Problem 3

In this problem, you will port code to a simple VLIW machine, and modify it to improve performance. Details about the VLIW machine:

- Three fully pipelined functional units (Integer ALU, Memory, and Floating Point)
- Integer ALU has a 1 cycle latency
- Memory Unit has a 2 cycle latency
- FPU has a 3 cycle latency and can complete one add or one multiply (but not both) per clock cycle
- No interlocks

C Code:

```
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
            st   f1, 0(r3)
            addi r1, r1, 4
            addi r2, r2, 4
            addi r3, r3, 4
            bne  r3, r4, loop
```

Problem 3.A

Schedule operations into the VLIW instructions in the following table. Show only one iteration of the loop. Make the code efficient, but do not use software pipelining or loop unrolling. You do not need to write in NOPs (can leave blank).

ALU	Memory Unit	FPU
addi r1, r1, 4	ld f1, 0(r1)	
addi r2, r2, 4	ld f2, 0(r2)	
		fmul f1, f1, f1
		fadd f1, f1, f2
addi r3, r3, 4		
bne r3, r4, loop	st f1, -4(r3)	

What performance did you achieve? (in FLOPS per cycle): **2/9**

Problem 3.B

Unroll the loop by one iteration (so two iterations of the original loop are performed for every branch in the new assembly code). You only need to worry about the steady-state code in the core of the loop (no epilogue or prologue). Make the code efficient, but do not use software pipelining. You do not need to write in NOPs (can leave blank).

ALU	Memory Unit	FPU
	ld f1, 0(r1)	
addi r1, r1, 8	ld f3, 4(r1)	
	ld f2, 0(r2)	fmul f1, f1, f1
addi r2, r2, 8	ld f4, 4(r2)	fmul f3, f3, f3
		fadd f1, f1, f2
		fadd f3, f3, f4
addi r3, r3, 8	st f1, 0(r3)	
bne r3, r4, loop	st f3, -4(r3)	

What performance is achieved now? (in FLOPS per cycle): **4/10**_____

Problem 3.C

With unlimited registers, if the loop was fully optimized (loop unrolling and software pipelining), how many FLOPS per cycle could it achieve? What is the bottleneck? (Hint: You should not have to write out the assembly code.)

It could achieve 2/3 flops per cycle. It will be bottlenecked by memory accesses, since each iteration has 3 memory ops (2 loads and 1 store) and only 2 floating point ops, and there is only one functional unit for each.