Problem M3.16: Vector Machines [?? Hours]

Problem M3.16.A

Consider the implementation of the C-code on the vector machine that executes in a minimum number of cycles. Assuming the following initial values, insert vector instructions to complete the implementation.

- R1 points to A[0]
- R2 points to B[0]
- R3 points to C[0]
- R4 contains the value 328

```
     ANDI R5, R4, 31       # 328 mod 32
     MTC1 VLR, R5          # set VLR to remainder
loop:
     LV   V1, R1           # load A
     LV   V2, R2           # load B
     LV   V3, R3           # load C
     MULV V4, V2, V1       # A * B
     ADDV V5, V3, V4       # C + A
     SV   V4, R1           # store A
     SV   V5, R3           # store C
     SLL  R7, R5, 2
     ADD  R1, R1, R7       # increment A ptr
     ADD  R2, R2, R7       # increment B ptr
     ADD  R3, R3, R7       # increment C ptr
     SUB  R4, R4, R5       # update loop counter
     LI   R5, 32           # reset VLR to max
     MTC1 VLR, R5
     BGTZ R4, loop
```

Problem M3.16.B

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**) or the MUL (**Y**), and the result is written back (**W**) to the vector register file. Assume that there is no structural conflict on the writeback port. A stalled vector instruction does not block a scalar instruction from executing.

LV_1 and LV_2 refer to the first and second LV instructions in the loop.

instr.	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
LV_1	F	D	R	M1	M2	M3	M4	W																																		
LV_1				R	M1	M2	M3	M4	W																																	
LV_1					R	M1	M2	M3	M4	W																																
LV_1						R	M1	M2	M3	M4	W																															
LV_2		F	D	—	—	—	R	M1	M2	M3	M4	W																														
LV_2								R	M1	M2	M3	M4	W																												
LV_2									R	M1	M2	M3	M4	W																											
LV_2										R	M1	M2	M3	M4	W																										
LV_3			F	D	—	—	—	—	—	—	R	M1	M2	M3	M4	W																										
LV_3												R	M1	M2	M3	M4	W																								
LV_3													R	M1	M2	M3	M4	W																								
LV_3														R	M1	M2	M3	M4	W																								
MULV				F	D	—	—	—	—	—	—	—	—	—	—	R	Y1	Y2	W																								
MULV																	R	Y1	Y2	W																						
MULV																		R	Y1	Y2	W																					
MULV																			R	Y1	Y2	W																				
ADDV					F	D	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	R	X1	W																	
ADDV																								R	X1	W															
ADDV																									R	X1	W														
ADDV																										R	X1	W													
SV_1						F	D	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	R	M1	M2	M3	M4	W														
SV_1																							R	M1	M2	M3	M4	W													
SV_1																								R	M1	M2	M3	M4	W												
SV_1																									R	M1	M2	M3	M4	W											
SV_2							F	D	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	R	M1	M2	M3	M4	W								
SV_2																													R	M1	M2	M3	M4	W						
SV_2																														R	M1	M2	M3	M4	W					
SV_2																															R	M1	M2	M3	M4	W				

Problem M3.16.C

instr.	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
LV_1	F	D	R	M1	M2	M3	M4	W																																		
LV_1				R	M1	M2	M3	M4	W																																	
LV_1					R	M1	M2	M3	M4	W																																
LV_1						R	M1	M2	M3	M4	W																															
LV_2		F	D	—	—	—	R	M1	M2	M3	M4	W																														
LV_2								R	M1	M2	M3	M4	W																												
LV_2									R	M1	M2	M3	M4	W																											
LV_2										R	M1	M2	M3	M4	W																										
LV_3			F	D	—	—	—	—	—	—	R	M1	M2	M3	M4	W																										
LV_3												R	M1	M2	M3	M4	W																								
LV_3													R	M1	M2	M3	M4	W																							
LV_3														R	M1	M2	M3	M4	W																						
MULV				F	D	—	—	—	—	—	—	R	Y1	Y2	W																										
MULV													R	Y1	Y2	W																								
MULV														R	Y1	Y2	W																							
MULV															R	Y1	Y2	W																						
ADDV					F	D	—	—	—	—	—	—	—	—	—	R	X1	W																							
ADDV																	R	X1	W																					
ADDV																		R	X1	W																				
ADDV																			R	X1	W																			
SV_1						F	D	—	—	—	—	—	—	—	R	M1	M2	M3	M4	W																						
SV_1																R	M1	M2	M3	M4	W																			
SV_1																	R	M1	M2	M3	M4	W																		
SV_1																		R	M1	M2	M3	M4	W																	
SV_1							F	D	—	—	—	—	—	—	—	—	—	R	M1	M2	M3	M4	W																		
SV_2																			R	M1	M2	M3	M4	W																
SV_2																				R	M1	M2	M3	M4	W															
SV_2																					R	M1	M2	M3	M4	W														
				1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19																		

Problem M3.16.D

What is the performance (flops/cycle) of the program with chaining?

2*32/19

Problem M3.16.E

Would loop unrolling of the assembly code improve performance without chaining? Explain. (You may rearrange the instructions when performing loop unrolling.)

Yes. We can overlap some of the vector memory instructions from different loops.

Problem M3.17: Vector Machines [?? Hours]

Problem M3.17.A

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.

LV_1 and LV_2 refer to the first and second LV instructions in the loop.

instr.	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
LV_1	F	D	R	M1	M2	M3	M4	W																																
LV_1				R	M1	M2	M3	M4	W																															
LV_1					R	M1	M2	M3	M4	W																														
LV_1						R	M1	M2	M3	M4	W																													
LV_2		F	D	—	—	—	R	M1	M2	M3	M4	W																												
LV_2								R	M1	M2	M3	M4	W																											
LV_2									R	M1	M2	M3	M4	W																										
LV_2										R	M1	M2	M3	M4	W																									
ADDV			F	D	—	—	—	—	—	—	—	—	—	—	—	R	X1	W																						
ADDV																	R	X1	W																					
ADDV																		R	X1	W																				
ADDV																			R	X1	W																			
SUBVS				F	D	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	R	X1	W																
SUBVS																							R	X1	W															
SUBVS																								R	X1	W														
SUBVS																									R	X1	W													
SV					F	D	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	R	M1	M2	M3	M4								
SV																													R	M1	M2	M3	M4							
SV																														R	M1	M2	M3	M4						
SV																															R	M1	M2	M3	M4					
ADDI						F	D	X	M	W																														
ADDI							F	D	X	M	W																													
ADDI								F	D	X	M	W																												
SUBI									F	D	X	M	W																											
BNEZ										F	D	X	M	W																										
LV_1											F	D	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	R	M1	M2	M3	M4	W			
LV_1																																	R	M1	M2	M3	M4	W		
LV_1																																		R	M1	M2	M3	M4	W	
LV_1																																			R	M1	M2	M3	M4	W

Problem M3.17.B

Vector processor configuration	Number of cycles between successive vector instructions					Total cycles per vector loop iter.
	LV_1, LV_2	LV_2, ADDV	ADDV, SUBVS	SUBVS, SV	SV, LV_1	
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**

Note, with 8 lanes and chaining, the SUBVS can not issue 2 cycles after the ADDV because there is only one ALU per lane. Also, since chaining is done through the register file, 2 cycles are required between the ADDV and SUBVS and between the SUBVS and SV even with 32 lanes (if bypassing was provided, only 1 cycle would be necessary).

Problem M3.17.C

Instr. Number	Instruction
I1	LV V1, R1
I2	LV V2, R2
I6	ADDI R1, R1, 128
I7	ADDI R2, R2, 128
I10	LV V5, R1
I11	LV V6, R2
I3	ADDV V3, V1, V2
I4	SUBVS V4, V3, R4
I5	SV R3, V4
I12	ADDV V7, V5, V6
I13	SUBVS V8, V7, R4
I8	ADDI R3, R3, 128
I14	SV R3, V8
I15	ADDI R1, R1, 128
I16	ADDI R2, R2, 128
I17	ADDI R3, R3, 128
I9	SUBI R5, R5, 32
I18	SUBI R5, R5, 32
I19	BNEZ R5, loop

This is only one possible solution. Scheduling the second iteration's LV's (I10 and I11) before the first iteration's SV (I5) allows the LV's to execute while the load/store unit would otherwise be idle. Interleaving instructions from the two iterations (for example, if I12 were placed between I3 and I4) could hide the functional unit latency seen with no chaining. However, doing so would delay the first SV (I5), and hence, increase the overall latency. This tension makes the optimal solution very tricky to find. Note that to preserve the instruction dependencies, I6 and I7 must execute before I10 and I11, and I8 must execute after I5 and before I14.

Problem M3.18: Vectorizing `memcpy` and `strcpy` [?? Hours]

Problem M3.18.A

Because there is only one load/store unit, SV instruction should wait at least till the last element of the LV instruction is issued. Since there is only one lane, each SV and LV instruction takes 32 cycles to issue. In steady state, it takes 32 (LV) + 10 (dead time) + 32 (SV) + 10 (dead time) cycles per 32 elements, and 2.62 cycles per element. All scalar instructions can be overlapped with SV.

Problem M3.18.B

We can vectorize strcpy using SEQSV and CLZM. The algorithm is as follows. First, we load 32 elements. Second, we use SEQSV to check whether each element has '\0' or not. Third, we use CLZM to count the number of the elements before the first '\0' in the vector and set the vector length to that number. Then, we do a vector store. If no element has '\0' (i.e. the number is 32), we go back to the first step and load the next 32 elements. If a vector has '\0', strcpy ends. As discussed in the function definition, our strcpy copies one word at a time, and assumes that the string is word-aligned with the terminating character of 32-bit '\0'.

```
    ADD     R5,R1,R0     ; store destination address in R5
    ADD     R4,R2,R0     ; store source address in R4
    ADDI    R6,R0,#32
    MTC1    VLR,R6       ; set vector length to 32
    CVM
    MOVI2FP F0,R0
loop:
    LV      V1,R4
    ADDI    R4,R4,#128   ; bump source pointer
    SEQSV   F0,V1        ; setup the mask register
    CLZM    R6,VM        ; number elements before '\0'
    MTC1    VLR,R6
    SV      R5,V1
    ADDI    R5,R5,#128   ; bump destination pointer
    SUBI    R7,R6,#32    ;
    BEQZ    R7,loop      ; if no element has '\0' goto loop
    SLLI    R6,R6,#2     ; move destination pointer to
    SUBI    R5,R5,#128   ; the end of the string
    ADD     R5,R5,R6     ; copy '\0'
```

Problem M3.18.C

Without vector chaining, `strcpy` takes more cycles per element than `memcpy` since it has one additional vector instruction, `SEQSV`. It takes 32+10 (`LV`) + 32 (`SEQSV`) + 1 (`CLZM`) + 1 (`MTC1`) + 32 (`SV`) + 10 (dead time) = 118 cycles per 32 elements or 3.69 cycles per element.

With vector chaining, the first element of V1 can be bypassed to `SEQSV` instruction after 10 cycles. Store can be executed only after we get the value of VLR, that is, after `SEQSV`, `CLZM`, and MTC1. Therefore, it takes 10 (`LV`) + 32 (`SEQSV`) + 1 (`CLZM`) + 1 (`MTC1`) + 32 (`SV`) + 10 (dead time) = 86 cycles per 32 elements or 2.69 cycles per element.

In `memcpy`, both vector instructions (`SV` and `LV`) use the same functional unit. Therefore, the execution of two instructions cannot be overlapped even with vector chaining. Copying each element takes 2.62 cycles as in M3.18.A. With vector chaining, the performance of `strcpy` is comparable to that of `memcpy`.

Problem M3.19: Performance of Vector Machines [?? Hours]

Problem M3.19.A

With 8 lanes, a 2-cycle dead time and no vector chaining, we get the following pipeline diagram.

	Cycle																					
	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**18**	**19**	**20**	**21**	**22**	**23**
I1	F	D	R	X1	X2	W																	
I1				R	X1	X2	W																
I1					R	X1	X2	W															
I1						R	X1	X2	W														
I2		F	D	D	D	D	***D***	***D***	R	X1	X2	W											
I2										R	X1	X2	W										
I2											R	X1	X2	W									
I2												R	X1	X2	W								
I3			F	D	D	D	D	D	D	D	D	D	D	D	D	R	X1	X2	X3	W			
I3																	R	X1	X2	X3	W		
I3																		R	X1	X2	X3	W	
I3																			R	X1	X2	X3	W

Since each vector has 32 elements, and there are 8 lanes, the vector register file needs to be read 4 times for each instruction. Although I2 does not need the results of I1, both instructions use the vector add unit, so I2 must wait until after I1 completes its last read, plus an additional 2 cycles for dead time before beginning its first read. And because there is no chaining, I3, which is dependent on I2, needs to wait until I2 has finished its last write back before beginning its first read.

The execution time is 18 cycles (from cycle 6 to cycle 23, inclusive).

Problem M3.19.B

With 8 lanes, no dead time and flexible chaining, we get the following pipeline diagram.

	Cycle																
	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**17**
I1	F	D	R	X1	X2	W											
I1				R	X1	X2	W										
I1					R	X1	X2	W									
I1						R	X1	X2	W								
I2		F	D	D	D	D	R	X1	X2	W							
I2								R	X1	X2	W						
I2									R	X1	X2	W					
I2										R	X1	X2	W				
I3			F	D	D	D	D	D	D	R	X1	X2	X3	W			
I3											R	X1	X2	X3	W		
I3												R	X1	X2	X3	W	
I3													R	X1	X2	X3	W

With no dead time, I2 can issue its first read after the last read of I1. And with flexible chaining, I3 can begin its first read in the same cycle as the first write of I2.

The execution time is 12 cycles (from cycle 6 to cycle 17, inclusive).

Problem M3.19.C

With 16 lanes, no dead time and flexible chaining, we get the following pipeline diagram.

	Cycle												
	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**
I1	F	D	R	X1	X2	W							
I1				R	X1	X2	W						
I2		F	D	D	R	X1	X2	W					
I2						R	X1	X2	W				
I3			F	D	D	D	D	R	X1	X2	X3	W	
I3									R	X1	X2	X3	W

Since each vector has 32 elements, and there are 16 lanes, the vector register file needs to be read 2 times for each instruction.

The execution time is 8 cycles (from cycle 6 to cycle 13, inclusive).