TASK 2: Implementing a datapath to execute Other Immediate (I-type) operations

Memory transactions and

Control flow instruction

Task 2: Reading from memory

31:26	25:21	20:16	15:0
op	rs	rt	address

	Instruction Fields				Expression
	6 bits	5 bits	5 bits	16 bits	
Inst.	opcode	rs	rt	offset	
lw	100011	used	used	offset value	Reg[rt] = DataMemory[Reg[rs]+offset]

- Register file has limited space to store values needed by a program. If we are operating on a larger array, then the data will initially be stored in the data memory. If the program needs to operate on a specific element stored in the memory, then we need a way to bring that data from the memory to the datapath. For this we will have "lw" (load word) instruction.
- Specification for this instruction is different from the R-type of operations presented earlier.
- lw: uses register "rs" contents as the base address. You can think of "rs" storing the pointer to the first element of the array we want to access in the data memory. If we want to index to the 4th element of the array then we simply need to use this constant value stored as "offset value" within the instruction as the offset amount relative to the address of the first element.
- Operation steps is as follows:
  - First read the contents of the "rs" register (Reg[rs])
  - Then add offset value stored in the least significant 16 bits of the instruction with Reg[rs]. (Reg[rs]+offset value)
  - Result of this addition is the actual address we want to access in the "DataMemory". Using this value now we can read from the memory DataMemory[Reg[rs]+offset value]
  - After reading from the memory we will write the output of the data memory to our destination register rt. Note that in R-type of instructions, destination register is always indicated by the "rd " field.
- You need to configure the datapath to manage all these steps in one clock cycle.
- Controller will determine that this is a "lw" instruction based on its unique opcode value of 35.

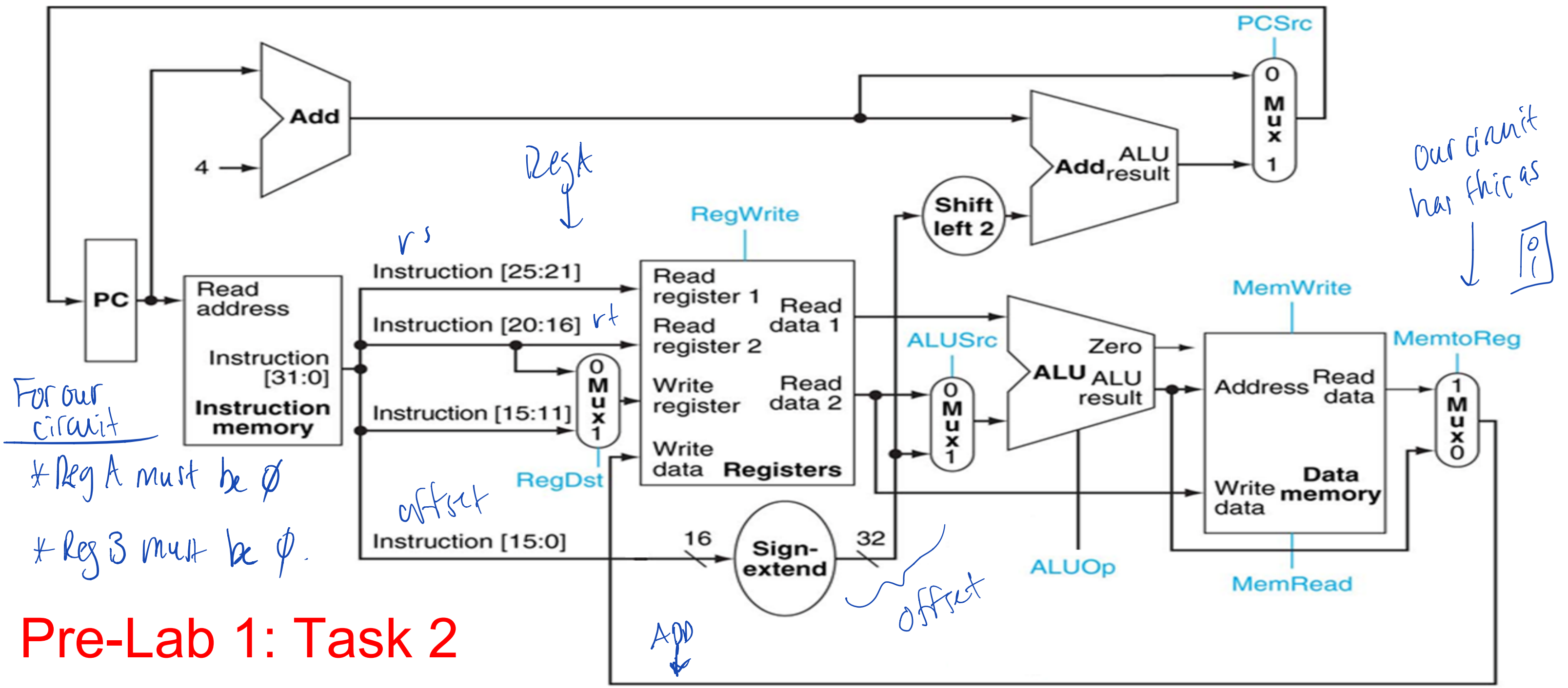
Task 2: Writing into memory

31:26	25:21	20:16	15:0
op	rs	rt	address

	Instruction Fields				Expression
	6 bits	5 bits	5 bits	16 bits	
Inst.	opcode	rs	rt	offset	
sw	101011	used	used	offset value	DataMemory[Reg[rs]+offset] = Reg[rt]

- Similarly, we need a way to store a value generated by the datapath back to the memory. For this we will have “sw” (store word) instruction. Specification for this instruction is similar to the “lw” instruction with a difference in the way we interpret it.
- sw: uses register “rs” contents as the base address and the “offset value” to calculate the address to write into in the Data Memory.
- Operation steps is as follows:
  - First read the contents of the “rs” register (Reg[rs])
  - Then add offset value stored in the least significant 16 bits of the instruction with Reg[rs]. (Reg[rs]+offset value)
  - Result of this addition is the actual address we want to write into in the “DataMemory”. Using this value now we will store the contents of the “rt” register into the memory as: DataMemory[Reg[rs]+offset value] = rt
  - Note that in lw we used “rt” as the destination address in the register file. For the sw, “rt” is a source register. We write its contents into the data memory.
- You need to configure the datapath to manage all these steps in one clock cycle.
- Controller will determine that this is a “sw” instruction based on its unique opcode value of 43.
- Now let’s identify the values of datapath control signals for executing lw and sw instructions. You will see the role of several muxes with this exercise. (next slide)
- Note: Some control signals may not be needed for these operations. In that case you must set those control values to “X” (don’t care) to receive full credit.

Pre-Lab 1: Task 2



For our circuit

* Reg A must be ∅

* Reg 3 must be ∅.

Type	RegDst	RegWrite	ALUSrc	ALUOp	MemRead	MemWrite	MemtoReg	PCSrc
lw	0	1	1	0000	1	0	1 *	X
sw	0	1	1	000	0	1	X	X

	Instruction Fields				Expression
	6 bits	5 bits	5 bits	16 bits	
Inst.	opcode	rs	rt	offset	
lw	100011	used	used	offset value	Reg[rt] = DataMemory[Reg[rs]+offset]
sw	101011	used	used	offset value	DataMemory[Reg[rs]+offset] = Reg[rt]

Task 2: Control Instruction (Part 1)

31:26	25:21	20:16	15:0
op	rs	rt	address

	Instruction Fields				Expression
	6 bits	5 bits	5 bits	16 bits	
Inst.	opcode	rs	rt	offset	
bne	000101	used	used	16 bits offset value	If (Reg[rs] != Reg[rt]) {PC = PC + 4 + offset } else {PC = PC + 4}

- Programs with control flow statements such as If-else, for-loop, and while-loop have conditions to be checked to decide which path to take during the execution. The “bne” (branch if not equal to) instruction is one of many that the datapath can execute to support these statements.
- This instruction uses both “rs” and “rt” fields as its source operand. After reading the contents of the source register, we use ALU to check the inequality condition.

  If the (Reg[rs] != Reg[rt]) condition is satisfied then we need to specify which part of the code in the instruction memory should be executed. This is handled by changing the value of the PC register.

- Now it is time to explain the role of the PC register. You can visualize PC as a register that holds a pointer to the Instruction Memory indicating which instruction will be executed next. Initially we set the PC value to 0 assuming that all our programs start with address 0. After compiling your C code, the binary instructions (each 32-bits) are stored in the Instruction Memory sequentially. With the clock (rising edge) the PC value is fed into the Instruction Memory triggering a read from the location pointed by the PC. As soon as the instruction is read from the instruction memory, the controller determines the nature of the operation and configures the datapath to execute that instruction. In the mean time PC value is increment by 4 (4 bytes, 32 bits) to point to the next instruction in line.
- Operation steps will be described in the next slide.

Task 2: Control Instruction (Part 2)

31:26	25:21	20:16	15:0
op	rs	rt	address

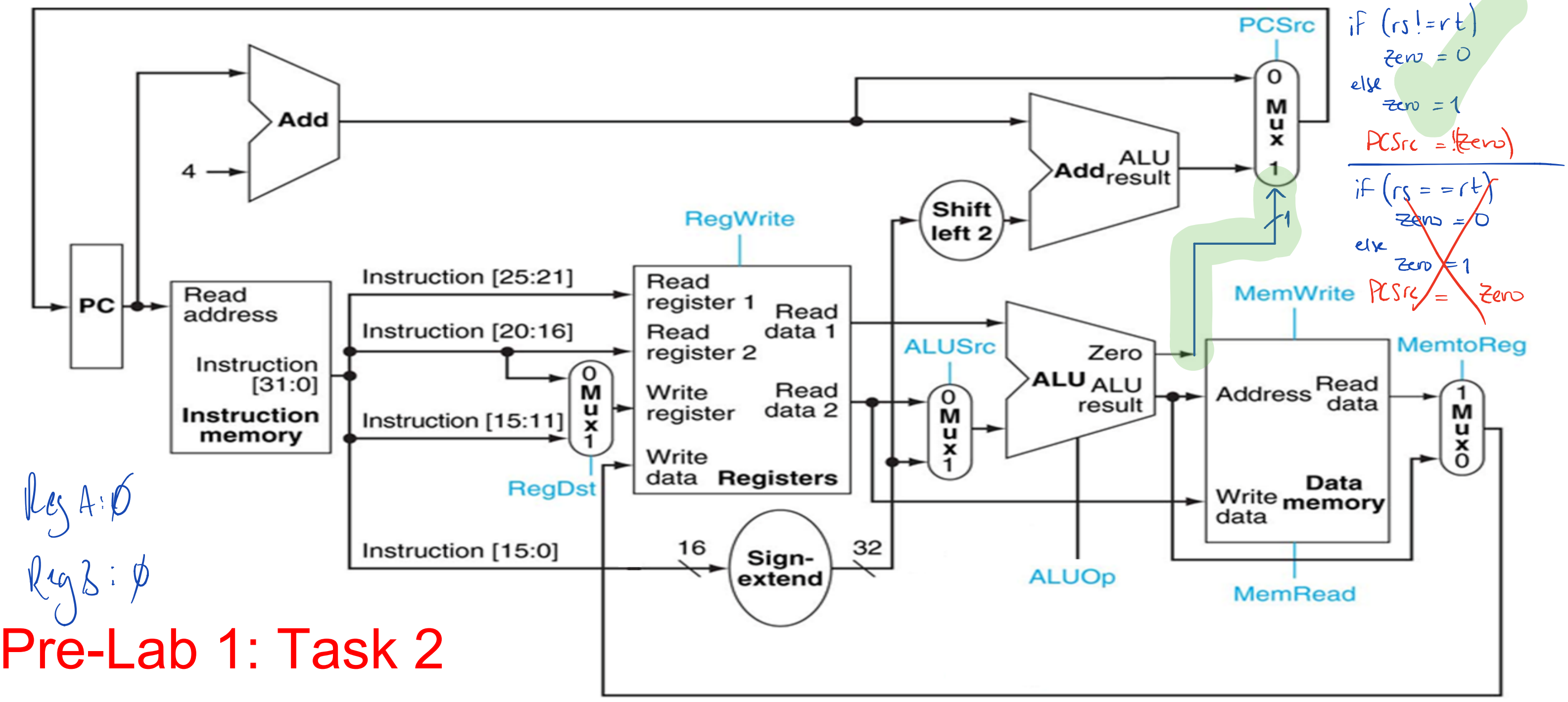
	Instruction Fields				Expression
	6 bits	5 bits	5 bits	16 bits	
Inst.	opcode	rs	rt	offset	
bne	000101	used	used	16 bits offset value	If (Reg[rs] != Reg[rt]) {PC = PC + 4 + offset*4 } else {PC = PC + 4}

- First read the contents of the "rs" and "rt" registers (Reg[rs], Reg[rt])
- Then feed these values to the ALU. Configure ALU to check for the "bne" condition)
- If the condition is met, PC should be updated with the new address to continue execution from. The top "Adder" component in the datapath handles the branch target address calculation by adding PC+4 with the 4*offset value (The offset amount is an integer value that needs to be sign extended and then multiplied by 4 (byte addressing) using the Shift Left 2 component).
- Note that PC+4 is already available to the datapath.

- Controller will determine that this is a "bne" instruction based on its unique opcode value of 5.
- Now let's identify the values of datapath control signals for executing the "bne" instruction. You will see the role of the mux controlled by the PCSrc signal with this exercise. (next slide)

- Note 1: You may need additional logic to control the PCSrc signal for bne instruction. If instruction is bne and the condition is met then PCSrc MUX should feed the address calculated by the adder component back in to the PC register, otherwise PCSrc MUX should feed the PC+4 back into the PC register.
- Note 2: Some control signals may not be needed for this operation. In that case you must set those control values to "X" (don't care) to receive full credit.

Pre-Lab 1: Task 2



if (rs != rt)
zero = 0
else
zero = 1
PCSrc = !(zero)

~~if (rs == rt)~~
~~zero = 0~~
~~else~~
~~zero = 1~~
~~PCSrc = zero~~

Type	RegDst	RegWrite	ALUSrc	ALUOp	MemRead	MemWrite	MemtoReg	PCSrc
bne	0	X	0	0111	X	X	X	!zero

	Instruction Fields				Expression
	6 bits	5 bits	5 bits	16 bits	
Inst.	opcode	rs	rt	offset	
bne	000101	used	used	16 bits offset value	If (Reg[rs] != Reg[rt]) {PC = PC + 4 + offset*4 } else {PC = PC + 4}

Task 2: Functional Verification

- Using the tables generated based on the pre-lab exercises (slides 4 and 7), revise your controller from Task-1 to incorporate the control for memory and branch type of operations.

- Implement the datapath to execute I-type of instructions
  - Verification will be based on post-routing simulation.
  - Instruction and data memory for testing I-type of instructions are given (read the next slide)

Task 2: Instruction and Data Memory

- The contents of “lw_sw_bne_inst_memory.txt” should be copied into your initialization block in the instruction memory.
- The contents of “lw_sw_bne_data_memory.txt” should be copied into your initialization block in the data memory.
  - You should initialize the data memory elements to 0. The file given here is just a sample.
- The “lw_sw_bne_type_with_answers.s” file shows the original source code with the expected results
  - Values in register 8 (RegFile[8]) will be monitored during your post-routing simulation. This corresponds to “t0” in the “lw_sw_bne_inst_memory.txt”
  - Values in second and third word (address 1 and 2) of the Data Memory will be monitored.

- After initializing the instruction memory and data memory, synthesize your design and run post-routing simulation.
- Bring register 8 (RegFile[8]) from your register file to your simulation waveform
- Bring 2nd and 3rd word of your Data Memory to your simulation waveform.
- Monitor the values

Expected Output and Grading Scheme (total of 150 points)

Cycle	Inst.	Value	Points
1	ori	Reg[9]=0	4
2	addi	Reg[18]=29	4
3	addi	Reg[19]=12	4
4	sw	DataMemory[1]=29	16
5	sw	DataMemory[2]=12	4
6	lw	Reg[8]=29	16
7	lw	Reg[8]=12	4
8	bne	Compares Reg[18] with Reg[8]. They are not equal so the condition is satisfied. Datpath should not execute the following two instructions	20
	addi	Reg[8]=0 (If value becomes 0 then bne is not working)	
	addi	Reg[8]=12 (If value becomes 12 then bne is not working)	
9	bne	Compares Reg[19] with Reg[8]. They are equal so the condition is not satisfied. Datapath should execute the following instruction	8
10	addi	Reg[8]=1 (means bne is working properly)	5
11	addi	Reg[8]=29	5
		Total (Behavioral simulation)	90
		Pre-lab	10

Note that in the table DataMemory[1] is in C syntax and refers to 2nd word of the memory. In hardware implementation memory is actually byte addressable. Each entry is 32-bits long (4 bytes). Therefore accessing the second element of the DataMemory is expressed as DataMemory[4] on the datapath. This indicates starting from the 4th byte in the data memory access bytes 4,5,6 and 7 to form a 32-bit word.

DataMemory[0] covers bytes 0,1,2, and 3. (first word in the memory)

DataMemmory[8] covers bytes 8,9,10,11 (3rd word in the memory)