Name ______________________________

Computer Architecture and Engineering
CS152 Quiz #4
April 9th, 2013

Professor Krste Asanović

Name: _____<ANSWER KEY>_____

This is a closed book, closed notes exam.

80 Minutes

15 pages

Notes:

- Not all questions are of equal difficulty, so look over the entire exam and budget your time carefully.
- Please carefully state any assumptions you make.
- Please write your name on every page in the quiz.
- You must not discuss a quiz's contents with other students who have not taken the quiz. If you have inadvertently been exposed to a quiz prior to taking it, you must tell the instructor or TA.
- You will get no credit for selecting multiple-choice answers without giving explanations if the instructions ask you to explain your choice.

Writing name on each sheet	________________	1 Point
Question 1	________________	22 Points
Question 2	________________	21 Points
Question 3	________________	12 Points
Question 4	________________	24 Points
TOTAL	________________	80 Points

Question 1: VLIW Machines [22 points]

In this question, we will consider the execution of the following code segment on a VLIW processor.

```
void kernel(int N, float A, float* X, float* Y, float* Z)
{
  for (int i=0; i<N; i++)
    Z[i] = Z[i] + A*X[i]*Y[i];
}
```

The code above translates to the following operation:

```
loop:
  flw f1, 0(x1) # load X[i]
  flw f2, 0(x2) # load Y[i]
  flw f3, 0(x3) # load Z[i]
  fmul f4, f0, f1 # A is in f0
  fmul f5, f4, f2
  fadd f6, f5, f3
  fsw f6, 0(x3)
  addi x1, x1, 4
  addi x2, x2, 4
  addi x3, x3, 4
  bne x3, x4, loop
```

This code will run on a VLIW machine that looks similar to the one presented in lecture and used in the problem set, shown here:

Int Op 1	Int Op 2	Mem Op 1	Mem Op 2	FP Add	FP Mul

Our machine has six execution units:

- **two** integer units, latency **one cycle** (i.e., dependent integer operations can be issued back-to-back), also used for branches.
- **two** memory units, latency **two cycles**, fully pipelined, each unit can perform both loads and stores.
- **two** FPU units, latency **four cycles**, fully pipelined, one unit can only perform a FP add operation, the other unit can only perform a FP mul operation.

This machine has no interlocks. All register values are read at the start of the instruction before any writes from the same instruction take effect (i.e., no WAR hazards between operations within a single VLIW instruction).

Please feel free to detach the Appendix page, which has the same code, for your convenience.

Name ______________________________

Q1.A Scheduling VLIW Code, Naïve scheduling [4 points]
Schedule operations into 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 don't need to write in NOPs.

Inst	Int Op 1	Int Op 2	Mem Op 1	Mem Op 2	FP Add	FP Mul
1	addi x1,x1,4	addi x2,x2,4	flw f1, 0(x1)	flw f2, 0(x2)		
2	addi x3,x3,4		flw f3,0(x3)			
3						fmul f4,f0,f1
4						
5						
6						
7						fmul f5,f4,f2
8						
9						
10						
11					fadd f6,f5,f3	
12						
13						
14						
15		bne x3,x4		fsw f6,-4(x3)		
16						
17						
18						
19						
20						
21						
22						
23						

What is the resulting throughput of the code in "floating-point operations per cycle"? *Don't count flw and fsw as floating-point operations.*

_______________3/15_______________

Name ______________________________

Q1.B Scheduling VLIW Code, Software pipelining [6 points]
Rewrite the assembly code to leverage software pipelining (do not use loop unrolling). Schedule VLIW instructions in the following table. You should show the loop prologue. Draw a box around the instructions that form the loop body.

Inst	Int Op 1	Int Op 2	Mem Op 1	Mem Op 2	FP Add	FP Mul
1	addi x1,x1,4	addi x2,x2,4	flw f1, 0(x1)			
2	addi x3,x3,4					
3	addi x1,x1,4	addi x2,x2,4	flw f1, 0(x1)			fmul f4,f0,f1
4	addi x3,x3,4					
5	addi x1,x1,4	addi x2,x2,4	flw f1, 0(x1)	flw f2,-8(x2)		fmul f4,f0,f1
6	addi x3,x3,4					
7	addi x1,x1,4	addi x2,x2,4	flw f1, 0(x1)	flw f2,-8(x2)		fmul f4,f0,f1
8	addi x3,x3,4					fmul f5,f4,f2
9	addi x1,x1,4	addi x2,x2,4	flw f1, 0(x1)	flw f2,-8(x2)		fmul f4,f0,f1
10	addi x3,x3,4		flw f3,-16(x3)			fmul f5,f4,f2
11	addi x1,x1,4	addi x2,x2,4	flw f1, 0(x1)	flw f2,-8(x2)		fmul f4,f0,f1
12	addi x3,x3,4		flw f3,-16(x3)		fadd f6,f5,f3	fmul f5,f4,f2
13	addi x1,x1,4	addi x2,x2,4	flw f1, 0(x1)	flw f2,-8(x2)		fmul f4,f0,f1
14	addi x3,x3,4		flw f3,-16(x3)		fadd f6,f5,f3	fmul f5,f4,f2
15	addi x1,x1,4	addi x2,x2,4	flw f1, 0(x1)	flw f2,-8(x2)		fmul f4,f0,f1
16	addi x3,x3,4	bne x3,x4	flw f3,-16(x3)	fsw f6,-28(x3)	fadd f6,f5,f3	fmul f5,f4,f2
17				flw f2,-8(x2)		fmul f4,f0,f1
18		bne x3,x4	flw f3,-16(x3)	fsw f6,-28(x3)	fadd f6,f5,f3	fmul f5,f4,f2
19				flw f2,-8(x2)		
20		bne x3,x4	flw f3,-16(x3)	fsw f6,-28(x3)	fadd f6,f5,f3	fmul f5,f4,f2
21						
22		bne x3,x4	flw f3,-16(x3)	fsw f6,-28(x3)	fadd f6,f5,f3	fmul f5,f4,f2
23						

What is the resulting throughput of the code in "floating-point operations per cycle"? Only consider the steady-state operation of the loop.

3/2

-2 if software pipeline loop is 3 cycles
-3 if software pipeline loop is 4 cycles
-4 if software pipeline loop is 5 cycles
-5 if software pipeline loop is 6 cycles
-6 if software pipeline loop is >= 7 cycles

-1 ~ -4 if work in prolog was not clear

Q1.C Impact on Performance
For the code given in Q1, how would the following changes impact "floating-point operations per cycle" in the steady-state operation of the loop? *Justify your answer*. You don't need to show the scheduled result as you did in Q1.A and Q1.B.

1) Loop unrolling [4 points]

The software pipelined loop is constrained by number of fmul instructions; loop unrolling wouldn't help.

New "floating-point operations per cycle" :

_______________3/2_______________

2) Additional FP Multiply unit [4 points]

The software pipelined loop is constrained by number of memory units.

New "floating-point operations per cycle" :

_______________3/2_______________

3) Memory Unit latency changes to 3 cycles [4 points]

Software pipelining will be able to hide memory unit latencies as long as it is constant.

New "floating-point operations per cycle" :

_______________3/2_______________

Question 2: Vector Machines [21 points]

In this question, we will consider the execution of the same code segment that was used in Q1 on a vector machine. Here the code is again, written in C:

```
void kernel(int N, float A, float* X, float* Y, float* Z)
{
  for (int i=0; i<N; i++)
    Z[i] = Z[i] + A*X[i]*Y[i];
}
```

The code above translates to the following traditional vector assembly code:

```
# x4 contains N at the beginning
vsetvl x5, x4
vflstw vf0, f0, x0 # f0 contains A
loop:
  vsetvl x5, x4
  vflw vf1, x1
  vflw vf2, x2
  vflw vf3, x3
  vfmul vf4, vf0, vf1
  vfmul vf5, vf4, vf2
  vadd vf6, vf5, vf3
  vfsw vf6, x3
  sub x4, x4, x5
  slli x5, x5, 2
  add x1, x1, x5
  add x2, x2, x5
  add x3, x3, x5
  bne x4, x0, loop
```

The baseline vector machine in this question has the following features:

- 64 elements per vector register
- 32 lanes
- one ALU per lane: 1 cycle latency
- one LD/ST unit per lane: 2 cycle latency
- one FP add unit per lane: 2 cycle latency
- one FP multiply unit per lane: 3 cycle latency
- all functional units have dedicated read/write ports into the vector register file
- no dead time
- no support for chaining
- scalar instructions execute separately on a control processor (5-stage, in-order)

Q2.A Scheduling Vector Code, No Chaining [4 points]

Complete the pipeline diagram of the baseline vector processor running the given code.

The following supplementary information explains the diagram:

- Scalar instructions execute in 5 cycles: fetch (**F**), decode (**D**), execute (**X**), memory (**M**), and writeback (**W**).
- A vector instruction is also fetched (**F**) and decoded (**D**). Then, it stalls (—) until its required vector functional unit is available.
- With no chaining, a dependent vector instruction stalls until the previous instruction finishes writing back all of its elements.
- A vector instruction is pipelined across all the lanes in parallel.
- For each element, the operands are read (**R**) from the vector register file, the operation executes on the load/store unit (**M**) or the ALU (**X**), and the result is written back (**W**) to the vector register file.
- A stalled vector instruction does not block a scalar instruction from executing.
- Assume that the vsetvl and vflstw instruction outside the loop has already been scheduled and executed.
- vflw1, vflw2, and vflw3 refer to the first, second, and third vflw instructions in the loop, etc.

Name ____________________

Inst	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	27	28	29	30	31	32	33	34	35	36	37	38	39	40
vsetvl	F	D	X	M	W																																			
vflw1		F	D	R	M1	M2	W																																	
					R	M1	M2	W																																
vflw2			F	D	-	R	M1	M2	W																															
							R	M1	M2	W																														
vflw3				F	D	-	-	R	M1	M2	W																													
									R	M1	M2	W																												
vfmul1					F	D	-	-	R	X1	X2	X3	W																											
										R	X1	X2	X3	W																										
vfmul2						F	D	-	-	-	-	-	-	-	R	X1	X2	X3	W																					
																R	X1	X2	X3	W																				
vfadd							F	D	-	-	-	-	-	-	-	-	-	-	-	-	R	X1	X2	W																
																						R	X1	X2	W															
vfsw								F	D	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	R	M	M	W											
																											R	M	M	W										
sub									F	D	X	M	W																											
slli										F	D	X	M	W																										
add											F	D	X	M	W																									
add												F	D	X	M	W																								
add													F	D	X	M	W																							
bne														F	D	X	M	W																						
vsetvl															F	D	X	M	W																					
vflw1																F	D	-	-	-	-	-	-	-	-	-	-	R	M	M	W									
																													R	M	M	W								
vflw2																	F	D	-	-	-	-	-	-	-	-	-	-	-	R	M	M	W							
																															R	M	M	W						

-1 per unique mistake.

Name ________________________________

Q2.B Scheduling Vector Code, with Chaining [5 points]
In this part, we analyze the performance benefits of chaining. Vector chaining is done through the register file and an element can be read (**R**) on the same cycle in which it is written back (**W**), or it can be read on any later cycle (the chaining is flexible).

To analyze performance, we calculate the total number of cycles per vector loop iteration by summing the number of cycles between the issuing of successive vector instructions. For example, in Question Q2.A, vflw1 begins execution in cycle 4 and vflw2 in cycle 6. Therefore, there are 2 cycles between vflw1 and vflw2.

Complete the following table. The first row corresponds to the baseline 32-lane vector processor with no chaining. The second row adds flexible chaining to the baseline processor.

Hint: You should consider each pair of vector instructions independently, and you can ignore the scalar instructions.

Vector Processor Config	Number of cycles between successive vector instructions							Total cycles per vector loop iteartion
	vflw1, vflw2	vflw2, vflw3	vflw3, vfmul1	vfmul1, vfmul2	vfmul2, vfadd	vfadd, vfsw	vfsw, vflw1	
32 lanes, no-chaining	2	2	1	6	6	5	2	24
32 lanes, chaining	2	2	1	4	4	3	2	18, see (b)

Q2.C Impact on Performance [12 points]
How would the following changes impact performance? Complete the following table.

Vector Processor Config	Number of cycles between successive vector instructions							Total cycles per vector loop iteartion
	vflw1, vflw2	vflw2, vflw3	vflw3, vfmul1	vfmul1, vfmul2	vfmul2, vfadd	vfadd, vfsw	vfsw, vflw1	
8 lanes, chaining	8	8	1	4	4	3	8	36, see (c)
32 lanes, chaining, additional LD/ST unit per lane	1	1	1	4	4	3	1	15, see (d)
32 lanes, chaining, additional FP mul unit per lane	2	2	1	4	4	3	2	18, see (e)

Name ____________

Inst	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	27	28	29	30	31	32	33	34	35	36	37	38	39	40
vsetvl	F	D	X	M	W																																			
vflw1		F	D	R	M1	M2	W																																	
					R	M1	M2	W																																
vflw2			F	D	-	R	M1	M2	W																															
							R	M1	M2	W																														
vflw3				F	D	-	-	R	M1	M2	W																													
									R	M1	M2	W																												
vfmul1					F	D	-	-	R	X1	X2	X3	W																											
										R	X1	X2	X3	W																										
vfmul2						F	D	-	-	-	-	-	R	X1	X2	X3	W																							
														R	X1	X2	X3	W																						
vfadd							F	D	-	-	-	-	-	-	-	-	R	X1	X2	W																				
																		R	X1	X2	W																			
vfsw								F	D	-	-	-	-	-	-	-	-	-	-	R	M	M	W																	
																					R	M	M	W																
sub									F	D	X	M	W																											
slli										F	D	X	M	W																										
add											F	D	X	M	W																									
add												F	D	X	M	W																								
add													F	D	X	M	W																							
bne														F	D	X	M	W																						
vsetvl															F	D	X	M	W																					
vflw1																F	D	-	-	-	-	R	M	M	W															
																							R	M	M	W														
vflw2																	F	D	-	-	-	-	-	R	M	M	W													
																									R	M	M	W												

(b) 32 lanes, chaining

Name ____________________________________

Inst	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	27	28	29	30	31	32	33	34	35	36	37	38	39	40
vflw3	F	D	R	M	M	W																																		
				R	M	M	W																																	
					R	M	M	W																														
						R	M	M	W																													
							R	M	M	W																												
								R	M	M	W																											
									R	M	M	W																										
										R	M	M	W																									
vfmul1		F	D	R	X	X	X	W																																
					R	X	X	X	W																													
						R	X	X	X	W																												
							R	X	X	X	W																											
								R	X	X	X	W																										
									R	X	X	X	W																									
										R	X	X	X	W																								
											R	X	X	X	W																							
vfmul2			F	D	-	-	-	R	X	X	X	W																												
									R	X	X	X	W																										
										R	X	X	X	W																									
											R	X	X	X	W																								
												R	X	X	X	W																							
													R	X	X	X	W																						
														R	X	X	X	W																					
															R	X	X	X	W																				
vadd				F	D	-	-	-	-	-	-	R	X	X	W																									
													R	X	X	W																								

(c) 8 lanes, chaining

Name ____________

Inst	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	27	28	29	30	31	32	33	34	35	36	37	38	39	40
vsetvl	F	D	X	M	W																																			
vflw1		F	D	R	M1	M2	W																																	
					R	M1	M2	W																																
vflw2			F	D	R	M1	M2	W																																
						R	M1	M2	W																															
vflw3				F	D	R	M1	M2	W																															
							R	M1	M2	W																														
vfmul1					F	D	R	X1	X2	X3	W																													
								R	X1	X2	X3	W																												
vfmul2						F	D	-	-	-	R	X1	X2	X3	W																									
												R	X1	X2	X3	W																								
vfadd							F	D	-	-	-	-	-	-	R	X1	X2	W																						
																R	X1	X2	W																					
vfsw								F	D	-	-	-	-	-	-	-	-	R	M	M	W																			
																			R	M	M	W																		
sub									F	D	X	M	W																											
slli										F	D	X	M	W																										
add											F	D	X	M	W																									
add												F	D	X	M	W																								
add													F	D	X	M	W																							
bne														F	D	X	M	W																						
vsetvl															F	D	X	M	W																					
vflw1																F	D	-	R	M	M	W																		
																				R	M	M	W																	
vflw2																	F	D	-	R	M	M	W																	
																					R	M	M	W																

(d) 32 lanes, chaining, additional LD/ST unit per lane

Name ____

Inst	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	27	28	29	30	31	32	33	34	35	36	37	38	39	40
vsetvl	F	D	X	M	W																																			
vflw1		F	D	R	M1	M2	W																																	
					R	M1	M2	W																																
vflw2			F	D	-	R	M1	M2	W																															
							R	M1	M2	W																														
vflw3				F	D	-	-	R	M1	M2	W																													
									R	M1	M2	W																												
vfmul1					F	D	-	-	R	X1	X2	X3	W																											
										R	X1	X2	X3	W																										
vfmul2						F	D	-	-	-	-	-	R	X1	X2	X3	W																							
														R	X1	X2	X3	W																						
vfadd							F	D	-	-	-	-	-	-	-	-	R	X1	X2	W																				
																		R	X1	X2	W																			
vfsw								F	D	-	-	-	-	-	-	-	-	-	-	R	M	M	W																	
																					R	M	M	W																
sub									F	D	X	M	W																											
slli										F	D	X	M	W																										
add											F	D	X	M	W																									
add												F	D	X	M	W																								
add													F	D	X	M	W																							
bne														F	D	X	M	W																						
vsetvl															F	D	X	M	W																					
vflw1																F	D	-	-	-	-	R	M	M	W															
																							R	M	M	W														
vflw2																	F	D	-	-	-	-	-	R	M	M	W													
																									R	M	M	W												

(3) x32 lanes, chaining, additional FP mul unit per lane (same as (b) since there's a RAW hazard between two vfmul)

Name ______________________________

Question 3: Multithreaded Machines [12 points]

In this question, we will consider the execution of the same code segment that was used in Q1 on a single-issue in-order multithreaded machine. Here the code is again, written in C:

```
void kernel(int N, float A, float* X, float* Y, float* Z)
{
  for (int i=0; i<N; i++)
    Z[i] = Z[i] + A*X[i]*Y[i];
}
```

The code above translates to the following assembly code:

```
loop:
  flw f1, 0(x1) # load X[i]
  flw f2, 0(x2) # load Y[i]
  flw f3, 0(x3) # load Z[i]
  fmul f4, f0, f1 # A is in f0
  fmul f5, f4, f2
  fadd f6, f5, f3
  fsw f6, 0(x3)
  addi x1, x1, 4
  addi x2, x2, 4
  addi x3, x3, 4
  bne x3, x4, loop
```

Each cycle, the processor can fetch and issue one instruction that performs any of the following operations:

- load/store, 25 cycle latency (fully pipelined)
- integer add, 1 cycle latency
- floating-point add, floating-point multiply, 4 cycle latency (fully pipelined)
- branch, no delay slots, 1 cycle latency

The processor does not have a cache. Each memory operation directly accesses main memory. If an instruction cannot be issued due to a data dependency, the processor stalls. We also assume that the processor has a perfect branch predictor with no penalty for both taken and not-taken branches.

Name ___________________________________

Q3.A Scheduling Multithreaded Code, Round-Robin Scheduling [6 points]

Consider a single-issue in-order, multithreaded pipeline, where threads are **switched every cycle** using a fixed round-robin schedule. If the thread is not ready to run on its turn, a bubble is inserted into the pipeline.

Each thread executes the above algorithm, and is calculating its own independent piece of the **Z** array (i.e., there is no communication required between threads). Assuming an infinite number of registers, what is the **minimum** number of threads we need to fully utilize the processor? You are free to re-schedule the assembly as necessary to minimize the number of threads required. Show your work.

```
loop:                                  start cycle
  flw f1, 0(x1) # load X[i]                1
  flw f2, 0(x2) # load Y[i]               N+1
  flw f3, 0(x3) # load Z[i]              2N+1
  addi x1, x1, 4                         3N+1
  addi x2, x2, 4                         4N+1
  addi x3, x3, 4                         5N+1
  fmul f4, f0, f1 # A is in f0           6N+1
  fmul f5, f4, f2                        7N+1
  fadd f6, f5, f3                        8N+1
  fsw f6, 0(x3)                          9N+1
  bne x3, x4, loop                      10N+1
```

Since the first flw is pushed to the memory system on cycle 1, we need the result written back to register file before (at cycle 6N) the fmul instruction issues.

6N >= 25

We need 5 threads to hide the memory latency. 5 threads is enough to cover the floating-point add and multiply functional latency.

-1 if said 4 threads, due to trivial cycle counting error
-3 if didn't reorder addi instructions (3N >= 25, need 9 threads)
-4 if didn't reorder addi instructions, and made a trivial cycle counting error (8 threads)

Name ___________________________________

Q3.B Scheduling Multithreaded Code, Dynamic Scheduling [6 points]

Now consider a single-issue in-order, multithreaded pipeline in which threads are **dynamically scheduled as needed** (i.e., the pipeline can pick the next instruction from any thread that is ready to execute). If the thread is not ready to run on its turn, a new thread is switched in.

What is the **minimum** number of threads we need to achieve peak performance? Again, you are free to re-schedule the assembly as necessary to minimize the number of threads required. Show your work.

In steady state:

```
  fadd f6, f5, f3
  flw f1, 0(x1) # load X[i]
  flw f2, 0(x2) # load Y[i]
  flw f3, 0(x3) # load Z[i]
  fsw f6, 0(x3)
  bne x3, x4, loop
  addi x1, x1, 4
  addi x2, x2, 4
  addi x3, x3, 4

...25-8 = 17 cycles to hide...

  fmul f4, f0, f1 # A is in f0

...3 cycles to hide...

  fmul f5, f4, f2

...3 cycles to hide...
```

You could hide 17 cycles with 2 threads executing the first 9 instructions. However, you cannot hide 3 cycles between the fmul instructions with 2 threads executing the fmul instruction. You need 3 threads to achieve peak performance.

I was surprised nobody software pipelined the loop. You could probably achieve peak performance with 2 threads. However, since nobody came up with this, I didn't take off any points.

-3 if said 2,4 threads

-1 ~ -5 if didn't justify your answer

Name ________________________________

Question 4: Iron Law of Processor Performance [24 points]

Mark whether the following modifications will cause each of the three categories to **increase**, **decrease**, or whether the modification will have **no effect**. ***Explain your reasoning*** to receive credit.

Assume the initial machine is a 5-stage in-order RISC pipeline. Also assume that any modification is done in a way that preserves correctness and maintains efficiency, but that the rest of the machine remains unchanged. Assume that applications can benefit from the proposed change.

		Instructions/Program	Cycles/Instruction	Seconds/Cycle
A	Add a single-lane vector unit.	Decrease – one vector instruction encodes more work.	Increase – vector instructions take more time to execute.	Unchanged, Might increase slightly – Vector units are highly
B	Move to a classic VLIW design with four operation slots per instruction.	Reduce – if enough parallelism to pack slots. Could increase if have to use many NOPs to avoid hazards.	Classic VLIW CPI = 1 -1 for increase due to hazards.	Should reduce as hardware is simpler – but not much change for single-issue. Might increase due to bypass complexities of quad-issue VLIW. -1 for increase but bad explanation

		Instructions/Program	Cycles/Instruction (Aggregate across threads)	Seconds/Cycle
C	Add fine-grain vertical multithreading and run a workload consisting of multiple separate single-threaded programs (multiprogramming)	Unchanged.	Should decrease as hazards are overlapped.	Similar – possibly slight increase.
D	Add fine-grain vertical multithreading and run a workload consisting of single programs that have been parallelized (multithreading)	Slight increase due to work distribution & synchronization between threads -1 if said same	Should decrease as hazards are overlapped with other thread's execution	Similar – possibly slight increase.

END OF QUIZ

Appendix. Code for your convenience

C Code:

```
void kernel(int N, float A, float* X, float* Y, float* Z)
{
  for (int i=0; i<N; i++)
    Z[i] = Z[i] + A*X[i]*Y[i];
}
```

Assembly Code:

```
loop:
  flw f1, 0(x1) # load X[i]
  flw f2, 0(x2) # load Y[i]
  flw f3, 0(x3) # load Z[i]
  fmul f4, f0, f1 # A is in f0
  fmul f5, f4, f2
  fadd f6, f5, f3
  fsw f6, 0(x3)
  addi x1, x1, 4
  addi x2, x2, 4
  addi x3, x3, 4
  bne x3, x4, loop
```

Vector Assembly Code:

```
# x4 contains N at the beginning
vsetvl x5, x4
vflstw vf0, f0, x0 # f0 contains A
loop:
  vsetvl x5, x4
  vflw vf1, x1
  vflw vf2, x2
  vflw vf3, x3
  vfmul vf4, vf0, vf1
  vfmul vf5, vf4, vf2
  vadd vf6, vf5, vf3
  vfsw vf6, x3
  sub x4, x4, x5
  slli x5, x5, 2
  add x1, x1, x5
  add x2, x2, x5
  add x3, x3, x5
  bne x4, x0, loop
```