Name _____

Computer Architecture and Implementation
EE 7700-4
Practice Final Examination

December 1998, 0:00-23:59 CST

Problem 1	_____	(20 pts)
Problem 2	_____	(20 pts)
Problem 3	_____	(20 pts)
Problem 4	_____	(40 pts)
Exam Total	_____	(100 pts)

Alias _____

Though the final exam will be cumulative this practice final only contains problems from the second half of the course to provide a better selection of problems on the new material.

Problem 1: The program below transposes a matrix.

```
for(row=0; row<nrows; row++)
 for(col=0; col<ncols; col++)
   b[ col + row * ncols ] = a[ row + col * nrows ];
```

(a) Rewrite the program in DLXV assembler making effective use of the vector instructions. The arrangement of the arrays should be the same in both programs. (20 pts)

```
! Initially:
! r1: Address of first element of a (to be read).
! r2: Address of first element of b (to be written).
! r3: Number of rows.
! r4: Number of columns.
! Start solution here:
```

Problem 2: Add support for full/empty bits to the directory cache coherence protocol described in the text. An FE bit is associated with each block. The system should support the following instructions:

`lwFE r1,0(r2)`, loads r1 with the contents of address 0(r2) when its FE bit is full, sets the bit to empty.

`swFE 0(r2),r1`, stores the contents of r1 at address 0(r2) when its FE bit is empty, sets the bit to full.

These instructions should be implemented so, if necessary, they wait for the FE bit to change to the state needed. Try to minimize the number of messages sent under the protocol. The protocol should work when there are multiple `lwFE` and `swFE` simultaneously accessing the same location. (The instructions would have to be completed in some order.)

Specify additional protocol messages and states and other storage added to the cache and at the memory. Show state transition diagrams with the new states. Only show new transitions and any existing transitions needed to describe the protocol. (20 pts)

Problem 3: Several DLX assembly language code fragments appear below. The comments indicate initial values in memory locations and the values retrieved by load instructions. The code fragments do not necessarily start at the same time; ellipses (. . .) indicate an arbitrary time delay. Memory locations accessed by the code below are not accessed by other code or other processors.

Indicate which code fragments could not have been executed on a coherent memory system (based on the comments) and explain why they are not coherent. (10 pts)

Indicate which code fragments could not have been executed on a sequentially consistent system (based on the comments) and explain why not. (10 pts)

```
! Fragment 1
! Contents of r11 same on all processors.
! Contents of r12 same on all processors, r11 != r12.
! Proc. 0
  lw r1, 0(r11)  ! r1 loaded with 1.
  lw r2, 0(r12)  ! r2 loaded with 12.

! Proc. 1
  lw r1, 0(r11)  ! r1 loaded with 11.
  lw r2, 0(r12)  ! r2 loaded with 2.

! Proc. 2 (Order of stores switched from earlier version.)
  addi r2, r0, #2
  sw o(r12), r2
  addi r1, r0, #1
  sw o(r11), r1


! Proc. 3  (Order of stores switched from earlier version.)
  addi r2, r0, #12
  sw o(r12), r2
  addi r1, r0, #11
  sw o(r11), r1
```

```
! Fragment 2
! Contents of r11 same on all processors.
! Before code below runs, 0 at address in r11.
! Proc 0.
 lw r1, 0(r11)  ! r1 loaded with 0.
 ...
 lw r2, 0(r11)  ! r2 loaded with 0.
 ...
 lw r3, 0(r11)  ! r3 loaded with 2.

! Proc 1.
 lw r1, 0(r11)  ! r1 loaded with 0.
 ...
 lw r2, 0(r11)  ! r2 loaded with 1.
 ...
 lw r3, 0(r11)  ! r3 loaded with 3.

! Proc 2.
 addi r1, r0, #1
 sw 0(r11), r1
 ...
 addi r1, r0, #2
 sw 0(r11), r1
 ...
 addi r1, r0, #3
 sw 0(r11), r1
 ...
```

```
! Fragment 3
! Contents of r11 same on all processors.
! Before code below runs, 9 at address in r11.
!
! Proc 0.
 addi r1, r0, #0
 sw 0(r11), r1
 ...
 lw r1, 0(r11)  ! r1 loaded with 0.

! Proc 1.
 addi r1, r0, #1
 sw 0(r11), r1
 ...
 lw r1, 0(r11)  ! r1 loaded with 2.

! Proc 2.
 addi r1, r0, #2.
 sw 0(r11), r1
 ...
 lw r1, 0(r11)  ! r1 loaded with 0.
```

```
! Fragment 4.
! Contents of r11 same on all processors.
! Before code below runs, 0 at address in r11.
!
! Proc 0
 addi r1, r0, #1
 sw 0(r11), r1
 ...
 addi r1, r0, #2
 sw 0(r11), r1

! Proc 1
 addi r1, r0, #3
 sw 0(r11), r1
 ...
 lw r2, 0(r11) ! r2 loaded with 2.
 ...
 addi r1, r0, #4
 sw 0(r11), r1

! Proc 2
 lw r1, 0(r11)  ! r1 loaded with 3
 ...
 lw r1, 0(r11)  ! r1 loaded with 2

! Proc 3
 lw r1, 0(r11)  ! r1 loaded with 4
 ...
 lw r1, 0(r11)  ! r1 loaded with 1
```

Problem 4: Answer each question below.

(a) An ordinary superscalar processor is to be converted to an SMT processor. Provide an argument against increasing the number of functional units. (5 pts)

(b) Why might a multithreaded program run faster (higher IPC) than a multiprocessing workload on a simultaneous multithreading processor? (5 pts)

(c) Why wouldn't an SMT version of a superscalar processor *not* improve the performance of code that contained frequent taken branches and jumps? Assume that branches are perfectly predicted. (5 pts)

(*d*) Contrast the relative performance of SMT and Tera on code that contains frequent cache misses (on the SMT). The programs contain as many threads as the respective processor can support. Assume the peak IPC of the two machines are identical.

How would performance compare if the program running on Tera's machine had the same number of threads as the SMT processor? (5 pts)

(*e*) Describe what the lookahead field in Tera instructions specifies and how the instruction pipeline would have to be changed if it were not present. (5 pts)

(*f*) A memory system works in the following way. Load and store instructions are placed in a load/store buffer in program order. When they are placed in the buffer the cache is checked; if the address is not present or in the proper state the line is fetched. When an instruction reaches the head of the load/store buffer the cache is checked again, if the address is not present (because of eviction or invalidation) it is fetched again, and when it arrives the access completes and is removed from the buffer. The caches are coherent and exclusive access is not granted until all invalidates are acknowledged. Which memory model does this implement? Justify your answer. (5 pts)

(g) Explain why it might not be a good idea to port programs that have many difficult to predict branches (on a conventional ISA) to a vector machine. (5 pts)

(h) A vector processor has a word size of eight bytes and memory banks in which words can be read every 14 cycles. The bus can sustain a transfer rate of 1 word per cycle.

What is the minimum number of banks needed to read a vector of length 256 and stride 32 bytes at a rate of 1 word per cycle?

What is the minimum number of banks needed to read a vector of length 256 and stride 112 bytes at a rate of 1 word per cycle?? (FYI, $112 = 7 \times 2^3$) (5 pts)