CS232 Midterm Exam 2
April 9, 2001

Name: Mini-Me

- This exam has 8 pages, including this cover.
- There are four questions, each worth 25 points.
- You have 50 minutes. Budget your time!
- No written references or calculators are allowed.
- To make sure you receive full credit, write clearly and show your work.
- We will not answer questions regarding course material.

Question	Maximum	Your Score
1	25	25
2	25	25
3	25	25
4	25	25
Total	100	100

Question 1: Single-cycle datapath

Let's say we want to execute the following immediate addition instruction in the single-cycle datapath:

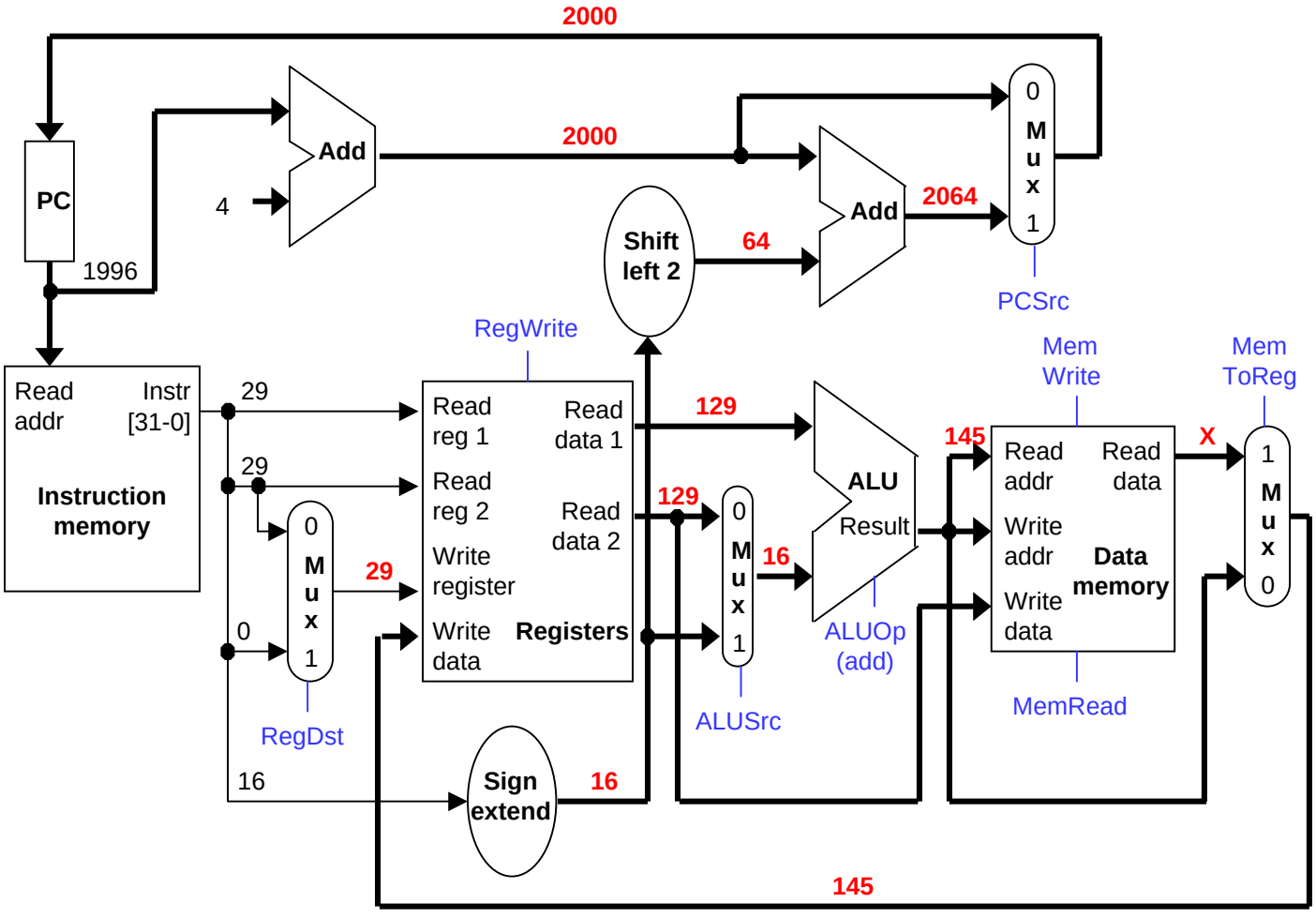
addi $29, $29, 16

The single-cycle datapath diagram below shows the execution of this instruction. Several of the datapath values are filled in already. You are to provide values for the twelve remaining signals in the diagram, which are marked with a **?** symbol. (2 points each)

You should:

- Write your answers directly on the diagram, but write clearly.
- Show decimal values.
- Assume register $29 initially contains the number 129.
- If a value cannot be determined, mark it as 'X.'

Notice that we didn't actually discuss immediate arithmetic instructions in lecture. However, it's pretty straightforward with this datapath. This problem is supposed to test your knowledge of the datapath, but it's possible to fill in values intuitively. For example, you know that the destination register is $29, so the "Write Register" input of the register file should clearly be 29. You are supposed to show as many values as possible, only filling 'X's for undeterminable values, but some people used 'X' as a don't care instead.



Question 2: Multicycle CPU implementation

We would like to add a "scaled" addressing mode to the MIPS multicycle architecture:

lws rd, rs, rt # rd = Mem[rt + (4 × rs)]

For example, if $a0 contains 1000 and $a1 contains 10, then "lws $t0, $a1, $a0" loads $t0 with the value at address 1040 (1000 + 4×10).

You need to show the correct control signals necessary to implement the lws instruction, by *either*:
- completing the finite state machine diagram on page 4, *or*
- filling in the microprogramming table on page 5.

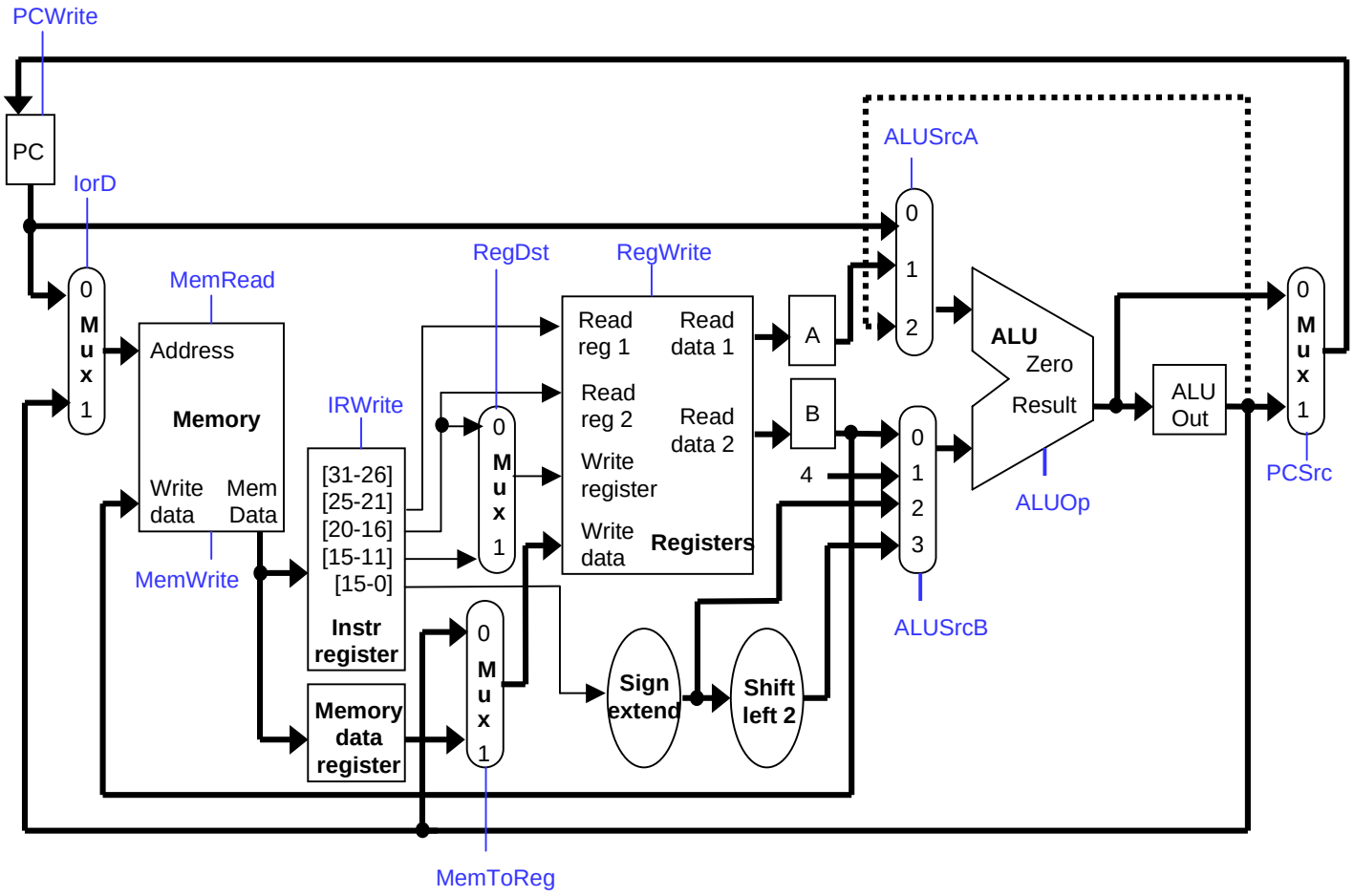
You only need to do one or the other, not both.

The multicycle datapath from lecture is shown below, with one important change: the ALUOut register is connected to the ALUSrcA mux (shown with a dotted line), which now has three inputs instead of two. No other changes to the datapath are needed.

You may assume that ALUOp = 100 performs an integer multiplication, and 010 performs an addition.

Finally, here is the instruction format, for your reference (shamt and func are not used):

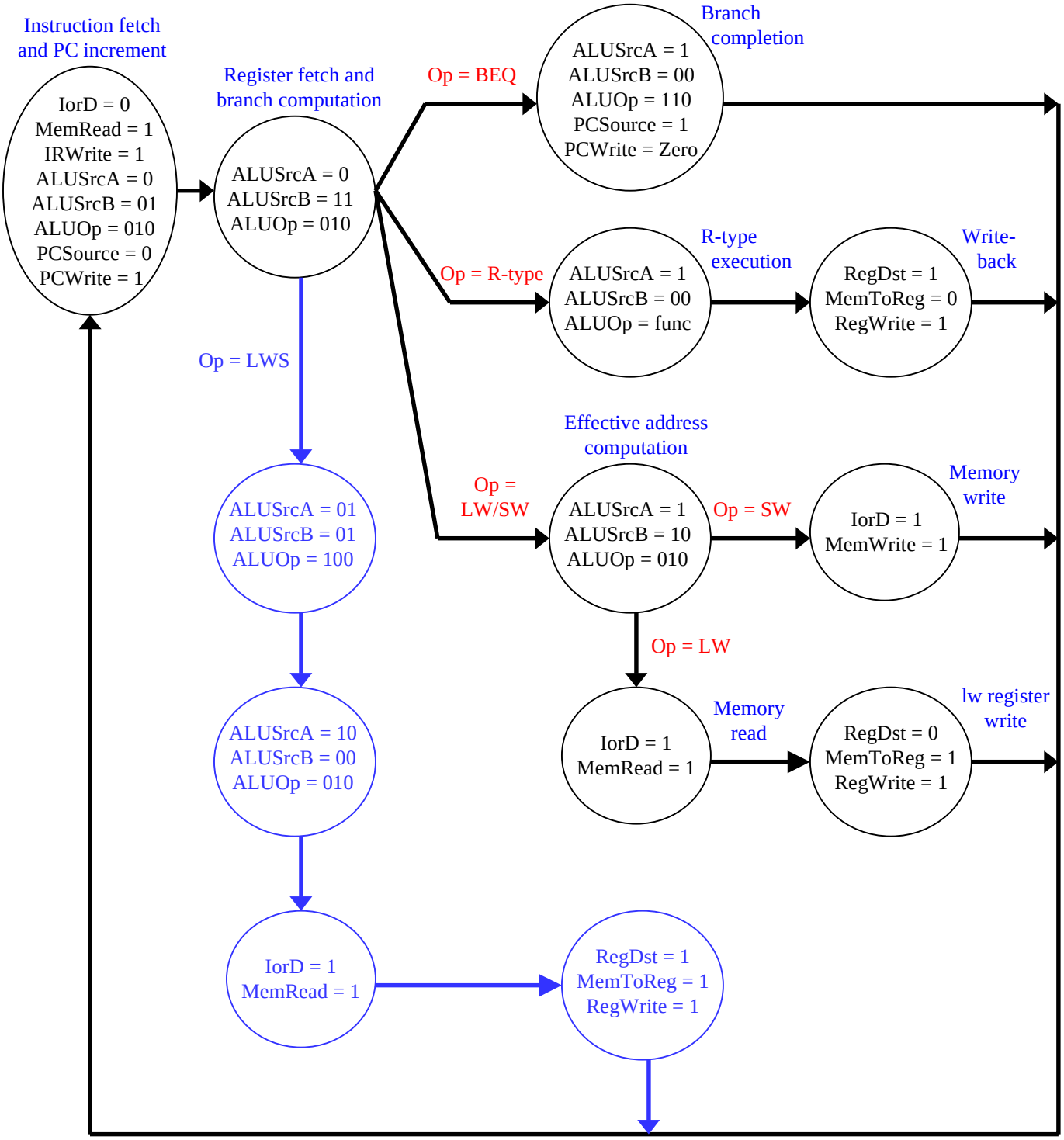
Field	op	rs	rt	rd	shamt	func
Bits	31-26	25-21	20-16	15-11	10-6	5-0



Question 2 continued

Complete this finite state machine diagram for the lws instruction, *or* fill in the microprogramming table on the next page, *but not both!*

You can show the control values in either binary or decimal, whichever is more convenient for you.



See the comments on the next page.

Question 2 continued

Fill in this microprogramming table for the lws instruction, *or* complete the finite state machine diagram on the previous page, *but not both!*

You may need to make up new values for some of these fields, but just make sure your intentions are clear.

Label	ALU Control	Src1	Src2	Register control	Memory	PCWrite control	Next
Fetch	Add	PC	4		Read PC	ALU	Seq
	Add	PC	Extshift	Read			Dispatch 1
BEQ1	Sub	A	B			ALU-Zero	Fetch
Rtype1	Func	A	B				Seq
				Write ALU			Fetch
Mem1	Add	A	Extend				Dispatch 2
SW2					Write ALU		Fetch
LW2					Read ALU		Seq
				Write MDR			Fetch
LWS1	Multiply	A	4				Seq
	Add	ALU Out	B				Seq
					Read ALU		Seq
				Write MDR to register rd			Fetch

You should only need to add a few symbols such as "Multiply" to get the ALU to multiply, and "ALUOut" to set the value of the expanded ALUSrcA mux.

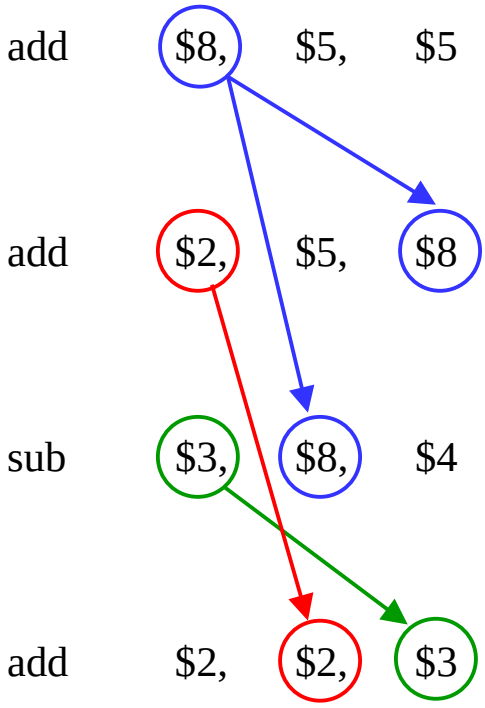
*Otherwise, the solution here is basically the same as the one on the previous page. The "lws" instruction has a more complicated addressing mode, so two cycles are required to compute the effective address. The first new cycle computes (4 * rs), and the second computes (4 * rs) + rt. The final two stages of the "lws" are almost the same as for the original "lw" instruction, but we need to store in register rd instead of rt.*

Since we've split this datapath into five distinct parts, you cannot combine any of the stages of the "lws" instruction together. For example, "Write MDR to register rd" in the microprogram requires an address to be supplied from the ALUOut register, but that must be produced in the previous clock cycle.

Finally, we said that the intermediate registers A, B, ALUOut and MDR are implicitly written to on every clock cycle. In this situation it's all right to use B (in "Add ALUOut B") two cycles after it's written (in the Register control "Read" stage)—within those two cycles, IR does not change, so none of the inputs to the register file change, so the register file outputs will not change either.

Question 3: Pipelining and forwarding

Part (a)
Show or list all of the dependencies in this program. For each dependency, indicate which instructions *and* register are involved. (5 points)



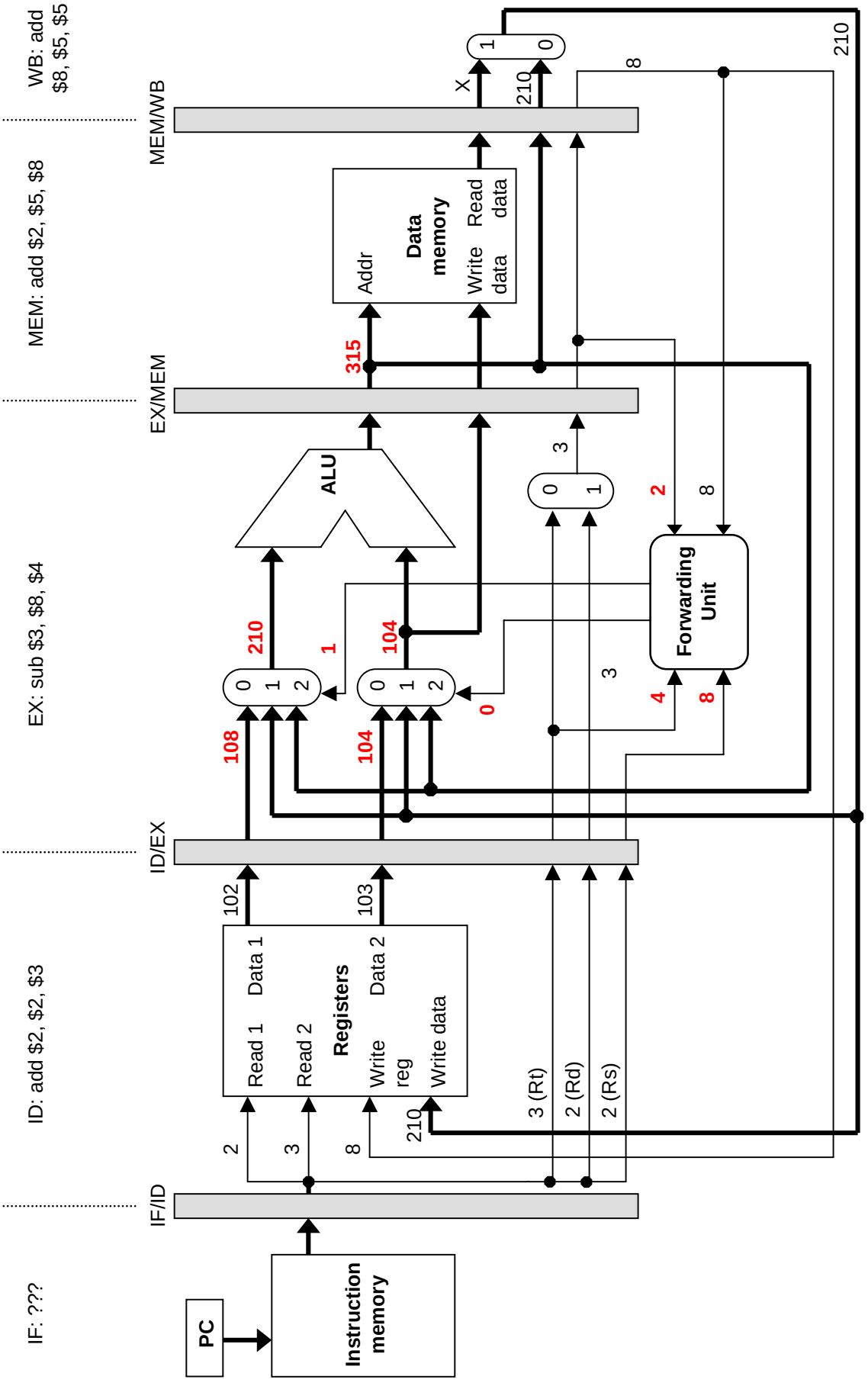
You can draw the dependencies like this, or describe them in words; something like "Source register $8 in the second add and the sub both depend upon $8 in the first instruction," and so forth.

Part (b)
The pipelined datapath on the next page shows the fifth cycle of executing this program, including values for several of the stages. Fill in the ten remaining values, marked with a **?** symbol, in the EX and MEM stages. (20 points; 2 points each)

Again, please:
- Write your answers directly on the diagram, but write clearly.
- Show decimal values.
- Assume that registers initially contain their number plus 100: $2 contains 102, $8 contains 108, etc.
- Write 'X' for any numbers that cannot be determined.

Your diagram should show the correct value of $8 (210) being forwarded to the second and third instructions; the EX/MEM pipeline should contain 315 (210 + 105), and the ALU's first input should be 210, not 108.

Question 3 continued



Question 4: Pipelining performance

IPG	FET	ROT	EXP	REN	WLD	REG	EXE	DET	WRB
1	2	3	4	5	6	7	8	9	10

One CPU manufacturer has proposed the 10-stage pipeline above for a 500MHz (2ns clock cycle) machine. Here are the correspondences between this and the MIPS pipeline:

- Instructions are fetched in the FET stage.
- Register reading is performed in the REG stage.
- ALU operations *and* memory accesses are both done in the EXE stage.
- Branches are resolved in the DET stage.
- WRB is the writeback stage.

Part (a)

How much time is required to execute one million instructions on this processor, assuming there are no dependencies or branches in the code? (5 points)

*It takes 9 cycles to fill the pipeline, and then 1,000,000 more cycles to complete the instructions, for a total of 1,000,009 * 2 = 2,000,018 ns.*

Part (b)

Without forwarding, how many stall cycles are needed for the following code fragment? (5 points)

```
lw      $t0, 0($a0)
add     $v1, $t0, $t0
```

The "lw" would not store its result into $t0 until the WRB stage, but that value must be read by the "add," in its REG stage. The diagram below shows that 2 stall cycles would be necessary. (If you didn't assume that the register file could be written and read on the same cycle, then a 3-cycle stall is needed.)

lw	*IPG*	*FET*	*ROT*	*EXP*	*REN*	*WLD*	*REG*	*EXE*	*DET*	*WRB*		
add		*IPG*	*FET*	*ROT*	*EXP*	*REN*	*WLD*			*REG*	*EXE*	*DET*	*WRB*

Part (c)

If a branch is mispredicted, how many instructions would have to be flushed from the pipeline? (5 points)

Branches aren't resolved until the 9th (DET) stage, at which point there could be eight subsequent instructions in the pipeline. They would all have to be flushed.

Part (d)

Assume that a program executes one million instructions. Of these, 15% are load instructions which stall, and 10% of the instructions are branches. The CPU predicts branches correctly 75% of the time. How much time will it take to execute this program? (10 points)

One million instructions would take 1,000,009 cycles, as from Part (a). But now, 25,000 (2.5%) of those instructions will be mispredicted branches, each of which would incur an 8-cycle penalty from flushing. Also, each of the 150,000 (15%) stalled loads results in 2 extra cycles. The total cycles is then 1,000,009 + (25,000 * 8) + (150,000 * 2) = 1,500,009, *for a run time of 3,000,018ns. The performance is about 1.5 times worse, which emphasizes the importance of accurate branch prediction and minimizing stalls.*