Problem M9.1: Memory Dependencies (Fall 2006)

Problem M9.1.A

For each of the following 3 instruction sequences, please give the condition for a memory dependency to exist or explain why there cannot be a dependency. Assume that there is no memory aliasing (i.e. all virtual memory pages are mapped to unique physical pages).

Instruction Pair	Condition under which Memory Dependency occurs
SW R2, 0(R3) LW R5, 0(R4)	
SW R2, 0(R3) LW R5, 4(R3)	
SW R2, 0(R3) LW R5, 4096(R3)	

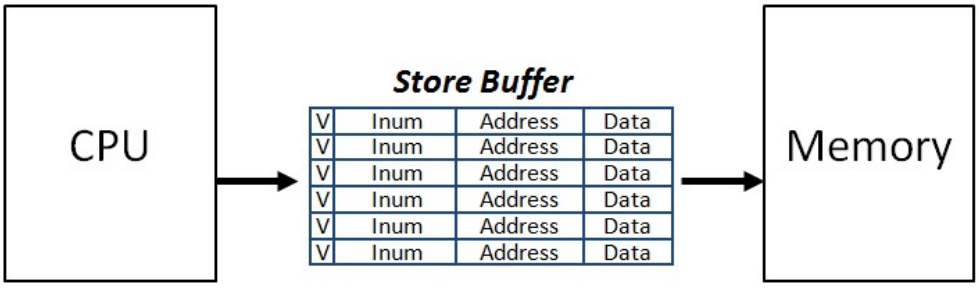
Problem M.9.1.B

If we allow memory aliasing to occur, how will it affect your answer in part A? Assume that the page size is 4KB and that the machine is byte addressed.

Instruction Pair	Condition under which Memory Dependency occurs
SW R2, 0(R3) LW R5, 0(R4)	
SW R2, 0(R3) LW R5, 4(R3)	
SW R2, 0(R3) LW R5, 4096(R3)	

Problem M9.2: Vector Store Buffers (Fall 2010)

Ben Bitdiddle designed an out-of-order vector machine with store buffers. This machine executes memory operations (both load and store) "in order", although other instructions can be executed out-of-order. (All the instructions are committed "in order".) There is a load/store issue queue to maintain the execution order of all the memory operations. Every load must check if the value it needs is in the store buffer, and to determine the proper value, every instruction is assigned a unique instruction number (Inum).



The machine has 4 vector registers, v1 through v4, and a special-purpose vector length register, vlr, which can be used like a general purpose register. Any MIPS integer operation (except jumps/branches) can be applied to the first vlr elements of one or more vector registers by prefixing a V to it; for example, if vlr is 16, the instruction VLW v1, 0 (r1) loads 16 consecutive words starting at the address in r1 into the first 16 locations in v1, and similarly the instruction VADD v3, v1, v2 adds each of the first 16 elements of v1 to the corresponding element of v2 and puts the result in the corresponding element of v3. Each vector register can hold at most 32 word values, and thus vlr can be at most 32. For the entire part, assume that **vlr has the value of 4**.

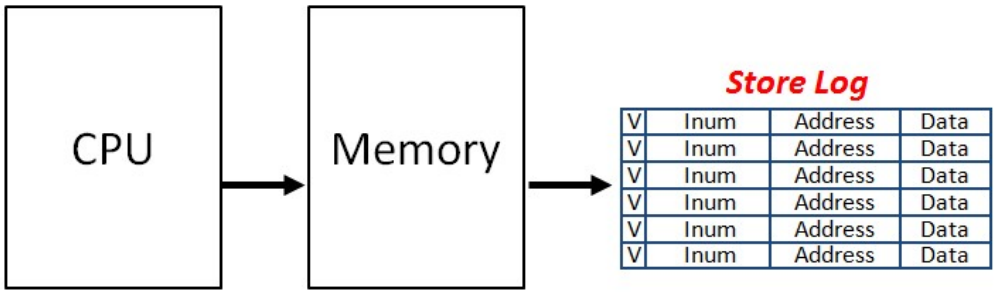
Problem M9.2.A

Suppose 10% of instructions are stores and the average lifetime of instructions in the store buffer is 100 cycles. Assuming the desired throughput of this machine to be 1 instruction per cycle, how many entries will the store buffer be holding at a given time on average?

Does this machine store into memory greedily or lazily?

Problem M9.2.B

Ben did not like the store buffers in the previous design, because the store buffers need to be looked up for every load instruction, and implementing a fast lookup to the buffers was too expensive. Thus, instead of using the store buffers, Ben decided to directly update the memory during execution (before the store instruction actually commits), and keep the old values in "**store logs**". **Each entry in the store logs consists of a valid bit, Inum, memory address and data value**. The data field holds the value that was in the memory before the store to the location writes to the memory.



Is this a greedy update or a lazy update?

Assume an arithmetic exception occurred, and thus, the processor state and memory need to be recovered appropriately. Which machine design (store buffer machine or store log machine) has the higher recovery cost, and why?

Problem M9.2.C

Now, we want to investigate how Ben's machine with the store logs changes the processor state. The diagram below shows the initial processor state. Assume that the Inum space is infinite.

[Initial State]

Vector Registers

V1	5,0,5,0,5,...
V2	1,1,1,1,1,...
V3	0,5,0,5,3,...
V4	0,1,2,3,4,...

Memory

0x1000	0,1,2,3,4,...
0x2000	5,5,0,0,5,...
0x3000	3,3,3,3,3,...

Store Logs

v	Inum	Address	Data
v	78	0x2000	1,1,1,1,1,...
v	83	0x3000	0,1,4,9,16,...
v	96	0x2000	2,2,2,2,2,...

In the following questions, **we always start from the same initial state**, and you have to update the diagram for each question **separately**, to reflect the processor state after each event has occurred. (The event specified in each question is the only event that takes place for that question.) There are extra blank boxes for vector registers and memory. Write down the new values in these boxes if the value changes due to the event specified in the question. You don't have to repeat the values if they do not change by the specified event. Also, cross out any entries in the store logs which are no longer valid.

Note that the vector length register (vlr) has the value of 4.

(i) The instruction **VADD V3, V1, V2** (Inum:99) is executed and committed.

Vector Registers

V1	5,0,5,0,5,...	
V2	1,1,1,1,1,...	
V3	0,5,0,5,3,...	
V4	0,1,2,3,4,...	

Memory

0x1000	0,1,2,3,4,...	
0x2000	5,5,0,0,5,...	
0x3000	3,3,3,3,3,...	

Store Logs

v	Inum	Address	Data
v	78	0x2000	1,1,1,1,1,...
v	83	0x3000	0,1,4,9,16,...
v	96	0x2000	2,2,2,2,2,...

(ii) The instruction **VST V4, 0x2000** (Inum:100) is executed.

Vector Registers

V1	5,0,5,0,5,...	
V2	1,1,1,1,1,...	
V3	0,5,0,5,3,...	
V4	0,1,2,3,4,...	

Memory

0x1000	0,1,2,3,4,...	
0x2000	5,5,0,0,5,...	
0x3000	3,3,3,3,3,...	

Store Logs

v	Inum	Address	Data
v	78	0x2000	1,1,1,1,1,...
v	83	0x3000	0,1,4,9,16,...
v	96	0x2000	2,2,2,2,2,...

(iii) An arithmetic exception occurs at the instruction ***with the Inum of 81***, and the recovery process takes place. (Do not worry about the vector register recovery.)

Vector Registers

V1	5,0,5,0,5,...	
V2	1,1,1,1,1,...	
V3	0,5,0,5,3,...	
V4	0,1,2,3,4,...	

Memory

0x1000	0,1,2,3,4,...	
0x2000	5,5,0,0,5,...	
0x3000	3,3,3,3,3,...	

Store Logs

v	Inum	Address	Data
v	78	0x2000	1,1,1,1,1,...
v	83	0x3000	0,1,4,9,16,...
v	96	0x2000	2,2,2,2,2,...

Problem M9.2.D

For this machine with store logs, when will the processor clear the log entry, other than on exceptions?

Ben still did not like the fact that the store log requires so much extra storage. Thus, he decided to eliminate the store log and instead, use a direct-mapped writeback cache to hold the values. On a store operation that hits in the cache, the data is written only into the cache. On a cache miss, the data block is first fetched into the cache, and then the new value is written in the cache (write allocate policy). Memory is written only when the entry is evicted. Two new status bits are added in each cache line – "***dirty***" and "***committed***".

- A new bit "***dirty***" is 0 when the cache line has a clean copy (same with memory), and is set to 1 when the cache line is written so that it has a different copy than memory (i.e., memory has a stale copy).

- A new bit "***committed***" is 0 while the instruction that wrote the cache line has not been committed, and is set to 1 when the corresponding store instruction is committed.

As described in the beginning, this machine executes memory operations (both load and store) "in order", although other instructions can be executed out-of-order. (All the instructions are committed "in order".) There is a load/store issue queue to maintain the execution order of all the memory operations.

Now suppose the following signals are given.

Signal	Description
cache(address).tag	Returns the tag stored in the cache for the corresponding address
cache(address).dirty	0: the corresponding cache line has a clean copy (same with memory) / 1: the corresponding cache line has a dirty copy (i.e., memory has a stale copy)
cache(address).committed	0: the instruction that wrote the corresponding cache line is not committed / 1: the instruction that wrote the corresponding cache line is committed
QHead.OpCode	LD / ST (Opcode of the instruction at the head of the load/store issue queue)
QHead.memAddr	the effective memory address of the instruction at the head of the load/store issue queue
TAG(address)	A function that returns the tag of address

<u>For this part, assume the cache always hits.</u>

Problem M9.2.E

Under what state of the cache line can you writeback the line at the address A to memory?

Problem M9.2.F

How should the dirty bit and the committed bit be updated after writeback of the block at address A?

Problem M9.2.G

Alice P Hacker warns Ben that the machine may stall if it finds that it cannot use the cache block. Thus, in order to make progress, the machine may need to explicitly evict and writeback the corresponding cache line.

Write down the Boolean equation for the stall signal for the load/store issue queue.

Stall =

Problem M9.2.H

Write down the Boolean equation for the signal to force eviction (equivalently, writeback) of the cache line so that the machine gets unstalled.

Evict(writeback) =