1.

1a. Cycle Time is based on the longest latency and lw will require the use of all modules.
Cycle Time = Register Read + Register Write + Data Memory + ALU + Instruction Memory

Yin/Yang: 100 + 100 + 250 + 150 + 200 = 800 ps
Spade: 80 + 80 + 220 + 120 + 180 = 680 ps
Omega: 90 + 90 + 230 + 130 + 190 = 730 ps
Infinity: 70 + 70 + 210 + 110 + 170 = 630 ps

1b. ET = CT * CPI * IC
CPI = 1 because it is a single cycle datapath
IC = 10 * 10^9

Yin/Yang: ET = 800 * 10^-12 * 10^10 = 800 * 10 ^-2 = 8 seconds
Spade: ET = 680 * 10^-12 * 10^10 = 6.8 seconds
Omega: ET = 7.3 seconds
Infinity: ET = 6.3 seconds

2. RegDst:

- lw: R[Imm[15…11]] =M[SE(IMM) + R[rs]}
- lw requires RegDst to be 0. It is supposed to save the value from memory into register Rt, but if RegDst is 1, it will store the value from memory into the register specified by the upper 5 bits of the immediate field, which is arbitrary.
- No other types will be affected. R-Types require RegDst to be 1 and sw and beq allow RegDst to be "Don't Care" because they specify RegWrite to be 0.

AluSrc:

- lw and sw require AluSrc to be 1, so they will work correctly
- R-Type: R[rd] = R[rs] OP SE(IMM)
- beq: if(R[rs] == SE(IMM)) then PC = PC + 4 + SE(IMM)
- If AluSrc is 1 then the second input into the ALU is the SE(IMM). This is used for lw/sw, but in the case of R-Type instructions, you will not be doing an operation on R[rs] and R[rt] as expected, you will do the operation on R[rs] and SE(IMM), where the IMM is actually the Rd, shamt, and funct.
- In a branch, you are supposed to subtract R[rs] and R[rt] in order to determine if they are equal. Instead, you will compare R[rs] and the SE(IMM) to determine whether to branch.

Branch:

- beq requires branch to be 1 so beq works correctly.
- lw/sw: if(R[rs] + SE(IMM) == 0) then PC = PC + 4 + SE(IMM)
- R-Type: if(R[rs] OP R[rt] == 0) then PC = PC + 4 + SE(IMM)
- lw and sw will do the correct memory/register operations, except if the output of the ALU happens to be 0, then the PC will branch to PC + 4 + SE(IMM), which should never happen in lw/sw.
- R-Type will also do the correct register write operation, except if the output of the ALU happens to be 0, then the PC will branch to PC + 4 + SE(IMM) where IMM is actually the rd, shamt, and funct.

AluOp1:

- ALUOp1 is the most significant bit in ALUOp. ALUOp1 is required to be 1 for R-Type instructions so R-Type instructions will work fine.
- lw: R[rt] = M[R[rs] OP SE(IMM)]
- sw: M[R[rs] OP SE(IMM)] = R[rt]
- beq: if(R[rs] OP R[rt] == 0) then PC = PC + 4 + SE(IMM)
- load and store will both operate on the right values, but the operation should be to add the base address with the offset. However, because the opcode is now 10, the operation will be defined by the "funct" field, which is actually the least significant 6 bits of the IMM, which is arbitrary.
- beq will also operate on the right values, but the operation should be subtract to compare the equality of R[rs] and R[rt]. Now the opcode is 11, which means that operation will be arbitrary or undefined.

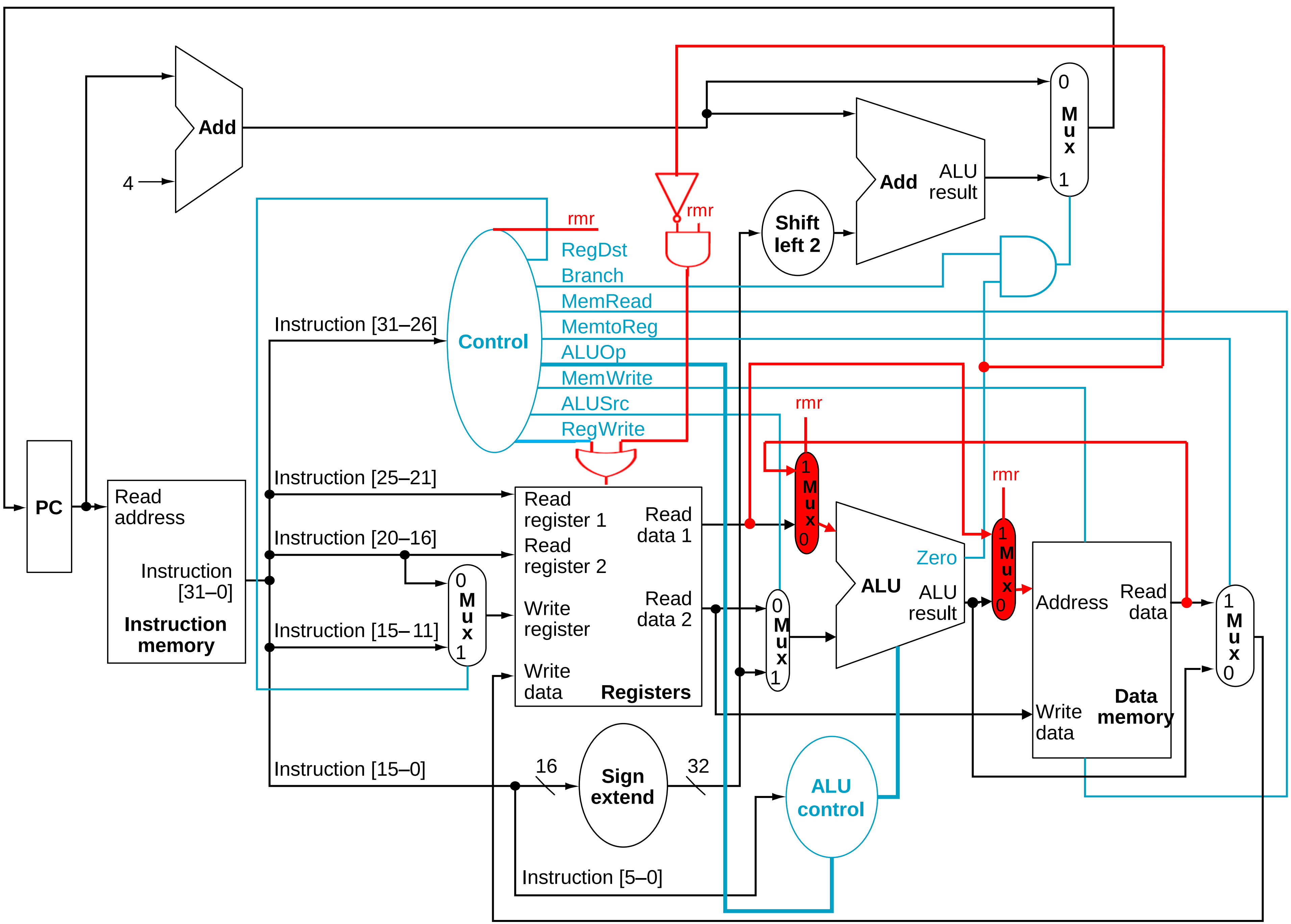
3.

Both branches are taken once, then not taken. We predict not taken, which means that the first time that each branch is reached, they will mispredict not taken. Because branch resolution occurs in EX stage, there is a 2 instruction/cycle penalty. With full forwarding the only dependency that must result in a single stall are load dependencies where a load is immediately followed by a dependent instruction. The actual set of instructions that will enter the pipeline are as follows. Load dependencies are highlighted. See attached for clock cycle chart.

lw $t0, 8($t1)
lw $t3, 0($t0)
add $t0, $t2, $t3
bne $t0, $s1, HERE
add // Mis-prediction
lw // Mis-prediction
add $t0, $s0, $t0
beq $t1, $s3, THERE
sw // Mis-prediction
next // Mis-prediction
lw $t0, 8($t1)
lw $t3, 0($t0)
add $t0, $t2, $t3
bne $t0, $s1, HERE
add $t0, $s0, $t0
lw $t0, 0($t0)
add $t1, $s1, $s2
add $t1, $t0, $t1
beq $t1, $s3, THERE
sw $t1, 0($t9)

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28
lw $t0, 8($t1)	IF	ID	EX	ME	WB																							
lw $t3, 0($t0)		IF	ID	{EX}	EX	ME	WB																					
add $t0, $t2, $t3			IF	{ID}	ID	{EX}	EX	ME	WB																			
bne $t0, $s1, HERE				{IF}	IF	{ID}	ID	EX	ME	WB																		
add //WRONG						{IF}	IF	ID	--																			
lw //WRONG								IF	--																			
add $t0, $s0, $t0									IF	ID	EX	ME	WB															
beq $t1, $s3, THERE										IF	ID	EX	ME	WB														
sw //WRONG											IF	ID	--															
next //WRONG												IF	--															
lw $t0, 8($t1)													IF	ID	EX	ME	WB											
lw $t3, 0($t0)														IF	ID	{EX}	EX	ME	WB									
add $t0, $t2, $t3															IF	{ID}	ID	{EX}	EX	ME	WB							
bne $t0, $s1, HERE																{IF}	IF	{ID}	ID	EX	ME	WB						
add $t0, $s0, $t0																		{IF}	IF	ID	EX	ME	WB					
lw $t0, 0($t0)																				IF	ID	EX	ME	WB				
add $t1, $s1, $s2																					IF	ID	EX	ME	WB			
add $t1, $t0, $t1																						IF	ID	EX	ME	WB		
beq $t1, $s3, THERE																							IF	ID	EX	ME	WB	
sw $t1, 0($t9)																								IF	ID	EX	ME	WB

4.



PC
Add
4
Read address
Instruction [31–0]
Instruction memory
Instruction [31–26]
Control
RegDst
Branch
MemRead
MemtoReg
ALUOp
MemWrite
ALUSrc
RegWrite
rmr
rmr
rmr
rmr
Instruction [25–21]
Instruction [20–16]
Instruction [15–11]
Instruction [15–0]
Instruction [5–0]
Read register 1
Read register 2
Write register
Write data
Read data 1
Read data 2
Registers
16
Sign extend
32
Shift left 2
Add
ALU result
ALU
Zero
ALU result
ALU control
Address
Read data
Write data
Data memory
0 M u x 1
1 M u x 0

Main Controller:

		R-format	lw	sw	beq	rmr
Opcode		000000	100011	101011	000100	New code
Outputs	RegDst	1	0	X	X	0
	ALUSrc	0	1	1	0	1
	MemtoReg	0	1	X	X	1
	RegWrite	1	1	0	0	0
	MemRead	0	1	0	0	1
	MemWrite	0	0	1	0	0
	Branch	0	0	0	1	0
	ALUOp1	1	0	0	0	1
	ALUOp2	0	0	0	1	1
	rmr	**0**	**0**	**0**	**0**	**1**

ALU Controller:

opcode	ALUOp	Operation	funct	ALU function	ALU control
lw	00	load word	XXXXXX	add	0010
sw	00	store word	XXXXXX	add	0010
beq	01	branch equal	XXXXXX	subtract	0110
R-type	10	add	100000	add	0010
		subtract	100010	subtract	0110
		AND	100100	AND	0000
		OR	100101	OR	0001
		set-on-less-than	101010	set-on-less-than	0111
rmr	**11**	**AND then Load**	XXXXXX	**AND**	**0000**