7 Tomasulo's Algorithm [50 points]

In this problem, we consider a scalar processor with in-order fetch, out-of-order dispatch, and in-order retirement execution engine that employs Tomasulo's algorithm. This processor behaves as follows:

- The processor has four main pipeline stages: Fetch (F), Decode (D), Execute (E), and Write-back (W).
- The processor implements a single-level data cache.
- The processor has the following *two types of execution units* but it is *unknown* how many of each type the processor has.
  - **Integer ALU:** Executes integer instructions (i.e., addition, multiplication, move, branch).
  - **Memory Unit:** Executes load/store instructions.
- The processor is connected to a main memory that has a fixed access latency.
- Load/store instructions spend cycles in the E stage exclusively for accessing the data cache or the main memory.
- There are two reservation stations, one for each execution unit type.

The reservation stations are all initially empty. The processor executes an arbitrary program. From the beginning of the program until the program execution finishes, *seven* dynamic instructions enter the processor pipeline. Table 3 shows the seven instructions and their execution diagram.

Instruction semantics:

- MV R0 ← #0x1000: moves the hexademical number 0x1000 to register $R0$.
- LD R1 ← [R0]: loads the value stored at memory address $R0$ to register $R1$.
- BL R1, #100, #LB1: a branch instruction that conditionally takes the path specified by label "#LB1" if the content of register $R1$ is smaller than integer value 100.
- MUL R1 ← R1, #5: multiplies $R1$ and 5 and writes the result to $R1$.
- ST [R0] ← R1: stores $R1$ to memory address specified by $R0$.
- ADD R1 ← R1, R0: adds $R1$ and $R0$ and writes the result to $R1$.

Instruction/Cycle:	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
1: MV R0 ← #0x1000	F	D	E1	E2	E3	E4	W																			
2: LD R1 ← [R0]		F	D	-	-	-	E1	E2	E3	E4	E5	E6	E7	E8	W											
3: BL R1 #100, #LB1						F	D	-	-	-	-	-	-	-	E1	E2	E3	E4	W							
4: MUL R1 ← R1, #5														F	D	E1	E2	E3	//squashed (i.e., killed)							
5: ST [R0] ← R1															F	D	-	-	//squashed (i.e., killed)							
6: ADD R1 ← R1, R0																			F	D	E1	E2	E3	E4	W	
7: ST [R0] ← R1																				F	D	-	-	-	E1	W

Table 3: Execution diagram of the seven instructions.

(a) [32 points] Using the information provided above, answer the following questions regarding the processor design. If a question has more than one correct answer or a correct answer cannot be determined using the information provided in the question, answer the question as specifically as possible. For example, use phrases such as "at least/at most" and try to narrow down the answer using the information that is provided in the question and can be inferred from Table 3. If nothing can be inferred, write "Unknown" as an answer. Explain your reasoning briefly.

What is the cache hit latency?

1 cycle. The last ST instruction that writes to the same memory location that is previously loaded by LD takes only 1 cycle in E stage.

What is the cache miss latency?

8 cycles. The first LD instruction spends 8 cycles in the E stage.

What is the cache line size?

Unknown. We cannot infer the cache line size from one LD and ST instructions and also we cannot determine the minimum cache line size since the register width is not given in the question.

What is the number of entries in each reservation station (R)?

ALU $\rightarrow$ at least 2, MU $\rightarrow$ unknown

How many ALUs does the processor have?

At least 2 ALUs if not pipelined, otherwise at least 1 ALU. This is because we see two arithmetic instructions simultaneously in E stage in the pipeline diagram.

Is the integer ALU pipelined?

Yes if the processor has 1 ALU, otherwise no. The explanation is similar to the above question.

Does the processor perform branch prediction?

Yes because there are squashed (as a result of branch misprediction) instructions in the pipeline.

At which pipeline stage is the correct outcome of a branch evaluated?

At the end of stage E4. This is because in the next cycle after the branch instruction completes E4 previously fetched instructions are killed and an instruction from the correct path is fetched.

(b) [18 points] What is the program (i.e., static instructions) that leads to the execution diagram shown in Table 3? Fill in the blanks below with the known instructions of the program and also (if applicable) show where and how many unknown instructions there are in the program.

Program:

MV R0 ← #0x1000
LD R1 ← [R0]
BL R1 #100, #LB1
MUL R1 ← R1, #5
ST [R0] ← R1
Any number of unknown instructions can be here
LB1: ADD R1 ← R1, R0
ST [R0] ← R1