6 Pipeline (Reverse Engineering) [40 points]

The following piece of code runs on a pipelined microprocessor as shown in the table (F: Fetch, D: Decode, E: Execute, M: Memory, W: Write back). Instructions are in the form "Instruction Destination, Source1, Source2." For example, "ADD A, B, C" means A ← B + C.

	Cycles	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
0	MUL R5, R6, R7	F	D	E1	E2	E3	E4	M	W										
1	ADD R4, R6, R7		F	D	E1	E2	E3	-	M	W									
2	ADD R5, R5, R6			F	D	-	-	E1	E2	E3	M	W							
3	MUL R4, R7, R7				F	-	-	D	E1	E2	E3	E4	M	W					
4	ADD R6, R7, R5							F	D	-	E1	E2	E3	M	W				
5	ADD R3, R0, R6								F	-	D	-	-	E1	E2	E3	M	W	
6	ADD R7, R1, R4										F	-	-	D	E1	E2	E3	M	W

Use this information to reverse engineer the architecture of this microprocessor to answer the following questions. Answer the questions as precise as possible with the provided information. If the provided information is not sufficient to answer a question, answer "Unknown" and explain your reasoning clearly.

(a) [5 points] How many cycles does it take for an adder and for a multiplier to calculate a result?

3 cycles for adder (E1, E2, E3) and 4 cycles for multiplier (E1, E2, E3, E4).

(b) [5 points] What is the minimum number of register file read/write ports that this architecture implements? Explain.

The register file has two read ports and one write port.

(c) [5 points] Can we reduce the execution time of this code by enabling more read/write ports in the register file? Explain.

It is not possible to reduce stall cycles of the given code by enabling more register file ports.

(d) [5 points] Does this architecture implement any data forwarding? If so, how is data forwarding done between pipeline stages? Explain.

There is data forwarding from the M stage to E1, as we observe that the instruction 2 starts using R5 at the clk cycle 7, which is one clk cycle after the instruction 0 finishes calculating its result in the execution unit.
Similarly, as another proof of this data forwarding, we observe that the instruction 4 starts using R5 at the clk cycle 10, which is one clk cycle after the instruction 2 finishes calculating its result in the execution unit.

Any other data forwarding is *unknown* with the given information.

(e) [5 points] Is it possible to run this code faster by adding more data forwarding paths? If it is, how? Explain.

Not possible.

All instructions that stall due to data dependency are already using the best possible data forwarding. There is no stall cycles that can be eliminated by enabling another form of data forwarding.

(f) [5 points] Is there internal forwarding in the register file? If there is not, how would the execution time of the same program change by enabling internal forwarding in the register file? Explain.

There already is internal forwarding in the register file, as instruction 6 can finish the decode stage by fetching the value of R4 from the register file in the same cycle that R4 is written (cycle 13).

(g) [10 points] Optimize the assembly code in order to reduce the number of stall cycles. You are allowed to *reorder*, *add*, or *remove* ADD and MUL instructions. You are expected to achieve the minimum possible execution time. Make sure that the register values that the optimized code generates at the end of its execution are identical to the register values that the original code generates at the end of its execution. Justify each individual change you make. Show the execution timeline of each instruction and what stage it is in the table below. (*Notice that the table below consists of two parts: the first ten cycles at the top, and the next ten cycles at the bottom.*)

- Instruction 1 is useless due to write-after-write, remove it.
- Instruction 3 stalls for decode logic, move it up.
- Instruction 6 does not have read-after-write dependency and can be executed before instr. 5. However, it cannot execute before instruction 4 as it would change the value of R7.

New total execution time is 17 cycles instead of 18.

Instr. No	Instructions	Cycles									
		1	2	3	4	5	6	7	8	9	10
0	MUL R5, R6, R7	F	D	E1	E2	E3	E4	M	W		
3	MUL R4, R7, R7		F	D	E1	E2	E3	E4	M	W	
2	ADD R5, R5, R6			F	D	-	-	E1	E2	E3	M
4	ADD R6, R7, R5				F	-	-	D	-	-	E1
6	ADD R7, R1, R4							F	-	-	D
5	ADD R3, R0, R6										F

		11	12	13	14	15	16	17	18	19	20
0	MUL R5, R6, R7										
3	MUL R4, R7, R7										
2	ADD R5, R5, R6	W									
4	ADD R6, R7, R5	E2	E3	M	W						
6	ADD R7, R1, R4	E1	E2	E3	M	W					
5	ADD R3, R0, R6	D	-	E1	E2	E3	M	W			