7. VLIW and Instruction Scheduling [60 points]

Explain the motivation for VLIW in one sentence.

> Enable multiple instruction issue with simple hardware. Independent instructions can be statically scheduled into a single VLIW instruction that can fed into multiple functional units concurrently.

You are the human compiler for a VLIW machine whose specifications are as follows:

- There are 3 **fully** pipelined functional units (ALU, MU and FPU).
- Integer Arithmetic Logic Unit (ALU) has a 1-cycle latency.
- Memory Unit (MU) has a 2-cycle latency.
- Floating Point Unit (FPU) has a 3-cycle latency, and can perform either FADD or FMUL (floating point add / floating point multiply) on **floating point registers**.
- This machine has **only** 4 integer registers (r1 .. r4) and 4 floating point registers (f1 .. f4)
- The machine does not implement hardware interlocking or data forwarding.

(a) For the given assembly code on the next page, fill **Table 1** (on the next page) with the appropriate VLIW instructions for only one iteration of the loop (The C code is also provided for your reference). Provide the VLIW instructions that lead to the **best** performance. Use the minimum number of VLIW instructions. Table 1 should **only** contain instructions provided in the assembly example. For all the instruction tables, show the NOP instructions you may need to insert. Note that `BNE` is executed in the **ALU**.

The base addresses for A, B, C are stored in r1, r2, r3 respectively. The address of the last element in the array C[N-1] is stored in r4, where N is an integer multiplier of 10! (read: 10 factorial).

Three extra tables are available for you to work with in the scratchpad, for the entirety of this question.

C code

```
float A[N];
float C[N];
int B[N];
... //code to initialize A and B
for (int i=0; i<N; i++)
        C[i] = A[i] * A[i] +B[i];
```

Assembly Code

```
loop:   LD      f1, 0 (r1)
        LD      f2, 0 (r2)
        FMUL    f1, f1, f1
        FADD    f1, f1, f2
        ADDI    r3, r3, 4
        ST      f1, -4 (r3)
        ADDI    r1, r1, 4
        ADDI    r2, r2, 4
        BNE     r3, r4, loop
```

VLIW Instruction	**ALU**	**MU**	**FPU**
1	ADDI r1, r1, 4	LD f1, 0(r1)	NOP
2	ADDI r2, r2, 4	LD f2, 0(r2)	NOP
3	NOP	NOP	FMUL f1, f1, f1
4	NOP	NOP	NOP
5	NOP	NOP	NOP
6	NOP	NOP	FADD f1, f1, f2
4	NOP	NOP	NOP
8	ADDI r3, r3, 4	NOP	NOP
9	BNE r3, r4, loop	ST f1, -4(r3)	NOP
10			
11			
12			
13			
14			
15			
16			
17			
18			
19			
20			
21			
22			
23			
24			
25			

Table 1

What is the performance in Ops/VLIW instruction (Operations/VLIW instruction) for this design? An operation here refers to an instruction (in the Assembly Code), excluding NOPs.

1

(b) Assume now we decide to unroll the loop once. Fill **Table 2** with the new VLIW instructions. You should optimize for latency first, then instruction count. **You can choose to use different offsets, immediates and registers, but you may not use any new instructions.**

VLIW Instruction	**ALU**	**MU**	**FPU**
1	NOP	LD f1, 0(r1)	NOP
2	ADDI r1, r1, 8	LD f3, 4(r1)	NOP
3	NOP	LD f2, 0(r2)	FMUL f1, f1, f1
4	ADDI r2, r2, 8	LD f4, 4(r2)	FMUL f3, f3, f3
5	NOP	NOP	NOP
6	NOP	NOP	FADD f1, f1, f2
7	NOP	NOP	FADD f3, f3, f4
8	NOP	NOP	NOP
9	ADDI r3, r3, 8	ST f1, 0(r3)	NOP
10	BNE r3, r4, loop	ST f3, -4(r3)	NOP
11			
12			
13			
14			
15			
16			
17			
18			
19			
20			
21			
22			
23			
24			
25			

Table 2

What is the performance in Ops/VLIW instruction for this design?

14/10

(c) Assume now we have **unlimited registers** and the loop is fully optimized (unrolled to the **best** performance possible). What is the performance in Ops/cycle for this design? Show your work and explain **clearly** how you arrived at your answer. You are not required to draw any tables, but you may choose to do so to aid your explanation. You will receive **zero** credit for a correct answer without any explanation. (Hint: trace the dependent instructions)

29/15. Notice that we can add 3 MU ops (2 LDs and 1 ST) and 2 FPU ops per unroll, while the ALU ops remain constant at 4. If you trace the table carefully, you will observe that the MU instruction stream will have 1 op/cycle by the time we unroll the loop five times. At this point, we have $4 + 15 + 10 = 29$ instructions over 15 cycles. Any further unrolling will result in a smaller ops/cycle since the MU instruction stream is already saturated.

What is the performance bottleneck for this code and why? Explain.

Memory Unit. We add 3 MU ops but only 2 FPU ops per unroll, eventually causing the MU instruction stream to be saturated before the FPU, despite the FPU having a longer compute latency. (We are more concerned with throughput than latency)