Name Partial Solution

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 Partial Solution

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)

```
! Solution
! 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.
 movi2s vlr, r4
 add r5, r0, r3
LOOP:
 lvws v1, (r1,r3)
 sv r2, v1
 add r2, r2, r4
 addi r1, r1, #1
 subi r5, r5, #1
 bnez r5, LOOP
```

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
```

Coherent. (Processor 0 and 1 load each address once, so there is no way for a processor to observe the order of stores to a particular address. Coherence does not specify any particular ordering of loads of different addresses by same processor.) Not sequentially consistent because first load at processor 1 must come after second store at processor 3 but second load at processor 1 must come before first store at processor 3.

```
! 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
 ...
```

Coherent and sequentially consistent.

```
! 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.
```

Coherent and sequentially consistent.

```
! 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
```

Not coherent because 4 cannot be written after 2 but before 1. Not sequentially consistent since not coherent.

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)

Increasing the number of functional units would increase the cost. The point of SMT is making more efficient use of resources that are already present.

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

Fewer cache misses since threads may share data.

(*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)

If the fetch unit could only handle consecutive instructions the frequent control transfers would result in a small number of instructions fetched per cycle. That would limit performance on a superscalar and an SMT processor.

(*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)

On code having frequent cache misses an SMT processor would quickly run out of threads to run while the Tera would always have enough, and so the Tera would be much faster.

If the number of threads were the same the SMT might run faster since it could execute instructions out of order, hiding some miss latency. The SMT would also run faster because of cache hits.

(*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)

It specifies the number of instructions that can be executed before the result is needed. If it were not present interlock hardware which stalls the pipeline when certain dependencies are present would have to be included.

(*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)

Sequential consistency. The operations complete in program order. Fetching a block early does not change its value, and if its value is changed at another processor between the time it is first fetched and the time the cache is checked a second time it will be invalidated.

(*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)

Vector instructions perform the same operation on some (using a mask) or all of a vector. If a program has many difficult to predict branches there may be few sets of data to which the same operation would be applied and even if a mask could be used to apply an operation to a subset of elements, the mask may be useful only once (so the time needed to create the mask would be greater than the time saved using it).

(*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)

Number of banks used is $\frac{\text{lcm}(B,S)}{S}$, where B is the number of banks present and S is the stride (in words). To sustain an access rate of one word per cycle the number of banks used must be at least the memory cycle time, 14 cycles. The minimum number of banks is found by finding the minimum B such that $\text{lcm}(B,S) = St_{\text{cycle}}$, where t_{cycle} is the memory cycle time. (Function $\text{lcm}(B,S)$ is the least common multiple, the smallest integer that is a multiple of both B and S. For example, $\text{lcm}(4,2) = 4$, $\text{lcm}(4,3) = 12$, $\text{lcm}(6,15) = \text{lcm}(2 \times 3, 3 \times 5) = 2 \times 3 \times 5 = 30$.)

Length 256, stride 32 bytes. Stride 32 bytes = 4 elements.

$$\min_B \left(t_{\text{cycle}} = \frac{\text{lcm}(B,S)}{S} \right)$$

$$\Rightarrow \text{lcm}(B,S) = St_{\text{cycle}}$$
$$\text{lcm}(B, 2 \times 2) = 4 \times 14 = 2 \times 2 \times 2 \times 7$$
$$\Rightarrow B = 2 \times 2 \times 2 \times 7 = 56 \quad \text{banks}$$

Length 256, stride 112 bytes. Stride 112 bytes = 7 elements:

$$\min_B \left(t_{\text{cycle}} = \frac{\text{lcm}(B,S)}{S} \right)$$

$$\Rightarrow \text{lcm}(B,S) = St_{\text{cycle}}$$
$$\text{lcm}(B,7) = 7 \times 14 = 2 \times 7 \times 7$$
$$\Rightarrow B = 2 \times 7 \times 7 = 98 \quad \text{banks}$$