7 Tomasulo's Algorithm [60 points]

Consider an in-order fetch, out-of-order dispatch, and in-order retirement execution engine that employs Tomasulo's algorithm. This engine has the following characteristics:

- The engine has four main pipeline stages: Fetch (F), Decode (D), Execute (E), and Write-back (W).
- The engine can fetch one instruction per cycle, decode one instruction per cycle, and write back the result of one instruction per cycle.
- The engine has two execution units: 1) an adder to execute ADD instructions and 2) a multiplier to execute MUL instructions.
- The execution units are fully pipelined. The adder has two stages (E1-E2), and the multiplier has four stages (E1-E2-E3-E4). Execution of each stage takes one cycle.
- The adder has a two-entry reservation station, and the multiplier has a three-entry reservation station.
- An instruction always allocates the first available entry of the reservation station (in top-to-bottom order) of the corresponding execution unit.
- Full data forwarding is available, i.e., during the last cycle of the E stage, the tags and data are broadcast to the reservation station and the Register Alias Table (RAT). For example, an ADD instruction updates the reservation station entries of the dependent instructions in the E2 stage. So, the updated value can be read from the reservation station entry in the next cycle. Therefore, a dependent instruction can potentially begin its execution in the next cycle (after E2).
- The multiplier and adder have separate output data buses, which allow both the adder and the multiplier to update the reservation station and the RAT in the same cycle.
- An instruction continues to occupy a reservation station slot until it finishes the Write-back (W) stage. The reservation station entry is deallocated after the Write-back (W) stage.

7.1 Problem Definition

The processor is about to fetch and execute *five* instructions. Assume the *reservation stations (RS)* are all initially empty, and the initial state of the *register alias table (RAT)* is given below in Figure (a). Instructions are fetched, decoded, and executed as discussed in class. At some point during the execution of the five instructions, a snapshot of the state of the RS and the RAT is taken. Figures (b) and (c) show the state of the RS and the RAT at the snapshot time. A dash (–) indicates that a value has been cleared. A question mark (?) indicates that a value is unknown to you.

Reg	Valid	Tag	Value
R0	1	–	1900
R1	1	–	82
R2	1	–	1
R3	1	–	3
R4	1	–	10
R5	1	–	5
R6	1	–	23
R7	1	–	35
R8	1	–	61
R9	1	–	4

(a) Initial state of the RAT

Reg	Valid	Tag	Value
R0	1	?	1900
R1	1	?	82
R2	1	?	1
R3	1	?	45
R4	0	A	?
R5	0	F	?
R6	1	?	23
R7	1	?	35
R8	0	L	?
R9	0	B	?

(b) State of the RAT at the snapshot time

ID	V	Tag	Value	V	Tag	Value
-	-	-	-	-	-	-
L	1	?	82	1	?	1

+

ID	V	Tag	Value	V	Tag	Value
F	1	?	45	1	?	1
A	0	F	?	1	?	10
B	1	?	23	1	?	45

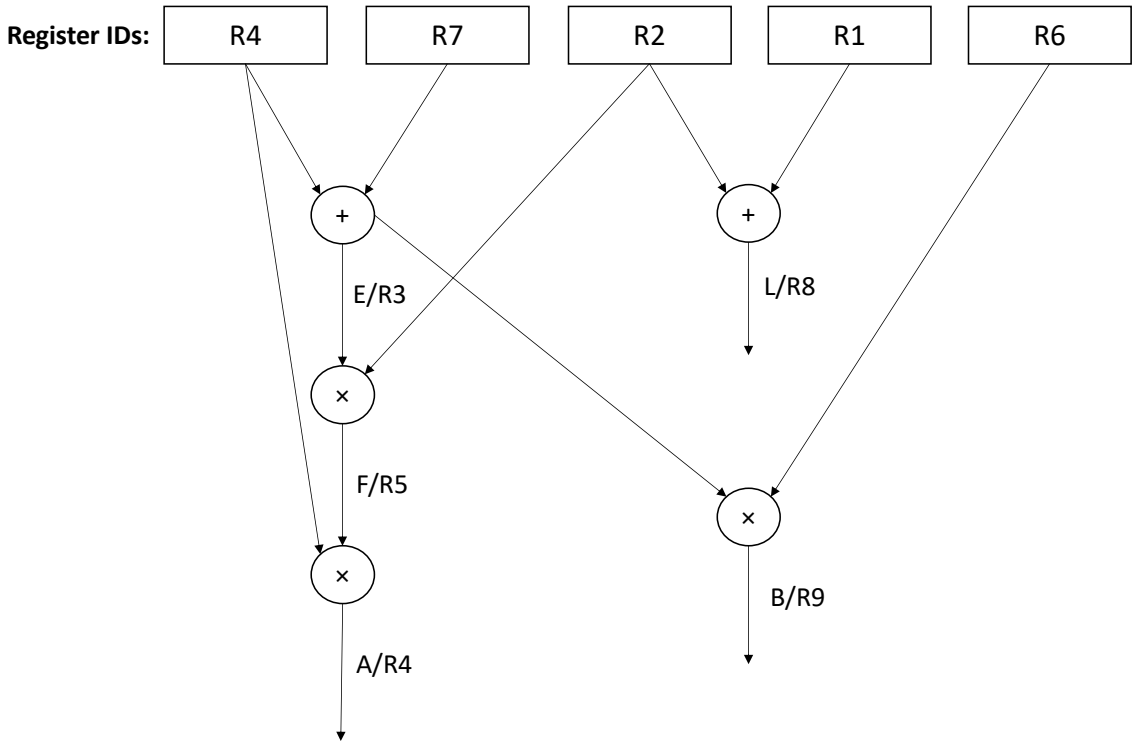
×

(c) State of the RS at the snapshot time

7.2 Questions

7.2.1 Dataflow Graph [40 points]

Based on the information provided above, identify the instructions and provide the dataflow graph below for the instructions that have been fetched. Please appropriately connect the nodes using edges and specify the direction of each edge. Label each edge with the destination architectural register and the corresponding Tag.



7.2.2 Program Instructions [20 points]

Fill in the blanks below with the five-instruction sequence in program order. There can be more than one correct ordering. Please provide *only one* correct ordering. When referring to registers, please use their architectural names (R0 through R9). Place the register with the smaller architectural name on the left source register box.
For example, `ADD R8 ⇐ R1, R5`.

ADD	R3	⇐	R4	,	R7
MUL	R5	⇐	R3	,	R2
MUL	R4	⇐	R5	,	R4
ADD	R8	⇐	R1	,	R2
MUL	R9	⇐	R6	,	R3