CS152 Computer Architecture and Engineering

VLIW, Vector, and Multithreaded Machines

Assigned *03/19/2026*	Problem Set #4, Version (1.0)	*Due April 3 @ 11:59:59PT*

https://cs152-teach.github.io/sp26-web/

The problem sets are intended to help you learn the material, and we encourage you to collaborate with other students and to ask questions in discussion sections and office hours to understand the problems. However, each student must turn in their own solution to the problems.

We will distribute solutions to the problem set after the deadline to give you feedback.

Assignments must be submitted through Gradescope by **11:59:59pm PT** on the specified due date. *Box/clearly mark all solutions that don't involve filling in a figure/table. Only boxed/clearly marked solutions and filled in figures/tables will be considered for grading. There is no extension possible for this assignment.*

Name: _____

SID: _____

Collaborators (Name, SID):

_____

Problem 1: VLIW machines

In this problem, we consider the execution of a code segment on a VLIW processor. The code we consider is the SAXPY kernel, which scales a single-precision vector X by a single-precision constant A, adding this quantity to a single-precision vector Y.

```
for(i = 0; i < N; i++) {
  Y[i] = Y[i] + A*X[i];
}
```

```
loop:   1.flw   f1, 0(x1)       # f1 = X[i]
        2.fmul f2, f0, f1       # A * X[i]
        3.flw   f3, 0(x2)       # f3 = Y[i]
        4.fadd f4, f2, f3       # f4 = Y[i] + A*X[i]
        5.fsw   f4, 0(x2)       # Y[i] = f4
        6.addi x1, x1, 4        # bump pointer
        7.addi x2, x2, 4        # bump pointer
        8.bne  x1, x3, loop     # loop
```

Now we have a VLIW machine with seven execution units:

- two ALU units, latency one cycle, also used for branch operations
- three memory units, latency three cycles, fully pipelined, each unit can perform either a store or a load
- two FPU units, latency four cycles, fully pipelined, one unit can perform **`fadd`** operations, the other **`fmul`** operations.

Below is the format of a VLIW instruction:

Int Op 1	Int Op 2	Mem Op 1	Mem Op 2	Mem Op 3	FP Add	FP Mul

Our machine has no interlocks. The result of an operation is written to the register file immediately after it has gone through the corresponding execution unit: one cycle after issue for ALU operations, three cycles for memory operations and four cycles for FPU operations. The old values can be read from the registers until they have been overwritten.

When writing code for this machine, you may:

1) Assume the arrays are long (> 32 elements)
2) Assume the arrays have an even number of elements
3) Reorder instructions and change immediates as long as the code is functionally correct

Do not count floating point memory operations towards FLOPS in this problem.

Problem 1.A: No Code Optimization

Schedule instructions for the VLIW machine in Table P4.2-1 without loop unrolling and software pipelining. What is the throughput of the loop in the code in floating point operations per cycle (FLOPS/cycle)?

Throughput = 2 / 12 = 1/6 FLOPS/cycle

Problem 1.B: Loop Unrolling

Schedule instructions for the VLIW machine in Table P4.2-2 only with loop unrolling. Write the assembly code by unrolling the loop once. What is the throughput of the loop in the code in floating point operations per cycle (FLOPS/cycle)? What is the speedup over Problem 1.A?

```
loop: 1.flw f1, 0(x1) # f1 = X[i]
2.fmul f2, f0, f1 # A * X[i]
3.flw f3, 0(x2) # f3 = Y[i]
4.fadd f4, f2, f3 # f4 = Y[i] + A*X[i]
5.fsw f4, 0(x2) # C[i] = f4
6.flw f5, 4(x1) # f5 = X[i+1]
7.fmul f6, f0, f5 # A * X[i+1]
8.flw f7, 4(x2) # f7 = Y[i+1]
9.fadd f8, f6, f7 # f8 = Y[i+1] + A*X[i+1]
10.fsw f8, 4(x2) # C[i+1] = f8
21.addi x1, x1, 8 # bump pointer
22.addi x2, x2, 8 # bump pointer
23.bne x1, x3, loop # loop
```

Throughput = 4 / 13 FLOPS/cycle
Speed up = (4 / 13) / (1 / 6) = 24 / 13 = 1.85

Problem 1.C: Software Pipelining

Schedule instructions for the VLIW machine in Table P4.1-3 only with software pipelining. Include the prologue and the epilogue in Table P4.1-3. What is the throughput of the loop in the code in floating point operations per cycle (FLOPS/cycle)? What is the speedup over Problem 1.A?

Throughput = 2 / 2 = 1 FLOPS
Speed up = (1 / 1) / (1/ 6) = 6

Problem 1.D: Loop Unrolling + Software Pipelining

Schedule instructions for the VLIW machine in Table P4.1-4 with both loop unrolling and software pipelining. Unroll the loop once as in Problem 1.B. Include the prologue and the epilogue in Table P4.1-4. What is the throughput of the loop in the code in floating point operations per cycle (FLOPS/cycle)? What is the speedup over Problem 1.A?

Throughput = 4 / 2 FLOPS
Speed up = (4 / 2) / (1 / 6) = 12

ALU	ALU2	MU1	MU2	MU3	FADD	FMUL
		`flw f1,0(x1)`	`flw f3,0(x2)`			
						`fmul f2,f0,f1`
					`fadd f4,f2,f3`	
	`addi x1,x1,4`					
`bne x1,x3,loop`	`addi x2,x2,4`		`fsw f4,0(x2)`			

Table P4.1-1: Code Scheduling without Optimization

ALU1	ALU2	MU1	MU2	MU3	FADD	FMUL
		flw f1,0(x1)	flw f3,0(x2)			
		flw f5,4(x1)	flw f7,4(x2)			
						fmul f2,f0,f1
						fmul f6,f0,f5
					fadd f4,f2,f3	
					fadd f8,f6,f7	
	addi x1,x1,8			fsw f4,0(x2)		
bne x1,x3,loop	addi x2,x2,8			fsw f8,0(x2)		

Table P4.1-2: Code Scheduling with Loop Unrolling

ALU1	ALU2	MU1	MU2	MU3	FADD	FMUL
	addi x1,x1,4	flw f1,0(x1)				
	addi x1,x1,4	flw f1,0(x1)				
						fmul f2,f0,f1
addi x2,x2,4	addi x1,x1,4	flw f1,0(x1)	flw f3,0(x2)			
						fmul f2,f0,f1
addi x2,x2,4	addi x1,x1,4	flw f1,0(x1)	flw f3,0(x2)			
					fadd f4,f2,f3	fmul f2,f0,f1
addi x2,x2,4	addi x1,x1,4	flw f1,0(x1)	flw f3,0(x2)			
					fadd f4,f2,f3	fmul f2,f0,f1
addi x2,x2,4	addi x1,x1,4	flw f1,0(x1)	flw f3,0(x2)			
	bne x1,x3,loop			fsw f4,-16(x2)	fadd f4,f2,f3	fmul f2,f0,f1
addi x2,x2,4			flw f3,0(x2)			
				fsw f4,-16(x2)	fadd f4,f2,f3	fmul f2,f0,f1
addi x2,x2,4			flw f3,0(x2)			
				fsw f4,-16(x2)	fadd f4,f2,f3	
				fsw f4,-12(x2)	fadd f4,f2,f3	
				fsw f4,-8(x2)		
				fsw f4,-4(x2)		

Table P4.1-3: Code Scheduling with Software Pipelining

ALU1	ALU2	MU1	MU2	MU3	FADD	FMUL
	addi x1,x1,8	flw f1,0(x1)				
		flw f5,-4(x1)				
	addi x1,x1,8	flw f1,0(x1)				
		flw f5,-4(x1)				fmul f2,f0,f1
	addi x1,x1,8	flw f1,0(x1)	flw f3,0(x2)			fmul f6,f0,f5
addi x2,x2,8		flw f5,-4(x1)	flw f7,4(x2)			fmul f2,f0,f1
	addi x1,x1,8	flw f1,0(x1)	flw f3,0(x2)			fmul f6,f0,f5
addi x2,x2,8		flw f5,-4(x1)	flw f7,4(x2)		fadd f4,f2,f3	fmul f2,f0,f1
	addi x1,x1,8	flw f1,0(x1)	flw f3,0(x2)		fadd f8,f6,f7	fmul f6,f0,f5
addi x2,x2,8		flw f5,-4(x1)	flw f7,4(x2)		fadd f4,f2,f3	fmul f2,f0,f1
	addi x1,x1,8	flw f1,0(x1)	flw f3,0(x2)		fadd f8,f6,f7	fmul f6,f0,f5
addi x2,x2,8		flw f5,-4(x1)	flw f7,4(x2)	fsw f4,-24(x2)	fadd f4,f2,f3	fmul f2,f0,f1
	addi x1,x1,8	flw f1,0(x1)	flw f3,0(x2)	fsw f8,-28(x2)	fadd f8,f6,f7	fmul f6,f0,f5
addi x2,x2,8	bne x1,x3,loop	flw f5,-4(x1)	flw f7,4(x2)	fsw f4,-24(x2)	fadd f4,f2,f3	fmul f2,f0,f1
			flw f3,0(x2)	fsw f8,-28(x2)	fadd f8,f6,f7	fmul f6,f0,f5
addi x2,x2,8			flw f7,4(x2)	fsw f4,-24(x2)	fadd f4,f2,f3	fmul f2,f0,f1
			flw f3,0(x2)	fsw f8,-28(x2)	fadd f8,f6,f7	fmul f6,f0,f5
addi x2,x2,8			flw f7,4(x2)	fsw f4,-24(x2)	fadd f4,f2,f3	
				fsw f8,-28(x2)	fadd f8,f6,f7	
				fsw f4,-24(x2)	fadd f4,f2,f3	
				fsw f8,-20(x2)	fadd f8,f6,f7	
				fsw f4,-16(x2)		
				fsw f8,-12(x2)		
				fsw f4,-8(x2)		
				fsw f8,-4(x2)		

Table P4.1-4: Code Scheduling with Loop Unrolling and Software Pipelining

Problem 2: Vector Machines

In this problem, we analyze the performance of vector machines. We start with a baseline vector processor with the following features:

- **32 elements per vector register**
- **8 lanes**
- **One ALU per lane: 1 cycle latency**
- **One load/store unit per lane: 4 cycle latency, fully pipelined**
- **No dead time**
- **No support for chaining**
- **Scalar instructions execute on a separate 5-stage pipeline**

To simplify the analysis, we assume a magic memory system with no bank conflicts and no cache misses.

We consider execution of the following loop:

C code

```
for (i = 0; i < N; i++) {
  C[i] = A[i] + B[i] – 1;
}
```

```
loop:    1.LV    V1, (x1)         # load A
         2.LV    V2, (x2)         # load B
         3.ADDV  V3, V1, V2       # A + B
         4.SUBVS V4, V3, x4       # subtract x4 = 1
         5.SV    V4, (x3)         # store C
         6.ADDI  x1, x1, 128      # bump pointer
         7.ADDI  x2, x2, 128      # bump pointer
         8.ADDI  x3, x3, 128      # bump pointer
         9.SUBI  x5, x5, 32       # i++ (x5 = N)
        10.BNEZ  x5, loop         # loop
```

Problem 2.A: Simple Vector Processor

Complete the pipeline diagram in Table P4.4-1 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.

Inst #	cycle																																										
	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
1	F	D	R	M1	M2	M3	M4	W																																
1				R	M1	M2	M3	M4	W																															
1					R	M1	M2	M3	M4	W																													
1						R	M1	M2	M3	M4	W																												
2		F	D	—	—	—	R	M1	M2	M3	M4	W																										
2								R	M1	M2	M3	M4	W																									
2									R	M1	M2	M3	M4	W																								
2										R	M1	M2	M3	M4	W																							
3			F	D	—	—	—	—	—	—	—	—	—	—	—	R	X1	W																				
3																	R	X1	W																		
3																		R	X1	W																	
3																			R	X1	W																
4				F	D	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	R	X1	W														
4																							R	X1	W												
4																								R	X1	W											
4																									R	X1	W										
5					F	D	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	R	M1	M2	M3	M4							
5																													R	M1	M2	M3	M4							
5																														R	M1	M2	M3	M4						
5																															R	M1	M2	M3	M4					
6						F	D	X	M	W																												
7							F	D	X	M	W																											
8								F	D	X	M	W																									
9									F	D	X	M	W																								
10										F	D	X	M	W																							
1											F	D	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	R	M1	M2	M3	M4	W			
1																																R	M1	M2	M3	M4	W		
1																																	R	M1	M2	M3	M4	W	
1																																		R	M1	M2	M3	M4	W

Table P4.2-1: Vector Pipeline Diagram (8 Lanes without Chaining)

Problem 2.B: Hardware Optimization (Chaining)

In this question, we analyze the performance benefits of chaining and additional lanes. 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*). For this question, we always assume 32 elements per vector register, so there are 4 elements per lane with 8 lanes, and 1 element per lane with 32 lanes.

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 3.A, Inst #1(LV) begins execution in cycle 3, Inst #2(LV) in cycle 7 and Inst #3(ADDV) in cycle 16. Therefore, there are 4 cycles between Inst #1 and Inst #2 and 9 cycles between Inst #2 and Inst #3.

First, fill in Table P4.3-2 for 8 lanes with chaining, Table P4.3-3 for 16 lanes with chaining, and Table P4.3-4 for 32 lanes with chaining.

Note that, with 8 lanes and chaining, Inst #4(SUBVS) cannot issue 2 cycles after Inst #3(ADDV) because there is only one ALU per lane. This would be possible in the absence of a structural hazard.

Also, complete the following table. The first row corresponds to the baseline 8-lane vector processor with no chaining. The second row adds flexible chaining to the baseline processor, and the last two rows increase the number of lanes from 8 to 32.

Vector Processor Configuration	Number of cycles between successive vector instructions					Total cycles per vector loop iteration
	#1(LV) #2(LV)	#2(LV) #3(ADDV)	#3(ADDV) #4(SUBVS)	#4(SUBVS) #5(SV)	#5(SV) #1(LV)	
8 lanes, no chaining	4	9	6	6	4	29
8 lanes, chaining	4	5	4	2	4	19
16 lanes, chaining	2	5	2	2	2	13
32 lanes, chaining	1	5	2	2	1	11

Inst #	Cycle																																							
	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
1	F	D	R	M1	M2	M3	M4	W																																
1				R	M1	M2	M3	M4	W																																
1					R	M1	M2	M3	M4	W																															
1						R	M1	M2	M3	M4	W																														
2		F	D	—	—	—	R	M1	M2	M3	M4	W																												
2								R	M1	M2	M3	M4	W																											
2									R	M1	M2	M3	M4	W																										
2										R	M1	M2	M3	M4	W																									
3			F	D	—	—	—	—	—	—	—	R	X1	W																										
3													R	X1	W																								
3														R	X1	W																							
3															R	X1	W																						
4				F	D	—	—	—	—	—	—	—	—	—	—	R	X1	W																						
4																	R	X1	W																				
4																		R	X1	W																			
4																			R	X1	W																		
5					F	D	—	—	—	—	—	—	—	—	—	—	—	R	M1	M2	M3	M4																			
5																			R	M1	M2	M3	M4																
5																				R	M1	M2	M3	M4															
5																					R	M1	M2	M3	M4														
6						F	D	X	M	W																														
7							F	D	X	M	W																													
8								F	D	X	M	W																												
9									F	D	X	M	W																										
10										F	D	X	M	W																									
1											F	D	—	—	—	—	—	—	—	—	—	R	M1	M2	M3	M4	W													
1																							R	M1	M2	M3	M4	W											
1																								R	M1	M2	M3	M4	W										
1																									R	M1	M2	M3	M4	W									

Table P4.2-2: 8 Lanes with Chaining

Inst #	Cycle																																							
	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
1	F	D	R	M1	M2	M3	M4	W																																
1				R	M1	M2	M3	M4	W																																
2		F	D	—	R	M1	M2	M3	M4	W																														
2						R	M1	M2	M3	M4	W																														
3			F	D	—	—	—	—	—	R	X1	W																												
3											R	X1	W																										
4				F	D	—	—	—	—	—	—	R	X1	W																										
4													R	X1	W																								
5					F	D	—	—	—	—	—	—	—	R	M1	M2	M3	M4																						
5															R	M1	M2	M3	M4																				
6						F	D	X	M	W																														
7							F	D	X	M	W																												
8								F	D	X	M	W																											
9									F	D	X	M	W																										
10										F	D	X	M	W																									
1											F	D	—	—	—	R	M1	M2	M3	M4	W																		
1																	R	M1	M2	M3	M4	W																	

Table P4.2-3: 16 Lanes with Chaining

Inst #	Cycle																																							
	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
1	F	D	R	M1	M2	M3	M4	W																																
2		F	D	R	M1	M2	M3	M4	W																															
3			F	D	—	—	—	—	R	X1	W																													
4				F	D	—	—	—	—	—	R	X1	W																											
5					F	D	—	—	—	—	—	—	R	M1	M2	M3	M4																							
6						F	D	X	M	W																													
7							F	D	X	M	W																												
8								F	D	X	M	W																											
9									F	D	X	M	W																										
10										F	D	X	M	W																									
1											F	D	—	R	M1	M2	M3	M4	W																				

Table P4.2-3: 32 Lanes with Chaining