Computer Architecture Project

SET -1

Use any language or simulator to solve the below problems and give a detail report for the same:

The transpose of a matrix interchanges its rows and columns

Here is a simple C loop to show the transpose:

```
for (i = 0; i < 3; i++) {
for (j = 0; j < 3; j++) {
output[j][i] = input[i][j];
}
}
```

Assume that both the input and output matrices are stored in the row major order (*row major order* means that the row index changes fastest). Assume that you are executing a 256 × 256 double-precision transpose on a processor with a 16 KB fully associative (don't worry about cache conflicts) least recently used (LRU) replacement L1 data cache with 64 byte blocks. Assume that the L1 cache misses or prefetches require 16 cycles and always hit in the L2 cache, and that the L2 cache can process a request every two processor cycles. Assume that each iteration of the inner loop above requires four cycles if the data are present in the L1 cache. Assume that the cache has a write-allocate fetch-on-write policy for write misses. Unrealistically, assume that writing back dirty cache blocks requires 0 cycles.

1. For the simple implementation given above, this execution order would be nonideal for the input matrix; however, applying a loop interchange optimization would create a nonideal order for the output matrix. Because loop interchange is not sufficient to improve its performance, it must be blocked instead.
   a. What should be the minimum size of the cache to take advantage of blocked execution?
   b. How do the relative number of misses in the blocked and unblocked versions compare in the minimum sized cache above?
   c. Write code to perform a transpose with a block size parameter *B* which uses *B* ×□*B* blocks.
   d. What is the minimum associativity required of the L1 cache

   e. Try out blocked and nonblocked 256 × 256 matrix transpositions on a computer. How closely do the results match your expectations based on what you know about the computer's memory system? Explain any discrepancies if possible.

2. Some applications read a large dataset first, then modify most or all of it. The base MSI coherence protocol will first fetch all of the cache blocks in the Shared state and then be forced to perform an invalidate operation to upgrade them to the Modified state. The additional delay

has a significant impact on some workloads. An additional protocol optimization eliminates the need to upgrade blocks that are read and later written by a single processor. This optimization adds the Exclusive (E) state to the protocol, indicating that no other node has a copy of the block, but it has not yet been modified. A cache block enters the Exclusive state when a read miss is satisfied by memory and no other node has a valid copy. CPU reads and writes to that block proceed with no further bus traffic, but CPU writes cause the coherence state to transition to Modified. Exclusive differs from Modified because the node may silently replace Exclusive blocks (while Modified blocks must be written back to memory). Also, a read miss to an Exclusive block results in a transition to Shared but does not require the node to respond with data (since memory has an up-to-date copy). Implement a new protocol using simulation software for a MESI protocol that adds the Exclusive state and transitions to the base MSI protocol's Modified, Shared, and Invalid states.

3. Design and Evaluation of Advanced Value Prediction Methods in Multi-Issue Superscalar Pipelined Architectures

Computer Architecture Project

SET -2

Use any language or simulator to solve the below problems and give a detail report for the same:

1. With software prefetching it is important to be careful to have the prefetches occur in time for use but also to minimize the number of outstanding prefetches to live within the capabilities of the microarchitecture and minimize cache pollution. This is complicated by the fact that different processors have different capabilities and limitations.

a. Create a blocked version of the matrix transpose with software prefetching.

b. Estimate and compare the performance of the blocked and unblocked transpose codes both with and without software prefetching.

2. Directory protocols are more scalable than snooping protocols because they send explicit request and invalidate messages to those nodes that have copies of a block, while snooping protocols broadcast all requests and invalidates to all nodes. Consider the eight-processor system illustrated in Figure 1 and assume that all caches not shown have invalid blocks. For each of the sequences below, identify which nodes (chip/processor) receive each request and invalidate. Use a simulator to solve the above problem.

    a. P0,0: write 100 <-- 80

    b. P0,0: write 108 <-- 88

    c. P0,0: write 118 <-- 90

    d. P1,0: write 128 <-- 98

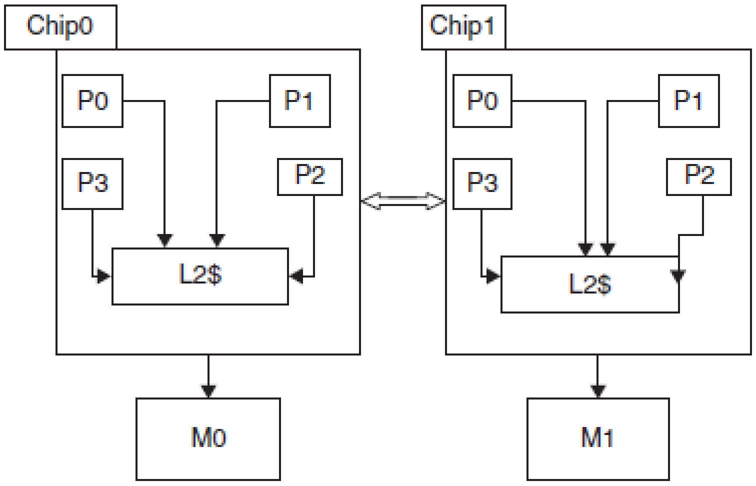


Figure 1.

3. Simulate and Evaluate cache behavior of networking applications or algorithms, with modification to exploit caches and memory hierarchies

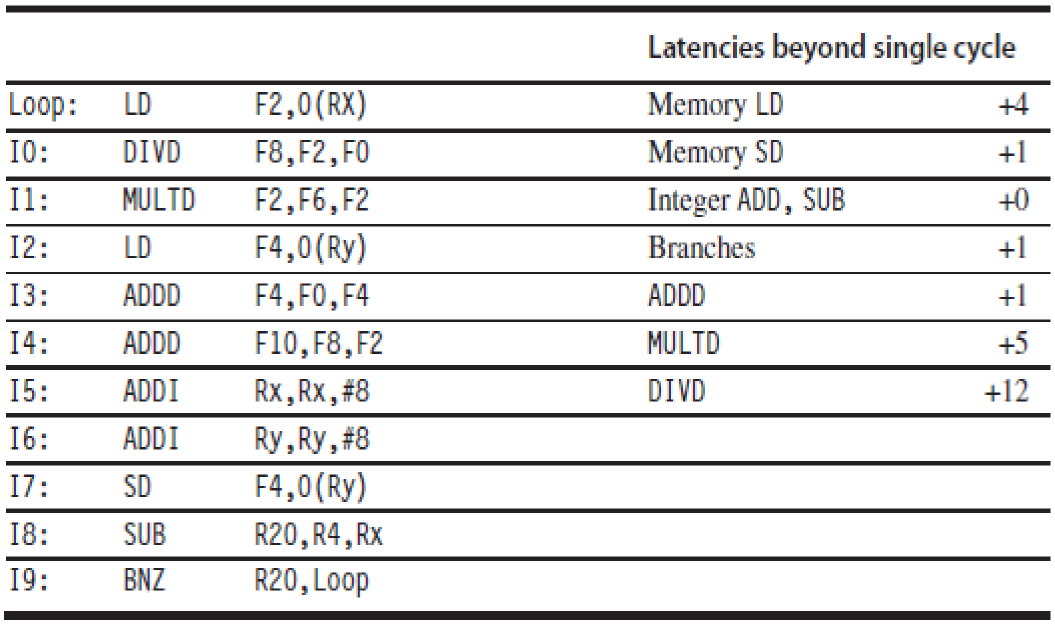
Computer Architecture Project

SET -3

Use any language or simulator to solve the below problems and give a detail report for the same:

1. Can you think of a way to test some of the characteristics of an instruction cache using a program? Hint: The compiler may generate a large number of nonobvious instructions from a piece of code. Try to use simple arithmetic instructions of known length in your instruction set architecture (ISA).

2. Think about what latency numbers really mean—they indicate the number of cycles a given function requires to produce its output, nothing more. If the overall pipeline stalls for the latency cycles of each functional unit, then you are at least guaranteed that any pair of back-to-back instructions (a "producer" followed by a "consumer") will execute correctly. But not all instruction pairs have a producer/consumer relationship. Sometimes two adjacent instructions have nothing to do with each other. How many cycles would the loop body in the code sequence in Figure given below require if the pipeline detected true data dependences and only stalled on those, rather than blindly stalling everything just because one functional unit is busy? Show the code with <stall> inserted where necessary to accommodate stated latencies. (Hint: An instruction with latency +2 requires two <stall> cycles to be inserted into the code sequence. Think of it this way: A one-cycle instruction has latency 1 + 0, meaning zero extra wait states. So, latency 1 + 1 implies one stall cycle; latency 1 + N has N extra stall cycles.



			Latencies beyond single cycle	
Loop:	LD	F2,0(RX)	Memory LD	+4
I0:	DIVD	F8,F2,F0	Memory SD	+1
I1:	MULTD	F2,F6,F2	Integer ADD, SUB	+0
I2:	LD	F4,0(Ry)	Branches	+1
I3:	ADDD	F4,F0,F4	ADDD	+1
I4:	ADDD	F10,F8,F2	MULTD	+5
I5:	ADDI	Rx,Rx,#8	DIVD	+12
I6:	ADDI	Ry,Ry,#8		
I7:	SD	F4,0(Ry)		
I8:	SUB	R20,R4,Rx		
I9:	BNZ	R20,Loop		

Figure 1.

3. Build an interesting circuit: extend your HW1 CPU, build a superscalar dependency detection unit, etc.

Computer Architecture Project

SET -4

Use any language or simulator to solve the below problems and give a detail report for the same:

1. Let's consider what dynamic scheduling might achieve here. Assume a microarchitecture as shown in Figure 1. Assume that the arithmetic-logical units (ALUs) can do all arithmetic ops (MULTD, DIVD, ADDD, ADDI, SUB) and branches, and that the Reservation Station (RS) can dispatch at most one operation to each functional unit per cycle (one op to each ALU plus

one memory op to the LD/ST).

   a. Suppose all of the instructions from the sequence in Figure 1 are present in the RS, with no renaming having been done. Highlight any instructions in the code where register renaming would improve performance. (*Hint:* Look for read-after-write and write-after-write hazards. Assume the same functional unit latencies as in Figure 2.)
   b. Suppose the register-renamed version of the code from part (a) is resident in the RS in clock cycle *N*, with latencies as given in Figure 2. Show how the RS should dispatch these instructions out of order, clock by clock, to obtain optimal performance on this code. (Assume the same RS restrictions as in part (a). Also assume that results must be written into the RS before they're available for use—no bypassing.) How many clock cycles does the code sequence take?
   c. If you wanted to improve the results of part (c), which would have helped most: (1) Another ALU? (2) Another LD/ST unit? (3) Full bypassing of ALU results to subsequent operations? or (4) Cutting the longest latency in half? What's the speedup?

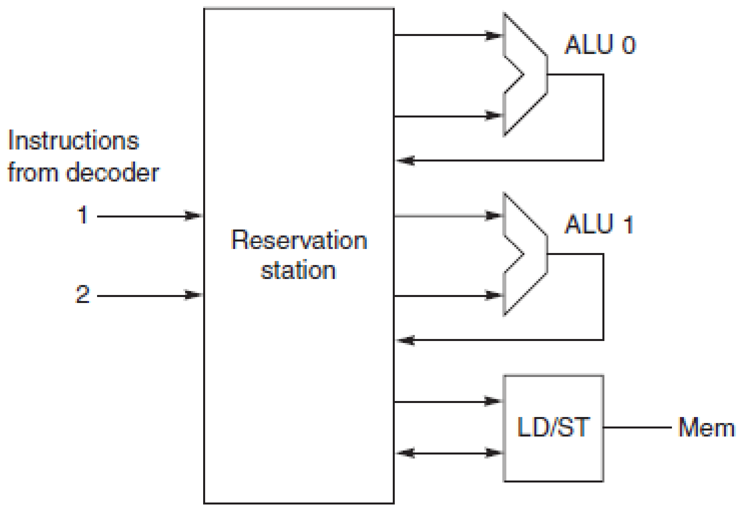


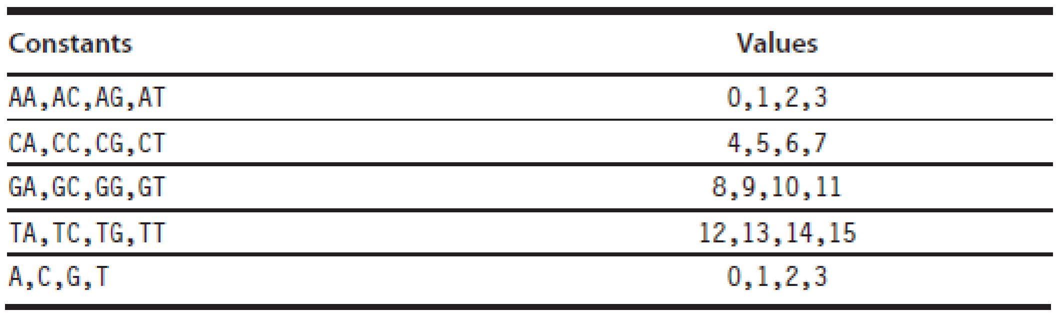
Figure 1.

2. Assume the constants shown in Figure 2 Show the code for MIPS and VMIPS. Assume we cannot use scatter-gather loads or stores. Assume the starting addresses of tiPL, tiPR, clL, clR, and clP are in RtiPL, RtiPR, RclL, RclR, and RclP, respectively. Assume the VMIPS register

length is user programmable and can be assigned by setting the special register VL (e.g., li VL 4). To facilitate vector addition reductions, assume that we add the following instructions to VMIPS: SUMR.S Fd, Vs Vector Summation Reduction Single Precision: This instruction performs a summation reduction on a vector register Vs, writing to the sum into scalar register Fd.

a. Assuming seq_length == 500, what is the dynamic instruction count for both implementations?

b. Assume that the vector reduction instruction is executed on the vector functional unit, similar to a vector add instruction. Show how the code sequence lays out in convoys assuming a single instance of each vector functional unit. How many chimes will the code require? How many cycles per FLOP are needed, ignoring vector instruction issue overhead?



Constants	Values
AA,AC,AG,AT	0,1,2,3
CA,CC,CG,CT	4,5,6,7
GA,GC,GG,GT	8,9,10,11
TA,TC,TG,TT	12,13,14,15
A,C,G,T	0,1,2,3

Figure 2.

3. Implement and compare victim caches and skewed-associative caches.

Computer Architecture Project

SET -5

Use any language or simulator to solve the below problems and give a detail report for the same:

1. With CUDA we can use coarse-grain parallelism at the block level to compute the conditional likelihoods of multiple nodes in parallel. Assume that we want to compute the conditional likelihoods from the bottom of the tree up. Assume that the conditional likelihood and transition probability arrays are organized in memory as described in question 4 and the group of tables for each of the 12 leaf nodes is also stored in consecutive memory locations in the order of node number. Assume that we want to compute the conditional likelihood for nodes 12 to 17, as shown in Figure 1. Change the method by which you compute the array indices in your answer include the block number.
a. Convert your code from above exercise into PTX code. How many instructions are needed for the kernel?
b. How well do you expect this code to perform on a GPU? illustrate your answer in simulation output.

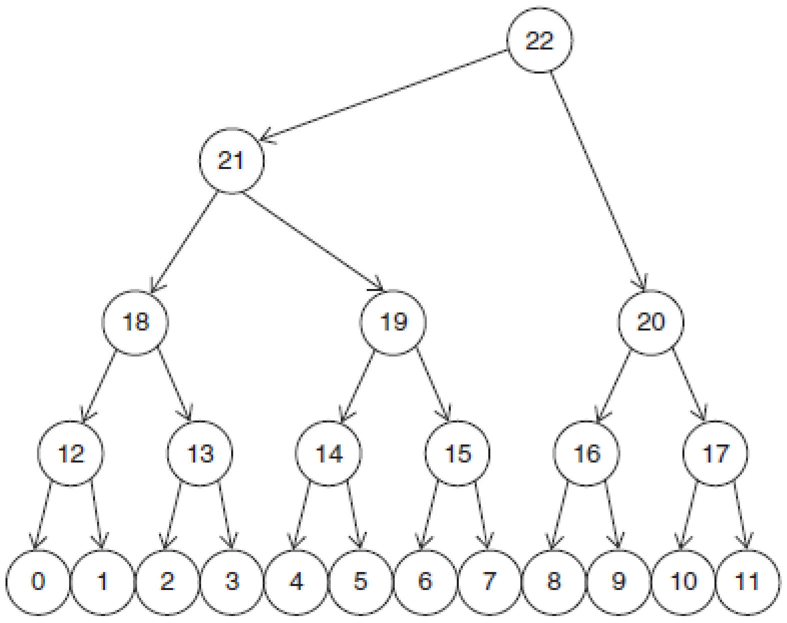


Figure 1

2. Assume you are designing a hardware prefetcher for the unblocked matrix transposition code above. The simplest type of hardware prefetcher only prefetches sequential cache blocks after a miss. More complicated "non-unit stride" hardware prefetchers can analyze a miss reference stream and detect and prefetch non-unit strides. In contrast, software prefetching can determine

non-unit strides as easily as it can determine unit strides. Assume prefetches write directly into the cache and that there is no “pollution” (overwriting data that must be used before the data that are prefetched). For best performance given a non-unit stride prefetcher, in the steady state of the inner loop how many prefetches must be outstanding at a given time?

3. Implement Branch prediction methods and analyze the performance