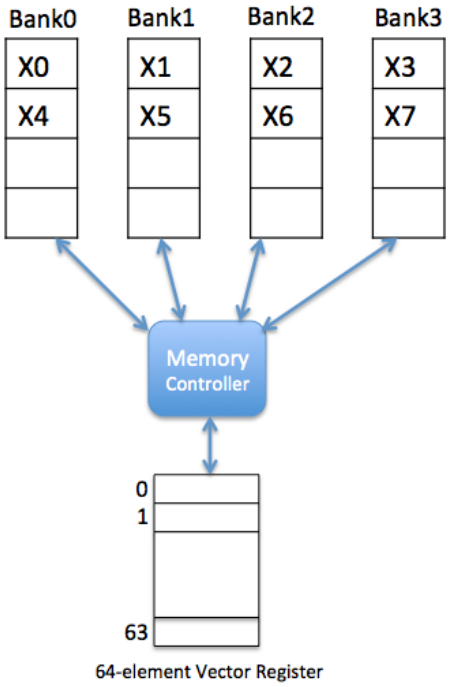
Computer Science 605.411

Final Example Set 1

1. The diagram below represents an interleaved memory system containing four 32-bit wide banks or modules that can operate in parallel. The memory controller connects the memory banks to a vector register over a 32-bit bus and sequences accesses to the banks. If the time to access each bank is 4 clock cycles, what is the total number of clock cycles required to completely fill the 64-element vector register with the 64 elements of the memory resident vector X? To save space, only the first 8 elements of X are shown.



In the first cycle, access to X0 in bank 0 can be started and will require 4 cycles to complete. A separate access to each of the remaining banks (1 through 3) can be started in each of the next 3 clock cycles. At the end of the 4th cycle X0 will be available and access to X4 can be started. Hence, it will take 4 clock cycles to fill the first element in the vector register, and each subsequent element will require one additional clock cycle to fill. Therefore completely filling the vector register will take 4 + 63 = 67 clock cycles.

2. What happens if consecutive vector elements are not in adjacent memory modules?

In this case, instructions such as LVWS (load vector with stride) and SVWS (store vector with stride) would be used to access the vector elements in memory. The distance or displacement in bytes between consecutive vector elements would be placed into a separate scalar register and used in generating the address of each vector element. For example, if consecutive integer elements appear in every other memory bank, the stride would be 2 and the value placed into the stride register would be 2*4 = 8. The following code would load a 5-element vector (VX) with stride 2 into the vector register $V0:

```
ori     $9,$0,8            #place stride displacement into $9
la      $8,VX              # put address of vector Vx into $8
ori     $7,$0,50           # set length register to reference 50 elements
LVWS  $V0,($8),$7,$9   # read vector VX into vector register $V0
```

3. Suppose that a vector processor that employs the memory system described above, has 64 adders each of which requires 1 clock cycle to produce the sum of its two inputs. Considering only the time required to obtain the operands from memory, compute the vector sum, and store the vector result back into memory, what is the total number of clock cycles required to carryout the following high level vector instruction without chaining?

C = A + B # A, B and C are 64-element integer vectors.

Without chaining, both vectors would be loaded into a separate vector register, then the vector sum would be generated, followed by storing the contents of the result vector register into memory. It takes 67 cycles to load one register with vector A and 67 cycles to load vector B into another vector register. Then the 64-element sum would be generated in one cycle by the 64 adders operating in parallel. So it would take 2*67 + 1 = 135 cycles to generate the vector sum. It would then take 67 cycles to store the vector result into memory.

4. How would vectors whose lengths are not the same as the length of the vector registers be handled?

The length register would be set to the size of the vectors that are less than the length of the vector register. If the length of the vector (say N) is greater than the number of elements in the vector register, then it would partitioned into multiple groups of N elements, and the final group would contain any left over elements.

5. An integer scalar value can be loaded into register $5 using the following instruction: lw $5,0($4) where $4 contains the memory address of the scalar value. How does this instruction differ from the following vector load instruction:

LDV $V1, 0($4),$8

where $V1 is a 64-element vector register and $4 contains the memory address of a 64-element integer vector and $8 is the length register indicating the number of elements in the vector?

The scalar load instruction would generate just one memory address and cause the single word at that address to be read from memory and loaded into the scalar register $5. The vector load instruction would generate a series of memory addresses (corresponding to the locations of the vector elements in memory) and would perform a read from each address, with each value read going into the next element within the vector register $V1.

6. How could chaining be used to generate the vector sum C = A + B using just one adder?

Assuming that the elements of the vectors A and B are in different memory modules, the reading of elements from A could be interleaved with reading elements from B (i.e., A0, B0, A1, B1, etc). As soon as the first elements of the vector registers for A and B are filled, the A and B elements could be added together. Each addition would be done in parallel with the reading of the next vector elements from memory.
The following table shows what happens in the first seven cycles:

Cycle	Operations
1	Read A0
2	Read B0
3	Read A1
4	Read B1, A0 available
5	Read A2, B0 available compute C0=A0+B0
6	Read A3, A1 available
7	Read B3, B1 available, compute C1

Hence the first element in the sum is available after 5 cycles, with each subsequent element becoming available in an additional 2 cycles per element for a total of 5 + 2*63 = 131 cycles which is faster and uses 63 fewer adders.

7. A uniprocessor has an L1 and an L2 cache. With no cache misses, the processor achieves a CPI of 1. Suppose that in reality an L1 miss occurs every 50 cycles and that an L2 miss occurs every 500 cycles. The L1 miss penalty is 40 cycles and the L2 miss penalty is 400 cycles. What would be the effective (i.e., average) CPI rating for the processor with these cache miss rates?

The L1 miss rate of 1 every 50 cycles corresponds to one L1 miss for every 50 instructions. The L2 miss rate corresponds to one L2 miss for every 500 instructions. Hence every 50 instructions would take 50 + 40 = 90 cycles; every 500 instructions would take 10*90 + 400 = 1300 cycles. Therefore the average CPI = 1300/500 = 2.6.

8. Three threads A, B and C are executed on a fine-grained multi-threaded version of our MIPS 5-stage pipeline system with no data hazard unit and no forwarding unit and no branch prediction. Thread A consists for 5 instructions (A0, A1, A2, A3 and A4); the instructions in thread B are B0 through B4 and those in thread C are C0 through C4. Show how the three threads would flow through the pipeline assuming no cache misses.

Since the system employs fine-grained multi-threading, the instructions from the three threads would be interleaved in a round-robin fashion within the pipeline. For each clock cycle, an instruction from a different thread would enter the pipeline until all instructions have been executed:

Cycle	Fetch	Decode	Execute	Memory	Write-back
1	A0				
2	B0	A0			
3	C0	B0	A0		
4	A1	C0	B0	A0	
5	B1	A1	C0	B0	A0
6	C1	B1	A1	C0	B0
7	A2	C1	B1	A1	C0
8	B2	A2	C1	B1	A1
9	C2	B2	A2	C1	B1
10	A3	C2	B2	A2	C1
11	B3	A3	C2	B2	A2
12	C3	B3	A3	C2	B2
13	A4	C3	B3	A3	C2
14	B4	A4	C3	B3	A3
15	C4	B4	A4	C3	B3
16		C4	B4	A4	C3
17			C4	B4	A4
18				C4	B4
19					C4

b) If instruction A1 requires the result from instruction A0, how many pipeline bubbles would result?
No bubbles would be required since by the time A1 reaches the decode stage, A0 will have written the result register in the write-back stage.

c) If instruction A2 is a conditional branch instruction that is taken, how many pipeline stages would have to be flushed?
The conditional evaluated by A2 determines which instruction from thread A should be executed after A2. No stages would have to be flushed since the condition tested by A2 would be complete when A2 finishes the execute stage, so the instruction within thread A to which A2 branches can be brought into the pipeline in the next cycle.