ELE 475 PS3 Problem 1 Solution

Simple Unrolling Preparation

	X	Y	Z	M	LS0	LS1
loop:					LW R6, 0(R1)	
	ADD R4, R5, R0			MUL R8, R4, R16		
	SUBI R3, R3, 1			MUL R9, R5, R17		
				MUL R10, R6, R18		
	ADDI R1, R1, 4					
	ADD R5, R6, R0					
	ADD R8, R8, R9					
	ADD R8, R8, R10					
	ADDI R2, R2, 4		BNEZ R3, loop			SW R8, 0(R2)

Unrolled three times with some code motion

	X	Y	Z	M	LS0	LS1
					LW R6, 0(R1)	
loop:		ADD R28, R28, R29		MUL R8, R4, R16		
	SUBI R3, R3, 3	ADD R28, R28, R30		MUL R9, R5, R17		
				MUL R10, R6, R18	LW R4, 4(R1)	SW R28, -8(R2)
		ADD R38, R38, R39		MUL R28, R5, R16		
		ADD R38, R38, R40		MUL R29, R6, R17		
				MUL R30, R4, R18	LW R5, 8(R1)	SW R38, -4(R2)
		ADD R8, R8, R9		MUL R38, R6, R16		
	ADDI R1, R1, 12	ADD R8, R8, R10		MUL R39, R4, R17		
	ADDI R2, R2, 12		BNEZ R3, loop	MUL R40, R5, R18	LW R6, 0(R1)	SW R8, 0(R2)

Final Code

	X	Y	Z	M	LS0	LS1
function:						
prolog:	ADD R5, R0, R0	ADDI R16, R0, 0x456	ADDI R17, R0, 0x789		LW R4, 0(R1)	
	ADD R6, R0, R0	ADDI R18, R0, 0x901		MUL R28, R5, R16		
				MUL R29, R6, R17		

		ADD R25, R5, R0		MUL R30, R4, R18	LW R5, 4(R1)	
				MUL R38, R6, R16		
	ADDI R1, R1, 8	SLTI R41, R3, 3		MUL R39, R4, R17		
		ADD R26, R6, R0	BNZ R41, epilog	MUL R40, R5, R18	LW R6, 0(R1)	
loop:		ADD R28, R28, R29		MUL R8, R4, R16		
	SUBI R3, R3, 3	ADD R28, R28, R30		MUL R9, R5, R17		
				MUL R10, R6, R18	LW R4, 4(R1)	SW R28, 0(R2)
		ADD R38, R38, R39		MUL R28, R5, R16		
		ADD R38, R38, R40		MUL R29, R6, R17		
		ADD R25, R5, R0		MUL R30, R4, R18	LW R5, 8(R1)	SW R38, 4(R2)
		ADD R8, R8, R9		MUL R38, R6, R16		
	ADDI R1, R1, 12	ADD R8, R8, R10		MUL R39, R4, R17		
	ADDI R2, R2, 12	ADD R26, R6, R0	BGTZ R3, loop	MUL R40, R5, R18	LW R6, 0(R1)	SW R8, 8(R2)
epilog:		ADD R28, R28, R29	BLEZ R3, ep_fix_0			
	SUBI R3, R3, 1	ADD R28, R28, R30				
	ADDI R2, R2, 4		BLEZ R3, ep_fix_1			SW R28, 0(R2)
		ADD R38, R38, R39				
	SUBI R3, R3, 1	ADD R38, R38, R40				
	ADDI R2, R2, 4		J ep_end			SW R38, 0(R2)
ep_fix_0:	ADD R4, R25, R0	ADD R5, R26, R0	J ep_end			
ep_fix_1	ADD R4, R26, R0	ADD R5, R4, R0				
ep_end:				MUL R8, R4, R16		
				MUL R9, R5, R17		
				MUL R10, R5, R16		
		ADD R8, R8, R9				
		ADDI R2, R2, 4				SW R8, 0(R2)
			JR R31			SW R10, 0(R2)

High Order Functionality: FIR filter

Average multiples per cycle: 1

ELE 475 PS#3 Solution

Problem # 2:

R6, R8, and R5 are all read in the shadow of being updated thus get old value. After execution, R12 = 7, R13 = 10, and R14 = 6.

```
{ADDI R6, R0, 6; ADDI R8, R0, 8; ADDI R5, R0, 5; LW R14, 8(R7);}
{LW R6, 0(R7); LW R8, 4(R7); ADDI R12, R6, 1; ADDI R13, R8, 2;}
{ADD R14, R14, R5; ADDI R9, R0, 9; ADDI R10, R0, 10;}
{MUL R5, R8, R10;}
{MUL R7, R6, R9;}
{ADD R15, R16, R17;}
{SUB R19, R18, R22;}
{NOP}
{NOP}
{ADD R5, R7, R5;}
```

Note: With register renaming, a more compact schedule could be created

The LEQ model is more flexible because it is easier to implement precise interrupts. An instruction can interrupt and the previous instructions can flow to the end of the pipeline and write the register file without creating erroneous results when the interrupted instruction is re-executed.

ELE 475 PS#3 Solution

Problem # 3:

```
ADDI R6, R0, 1
ADDI R3, R0, 50
loop:
LW R8, 0(R9)
ADDI R27, R24, 10
ADD R12, R15, R8
SUB R26, R24, R12
MOVZ R24, R27, R8
MOVN R24, R26, R8
SUBI R3, R3, 1
BNEZ R3, loop
```

Yes, it is beneficial to predicate. In the original code, the fall-through case took 4 cycles, plus an average 5 cycle mispredict penalty = 9 cycles. In the branch taken case, execution took 2 cycles plus a 5 cycle mispredict penalty = 7 cycles. Each of these outcomes has a 50% probability; therefore the average latency of the original code is 8 cycles. In the predicated case, the latency of the replaced instructions is 5 cycles.

ELE 475 PS3 Problem 4 Solution

		Iteration 1		Iteration 2		Iteration 3		Iteration 4		Iteration 5		Iteration 6		Iteration 7		Accuracy
Name	Initial	Outcome	After	Outcome	After	Outcome	After	Outcome	After	Outcome	After	Outcome	After	Outcome	After	
b_0	SNT	T	NT	NT	SNT	T	NT	NT	SNT	T	NT	NT	SNT	T	NT	0.429
b_1	SNT	x	SNT	T	NT	x	NT	T	T	x	T	NT	NT	x	NT	0
b_3	SNT	T	NT	T	T	T	ST	T	ST	T	ST	T	ST	NT	T	0.571

ELE 475 PS#3 Solution

Problem # 5:

We start with the solution to problem #4 ad see that there are 17 braches for one iteration of the outer loop with a sequence of outcomes being: T, T, NT, T, T, T, T, NT, T, T, T, T, NT, NT, T, T, NT. We analyze two of these functions p_4 executing next to each other and draw the content of the BHR for each branch:

	Branch in Program Order	BHR Old 10	9	8	7	6	5	4	3	2	1	Recent 0
New	0	T	N	T	T	T	T	N	N	T	T	N
	1	T	T	N	T	T	T	T	N	N	T	T
	2	T	T	T	N	T	T	T	T	N	N	T
	3	T	T	T	T	N	T	T	T	T	N	N
	4	N	T	T	T	T	N	T	T	T	T	N
	5	T	N	T	T	T	T	N	T	T	T	T
	6	T	T	N	T	T	T	T	N	T	T	T
	7	N	T	T	N	T	T	T	T	N	T	T
	8	T	N	T	T	N	T	T	T	T	N	T
	9	T	T	N	T	T	N	T	T	T	T	N
	10	N	T	T	N	T	T	N	T	T	T	T
	11	N	N	T	T	N	T	T	N	T	T	T
	12	T	N	N	T	T	N	T	T	N	T	T
	13	T	T	N	N	T	T	N	T	T	N	T
	14	T	T	T	N	N	T	T	N	T	T	N
	15	T	T	T	T	N	N	T	T	N	T	T
Old	16	N	T	T	T	T	N	N	T	T	N	T

Looking at this table, we see that each branch has a unique BHR for each execution of the branch inside of the loop. This means that they will each train up a unique PHT entry to have the prediction of the previous time the branch was executed. The execution on the 50th execution matched the 51st execution, therefore every branch in the 51st execution will be predicted correctly. The final state of the BHT will be from oldest to newest: T, NT, T, T, T, T, NT, NT, T, T, NT.

ELE 475 PS#3 Solution
Problem # 6:

BTBs are especially used at aggressive clock frequencies because at aggressive clock frequencies, it is common to add extra stages to the front of the pipeline which makes the point at which an instruction is decoded and known to be a branch relatively late. Also, the destination of the branch cannot be computed until at minimum when the instruction is known to be a branch, therefore this adds stall cycles for every branch until it is known that it is a branch. With a BTB, given the PC of the current instruction, the fact that the instruction is a control flow instruction and the destination of that instruction can be predicted simultaneously and not have to wait for the decode which may be several cycles later.

The destination of a branch in a 5-stage pipeline is known in the decode stage of the pipeline.

A dedicated branch address adder does not help with JALR instructions as the jump address needs to come from a register.

A BTB can aid in determining the target and that an instruction is a control flow instruction. Although it can be to limited success because JR/JALR's can jump to any address. But, for example, a leaf function is called repeatedly from the same location, a BTB will do very well at predicting the return address. This is particularly interesting for JALR's which are typically used for calls to function pointers, which may or may not be predicted well depending on consistency of the call location.