Part A: VLIW

Question 1

I1→I2: 1 cycle
I3→I4: 1 cycle (overlaps with I2 → I3 dependency)
I4→I5: 2 cycles
Total: 4 cycles

Question 2

3 iterations, or 2 iterations (with code reordering).

Question 3

Inst.	ALU/Branch Unit	Memory Unit	Floating Point Unit
1 `loop:`	`addi x1, x1, 4`	`lw f0, 0(x1)`	
2	`addi x2, x2, 4`	`lw f3, 0(x2)`	
3			`fmul f2, f0, f1`
4			
5			
6			`fadd f4, f2, f3`
7			
8			
9	`bne x1, x3, loop`	`sw f4, -4(x2)`	
10			
11			
12			
13			
14			

Question 4

Inst.	ALU/Branch Unit	Memory Unit	Floating Point Unit
1		`lw f0, 0(x1)`	
2		`lw f5, 4(x1)`	
3		`lw f7, 8(x1)`	`fmul f2, f0, f1`
4	`addi x1, x1, 12`	`lw f3, 0(x2)`	`fmul f9, f5, f1`
5		`lw f6, 4(x2)`	`fmul f10, f7, f1`
6		`lw f8, 8(x2)`	`fadd f4, f2, f3`
7	`addi x2, x2, 12`		`fadd f11, f9, f6`
8			`fadd f12, f10, f8`

9		`sw f4, -12(x2)`	
10		`sw f11, -8(x2)`	
11	`bne r1, r3, loop`	`sw f12, -4(x2)`	
12			
13			
14			

Question 5

We need 3 VLIW instructions per iteration.
Each iteration of the loop has 3 memory operations, and we can issue 1 memory op per VLIW instruction. Hence we need at least 3 instructions per iteration.
Each iteration has 2 floating point operations (one mul and one add) per iteration. So the throughput is 2/3 floating point operations per cycle.

The software pipelined code of the in-order processor achieves zero stalls (Question 2). But this code still has 8 instructions per iteration. And one instruction is issued per cycle (assuming no stalls). So one iteration takes 8 cycles. So the VLIW processor is 8/3 = 3x faster. If you assume that you can also hide the cost of the 3 bookkeeping instructions (you probably can), then the speedup is 5/3 = 2x faster. Note that we only gave full points if you *clearly explained* these cycle numbers and how you derived them.

Part B: Transactional Memory and Reliability

Question 1

(a) Not serializable

(b) Solution:

	Conflict cycle	Aborted Transaction (X, Y, or Neither)
Eager & Pessimistic	**20**	**Y**
Lazy & Optimistic	**40**	**Y**

Question 2

	Cycle	0	5	10	15	20	25	30	35	40	45
Transaction X		Begin		Rd A		Wr A				End	
Transaction Y			Begin		Rd A						End
Txn X	Write bit					🟥	🟥	🟥	🟥	🟥	
	Read bit										
Txn Y	Write bit										
	Read bit				🟥	🟥	🟥	🟥	🟥	🟥	

Question 3

Yes, just abort transactions on all cores when a bitflip is detected. Then have the cores sync at a barrier, clear the bitflip error flag, and continue normal execution.

Part C: Security

Question 1

(a) Guess one int of the password at a time: when you get the right i-th int, the code will run slightly longer (one extra iteration of the loop).

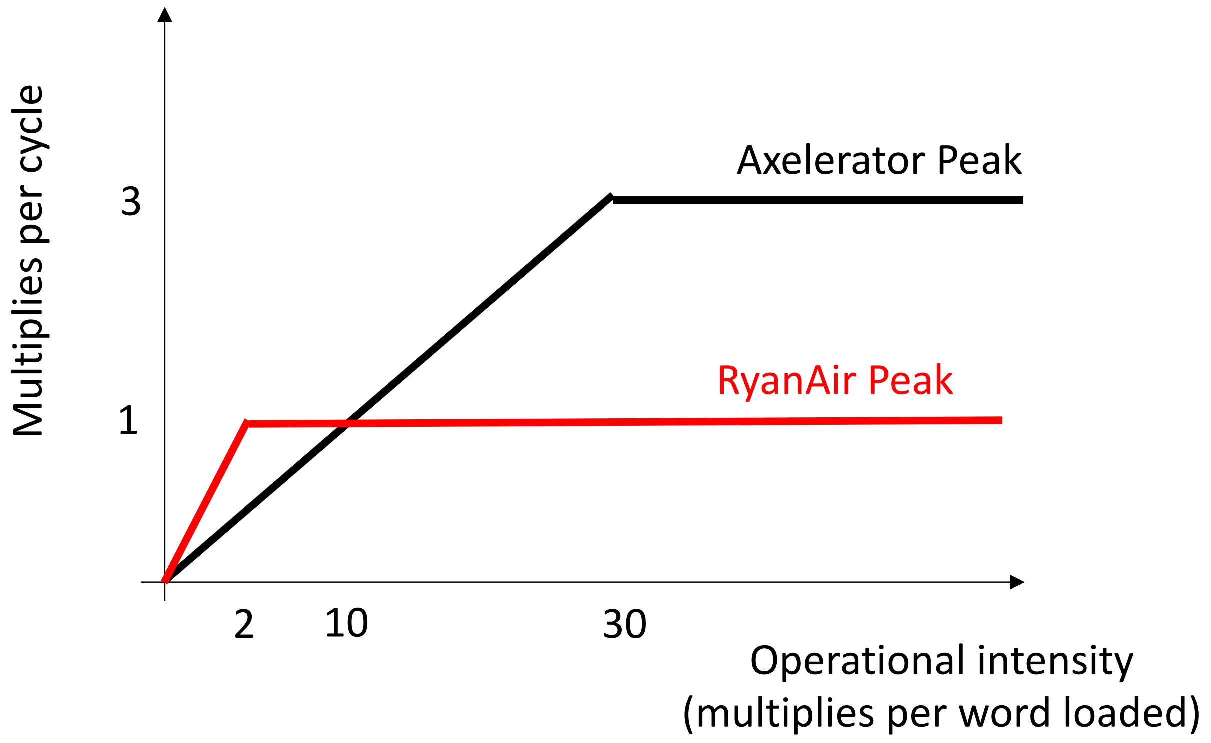
(b) L2 an L3

Question 2

```
        li      a0,1
loop:
        lw      t0,0(a0)
        lw      t1,0(a1)
        sub     t0,t0,t1
        seqz    t0,t0
        and     a0,a0,t0
        addi    t0,t0,4
        addi    t1,t1,4
        addi    a2,a2,-1
        bgt     a2,x0,loop
retTrue:
        ret
retFalse:
        ret
```

Part D: Accelerators

Question 1



The two plots intersect at x=10 (we also accept x=20).

Question 2

N^2 multiplies. N^2 + N loads. Operational intensity = N^2/(N^2+N) = N/(N+1) > 0.5 and < 1.

Question 3

RyanAir for all N (see roofline).

Question 4

2*N^2 loads. N^3 operations. Operational intensity = N^3/(2*N^2) = N/2.

Question 5

RyanAir better for N<20 (we also accept N<40). Axelerator better for N>20 (we also accept N>40). See roofline.

Question 6

0 loads. N operations. Operational intensity = infinity.

Question 7

Both systems perform equally well. The factorial function has a dependency between every iteration, so it is parallelism bound.

Question 8

Axel needs to increase memory bandwidth to at least match that of RyanAir (0.5 words/cycle).