: $5xE_b + (n+1) \times E_t \times 5$

(b) Delay analysis [8 pts]: in terms of the parameters described above, derive an equation for the critical delay path for each predictor, given that the inputs (the branch address and global BHR bits) are available at *t=0*. You can use a max(a,b) operator if your equation needs one. Hint: first identify the critical path through each predictor.

$D_{bimodal\text{-}TAGE} = 2 \times sqrt(2 \times E_b)$
$D_{bimodal\text{-}Neural} = 2 \times sqrt(5 \times E_b)$
$D_{TAGE} = 2 \times sqrt(13 \times E_t) + D_{hash} + D_{tag}$
$D_{Neural} = 2 \times sqrt(5 \times E_t) + D_{hash}$

TAGE: $\max(D_{bimodal\text{-}TAGE}, D_{TAGE}) + (n+1) \times D_{mux}$

Neural: $\max(D_{bimodal\text{-}Neural}, D_{Neural}) + (n+2) \times D_{add}$

(c) Delay discussion [4 pts]: Assuming that $\mathbf{D_{add}}$ is approximately equal to ($\mathbf{D_{tag}}$ + $\mathbf{D_{mux}}$), which predictor will have a lower access time? Justify your answer.

It is not obvious from what is given. We know that $D_{Neural} < D_{TAGE}$ since it is smaller and has no tag match (D_{tag}), but Neural has (n+2)xD_{add} (one extra level for the threshold comparison).

If (($D_{TAGE} - D_{Neural}$) > D_{add}) then Neural is faster, else TAGE is faster
However, if ($D_{bimodal} > D_{Neural}$) then TAGE will also be faster (unlikely, unless E_b is very large)

(d) Delay optimization [4 pts]: Is there any way you can restructure the logic shown in the diagram for either predictor to reduce the delay without changing its function or accuracy? If so, describe the change you would make and its impact on the total delay. Would this change your conclusion for part (c)?

You can use a tree adder for the Neural reduction. Now instead of (n+2) x D_{add}, we have $\log_2(n+1)$ levels of adders plus the threshold check. E.g. for n=7 we have (3+1) x D_{add} for Neural, vs. 8 x (D_{tag}+D_{mux}) for TAGE, so Neural would be faster here.

23. Register Renaming [20 pts]

The following set of instructions is to be renamed onto a set of physical registers. The initial register map and free list are given. The first instruction has been done for you. Assume that registers are allocated from the left of the free list, and that freed registers are returned to the right of the free list.

a) Show the rename mappings after each instruction has been dispatched by filling out the table below, updating the rename mappings and free list in the table appropriately. [16 pts]

Cycle		Original Instruction	Rename mappings after dispatch								Free list at end of dispatch cycle
			R1	R2	R3	R4	R5	R6	R7	R8	
Dispatched	Committed	Initial register mappings	P1	P2	P3	P4	P5	P6	P7	P8	P9, P10, P11, P12
1	3	R5 <= R1 + R5					P9				P10, P11, P12
1	3	R2 <= mem (R3 + R1)		P10							P11, P12
2	4	R1 <= R3 + 4	P11								P12
2	5	R7 <= R1 + R2							P12		<empty>
4	6	R4 <= R4 – 1				P5					P2,P1
4	6	R4 <= R4 + R8				P2					P1
6	9	R7 <= R4 – R7							P1		P7,P4,P5
6	9	R4 <= mem(R1 + R6)				P7					P4,P5
8	10	R3 <= R5 – R4			P4						P5

b) Does the allocator ever have to stall to wait for registers to be added to the free list? Explain why or why not. (2 pts)

No: P5 & P2 are freed in cycle 3, P1 is freed in cycle 4, so alloc in cycle 4 is OK. Similarly, P7 is freed in cycle 5, and P4 & P5 are freed in cycle 6, so alloc in cycles & 8 are OK

c) If the design uses RAM map table checkpoints like the MIPS R10000, for up to four branches in flight, how many bits are needed in total for the current map + 4 checkpoints? (2 pts)

(1 current table + 4 ckpt tables) x (8 rows/table) x (4 bits/row) = 160 bits