Computer System Architecture
6.823 Final Examination
Spring 2002

Name:_____

This is an open book, open notes exam.
180 Minutes
22 Pages

Notes:

- Not all questions are of equal difficulty, so look over the entire exam and budget your time carefully.
- Please carefully state any assumptions you make.
- Please write your name on every page in the final (you get 6 points for doing this).
- For partial credit, be sure to show your work.

			Name: _____	6 Points
Part A:	(Question 1	- Question 4)	_____	20 Points
Part B:	(Question 5	- Question 7)	_____	16 Points
Part C:	(Question 8	- Question 12)	_____	24 Points
Part D:	(Question 13	- Question 20)	_____	24 Points
Part E:	(Question 21	- Question 24)	_____	31 Points
Part F:	(Question 25	- Question 27)	_____	16 Points
Part G:	(Question 28	- Question 30)	_____	12 Points
Part H:	(Question 31	- Question 37)	_____	31 Points
			Total: _____	**180 Points**

Name_____

Part A: Complex Pipelining (20 points)

Consider an out-of-order superscalar DLX processor that uses a register-renaming scheme with a single unified physical register file (Lecture 14). This machine has 128 physical registers. The DLX ISA has 32 general-purpose registers and 32 floating-point registers.

Question 1 (4 points)

What is the maximum number of registers on the free list? Describe a situation in which the maximum number of registers is on the free list.

Question 2 (4 points)

What is the minimum number of registers on the free list? Describe a situation in which the minimum number of registers is on the free list.

Question 3 (6 points)

Which of the following conditions needs to be checked for an instruction before performing register renaming and placing the renamed instruction into the reorder buffer? Circle all that could affect whether an instruction is placed into the reorder buffer.

A. if the functional unit needed by the instruction is busy

B. if there are any physical registers on the free list

C. if the instruction will cause an exception

D. if the instruction causes a WAR hazard with any earlier instruction in the ROB

E. if the instruction causes a WAW hazard with any earlier instruction in the ROB

F. if there are more rename table snapshots available

Name_____

Question 4 (6 points)

Suppose we added the instruction `MOVZ` to the DLX ISA.

The semantics of `MOVZ` are:

```
MOVZ Rd,Rs1,Rs2      ; if (Rs2 == 0) then Rd <- Rs1
```

If the value in `Rs2` is equal to zero, then the contents of `Rs1` are placed into `Rd`. `MOVZ` is an R-type instruction.

What difficulties might arise in implementing `MOVZ` in the out-of-order superscalar described at the beginning of Part A?

Name_____

Part B: Caches (16 points)

Consider a physically-indexed, virtually-tagged, set-associative write-back cache. Bits from the translated physical address are used to index the cache but bits from the untranslated virtual address are used for the tag.

The virtual page size is 2^k bytes. The cache has 2^L sets and 2^w ways. Each cache line is 2^b bytes. The amount of virtually addressable memory is 2^{32} bytes and the amount of physically addressable memory is also 2^{32} bytes.

Question 5 (6 points)

Can an aliasing problem occur? Explain.

Question 6 (6 points)

Write an expression for the number of bits needed for the tag of a cache line in terms of k, L, b, and w.

Question 7 (4 points)

What is the primary reason why it does not make sense to use a physically-indexed, virtually-tagged cache?

Name_____

Part C: Branch Prediction (24 points)

Inspired by the lectures on deep pipelining, Ben Bitdiddle decides to build an in-order, single-issue, deeply-pipelined DLX with no delay slots. His design has the following pipeline:

IF1	IF2	ID1	ID2	EX1	EX2	MA1	MA2	WB

The original instruction fetch (IF), instruction decode (ID), execute (EX), and memory (MA) stages have each been divided into two stages. Instruction fetch uses a simple, predict-not-taken strategy, and fetches instructions sequentially from the instruction memory unless redirected by a resolved taken branch. The register file is now read at the beginning of the first execute stage (EX1).

Question 8 (3 points)

What is the **minimum** penalty, in cycles, for a taken conditional branch (`BEQZ` or `BNEZ`) in this pipeline? Clearly state in which stage the branch is resolved.

Question 9 (8 points)

To improve the performance of his processor, Ben decides to add dynamic branch prediction. He adds a BHT to the first decode stage (ID1). Instructions are fetched sequentially from the instruction memory unless redirected by a predicted or resolved branch.

In the following table, enter the **minimum** conditional branch penalty, in cycles, for each case. Clearly state any assumptions on which your answer depends.

Predicted Branch Direction	**Actual Branch Direction**	
	Taken	Not Taken
Taken		
Not Taken		

Question 10 (3 points)

What is the **minimum** penalty, in cycles, for an indirect jump (`JR`) using this scheme? Clearly state any assumptions on which your answer depends.

Question 11 (4 points)

Louis Reasoner thinks that Ben can decrease the branch penalty by moving the BHT into the first fetch stage (IF1). What's wrong with this idea?

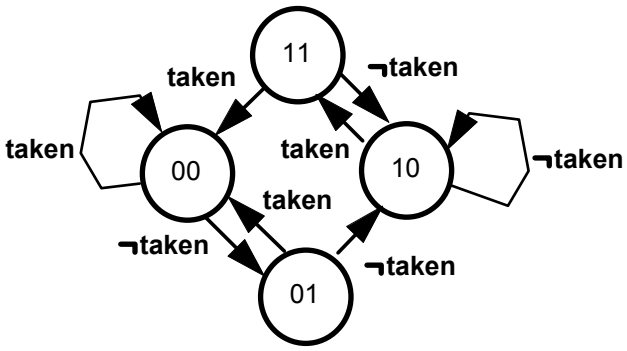
Name_____

Question 12 (6 points)

To evaluate the benefits of adding a BHT to the ID1 stage, Ben uses the following benchmark program (from PS4). The input to the program is an array of alternating 0's and 1's ("0101010…").

```
; The initial contents of R3 is a pointer to
; the beginning of an array of 32-bit integers.
; The initial contents of R1 is the length of
; the array, which is > 0.
; The initial contents of R2 is 0.
; (R2 holds the result of the program.)
loop:
       LW    R4,0(R3)
       ADDI  R3,R3,#4
       SUBI  R1,R1,#1
b1:
       BEQZ  R4,b2
       ADDI  R2,R2,#1
b2:
       BNEZ  R1,loop
```

He is using the same 2-bit predictor as in PS4 (L13-11), where states 0X predict taken, and states 1X predict not taken:



Unlike the BHT in PS4, however, Ben's BHT has only one entry, which is updated when a branch is resolved. What is the average steady-state prediction accuracy of the BHT for the above benchmark, given that the BHT entry has initial value "00"? Explain.

Name_____

Part D: Vector Computers (24 points)

Each of the following loops is to be translated to run on a vector machine with vector registers. In each case, give the **maximum** vector length that can be used when vectorizing the code and briefly explain what features of the code limit vector length.

Assume the vector machine has infinite length vector registers and that arrays with different names are stored in non-overlapping regions of memory. All operands are integers.

Question 13 (3 points)

```
for (i=0; i<N; i++)
    C[i] = A[i] + B[i];
```

Question 14 (3 points)

```
for (i=0; i<N; i++)
    A[i] = A[i] + A[i+1];
```

Question 15 (3 points)

```
for (i=1; i<N+1; i++)
    A[i] = A[i] + A[i-1];
```

Question 16 (3 points)

```
for (i=13; i<N+13; i++)
    A[i] = A[i] + A[i-13];
```

Name_____

Question 17 (3 points)

```
for (i=0; i<N; i++)
    A[i] = A[i] + B[C[i]];
```

Question 18 (3 points)

```
for (i=0; i<N; i++)
    A[i] = A[C[i]] + B[i];
```

Question 19 (3 points)

```
for (i=0; i<N; i++)
    A = A + B[i];   // A is a scalar variable
```

Question 20 (3 points)

```
for (j=0; j<N; j++)
    for (i=0; i<M; i++)
        A[i][j] = A[i][j] + B[i][j];
```

Part E: Cache Coherence Update Protocols (31 points)

In PS6-3, we examined a cache-coherent distributed shared memory system. Ben wants to convert the directory-based invalidate cache coherence protocol from the problem set into an update protocol. He proposes the following scheme.

Caches are write-through, no write allocate. When a processor wants to write to a memory location, it sends a store-request to the home site, along with the data word that it wants written. The home site updates memory, and sends an update-request with the new data to each of the sites caching the block, unless that site is the processor performing the store, in which case it sends a store-reply containing the new data.

If the processor performing the store is caching the block being written, it must wait for the reply from the home site to arrive before storing the new value into its cache. If the processor performing the store is not caching the block being written, it can proceed after issuing the store-request.

Note that store-request now has a meaning and usage different from the protocol in PS6. Also note that store-requests and update-requests contain data at the word-granularity, and not at the block-granularity.

In the proposed scheme, memory will always have the most up-to-date data, and the states C-modified, H-modified, and H-transient are no longer used.

As in PS6, the interconnection network guarantees that message-passing is reliable, and free from deadlock, livelock, and starvation. Also as in PS6, message-passing is FIFO.

Each home site keeps a FIFO queue of incoming requests, and processes these in the order received.

Question 21 (5 points)

Alyssa claims that Ben's protocol does not preserve sequential consistency because it allows two processors to observe stores in different orders. Describe a scenario in which this problem can occur.

Name_____

Question 22 (16 points)

Noting that many commercial systems do not guarantee sequential consistency, Ben decides to implement his protocol anyway. Fill in the following state transition tables for the proposed scheme. We use *k* to represent the site that issued the received message.

No.	Current State	Event Received	Next State	Action
1	C-invalid	Load	C-transient	load-request → home
2	C-invalid	Store		
3	C-invalid	update-request		
4	C-shared	Load	C-shared	processor reads cache
5	C-shared	Store		
6	C-shared	Replace	C-invalid	nothing
7	C-shared	update-request		
8	C-transient	load-reply	C-shared	data → cache, processor reads cache
9	C-transient	store-reply		
10	C-transient	update-request		

Table 1: Cache State Transitions

No.	Current State	Message Received	Next State	Action
1	H-uncached	load-request	H-shared[$\{k\}$]	load-reply → k
2	H-uncached	store-request		
3	H-shared[S]	load-request	H-shared[$S \cup \{k\}$]	load-reply → k
4	H-shared[S]	store-request		

Table 2: Home Directory State Transitions

Question 23 (5 points)

After running a system with this protocol for a long time, Ben finds that the network is flooded with update-requests. Alyssa says this is a bug in his protocol. What is the problem and how can you fix it?

Question 24 (5 points)

As in PS6, FIFO message passing is a necessary assumption for the correctness of the protocol. If the network were non-FIFO, it becomes possible for a processor to never see the result of another processor's store. Describe a scenario in which this problem can occur.

Name_____

Part F: Synchronization I (16 points)

In this part, we consider Dekker's protocol for mutual exclusion discussed in L17-18. The protocol is based on three shared variables `c1`, `c2`, and `turn`. Initially, both `c1` and `c2` are `0`. The following pseudo-code implements the protocol. The critical section reads the next data item to be processed and updates the pointer.

Processor 1	Processor 2
`c1 = 1;`	`c2 = 1;`
`turn = 1;`	`turn = 2;`
`while (c2 == 1 && turn == 1) {}`	`while (c1 == 1 && turn == 2) {}`
`/* start of critical section */`	`/* start of critical section */`
`data = *ptr;`	`data = *ptr;`
`ptr = ptr + 1;`	`ptr = ptr + 1;`
`/* end of critical section */`	`/* end of critical section */`
`c1 = 0;`	`c2 = 0;`

The protocol for Processor 1 is implemented in DLX below. Assume DLX has no delay slots.

```
       ; R1, R2: pointer to 'c1' and 'c2' respectively
       ; R3: pointer to 'turn'
       ; R4: pointer to 'ptr'
       ; R5: data
       ;
       ADDI       R6, R0, #1
       SW         0(R1), R6          ; c1 = 1
       SW         0(R3), R6          ; turn = 1
Loop:  LW         R6, 0(R2)          ; load 'c2'
       LW         R7, 0(R3)          ; load 'turn'
       SEQI       R6, R6, #1         ; set R6 if c2 == 1
       SEQI       R7, R7, #1         ; set R7 if turn == 1
       AND        R6, R6, R7
       BNEZ       R6, Loop           ; while (c2==1 && turn==1)

       LW         R6, 0(R4)          ; load 'ptr'
       LW         R5, 0(R6)          ; data = *ptr
       ADDI       R6, R6, #4         ; ptr = ptr + 1
       SW         0(R4), R6          ; store 'ptr'

       SW         0(R1), R0          ; c1 = 0;
```

Question 25 (2 points)

Dekker's protocol assumes the system is sequentially consistent. Unfortunately, many modern computer systems have relaxed memory models. Consider a system that provides partial store ordering (PSO), where a read or a write may complete before an earlier write if they are to different addresses. What problems occur if Dekker's protocol is run on a system with two processors using PSO? Circle all possible problems from the following list.

A. None

B. Both processors can enter the critical section at the same time

C. No processor can enter the critical section (deadlock or livelock)

D. One of the processors cannot enter the critical section (starvation)

Question 26 (8 points)

Briefly explain how the above problem(s) can occur.

Name_____

Question 27 (6 points)

Our PSO system has `MEMBAR` instructions to enforce proper memory access ordering when necessary. The `MEMBAR` instruction ensures that all memory operations preceding the `MEMBAR` in program order are globally visible before any memory operations following the `MEMBAR`. Insert the **minimum** number of `MEMBAR` instructions in the following listing to make Dekker's protocol work properly on the PSO system:

```
        ADDI    R6, R0, #1

        SW      0(R1), R6       ; c1 = 1

        SW      0(R3), R6       ; turn = 1

Loop:   LW      R6, 0(R2)       ; load 'c2'

        LW      R7, 0(R3)       ; load 'turn'

        SEQI    R6, R6, #1      ; set R6 if c2 == 1

        SEQI    R7, R7, #1      ; set R7 if turn == 1

        AND     R6, R6, R7

        BNEZ    R6, Loop        ; while (c2==1 && turn==1)

        LW      R6, 0(R4)       ; load 'ptr'

        LW      R5, 0(R6)       ; data = *ptr

        ADDI    R6, R6, #4      ; ptr = ptr + 1

        SW      0(R4), R6       ; store 'ptr'

        SW      0(R1), R0       ; c1 = 0;
```

Name_____

Part G: Synchronization II (12 points)

Below is Lamport's Bakery algorithm for mutual exclusion for N processes (L17-18).

```
// initially num[j] = 0, for all j
// i is the current process
choosing[i] = 1;
num[i] = max(num[0],..,num[N-1]) + 1;
choosing[i] = 0;
for (j = 0; j < N; j++) {
    while(choosing[j]) {}
    while(num[j] &&
          ((num[j] < num[i]) ||
           ((num[j] == num[i]) && (j < i)))) {}
}
<critical section>
num[i] = 0;
```

Question 28 (4 points)

If multiple processes are simultaneously trying to enter the critical section, which process gets to enter the critical section first? Assume a sequentially consistent memory model.

Name_____

Question 29 (2 points)

Below is Lamport's Bakery algorithm with the `choosing` variable omitted.

```
1: num[i] = max(num[0],..,num[N-1]) + 1;
2: for (j = 0; j < N; j++) {
3:     while(num[j] &&
4:           ((num[j] < num[i]) ||
5:            ((num[j] == num[i]) && (j < i)))) {}
5: }
6: <critical section>
7: num[i] = 0;
```

What problem(s) can occur if the `choosing` variable is omitted?

A. Deadlock

B. Livelock

C. One process cannot enter the critical section (starvation)

D. More than one process can enter the critical section

Question 30 (6 points)

Describe a scenario where a problem occurs. (An example scenario: If process1 executes line1 and then process2 executes line1, deadlock occurs.)

Name_____

Part H: Multithreading (31 points)

This part evaluates the effectiveness of multithreading using a simple database benchmark. The benchmark searches for an entry in a linked list built from the following structure, which contains a key, a pointer to the next node in the linked list, and a pointer to the data entry.

```
struct node {
      int key;
      struct node *next;
      struct data *ptr;
}
```

The following DLX code shows the core of the benchmark, which traverses the linked list and finds an entry with a particular key. Assume DLX has no delay slots.

```
       ;
       ; R1: a pointer to the linked list
       ; R2: the key to find
       ;
Loop:  LW        R3, 0(R1)   ; load a key
       LW        R4, 4(R1)   ; load the next pointer
       SEQ       R3, R3, R2  ; set R3 if R3 == R2
       BNEZ      R3, End     ; found the entry
       ADD       R1, R0, R4
       BNEZ      R1, Loop    ; check the next node
End:
       ; R1 contains a pointer to the matching entry or zero if
       ; not found
```

We run this benchmark on a single-issue in-order processor. The processor can fetch and issue (dispatch) one instruction per cycle. If an instruction cannot be issued due to a data dependency, the processor stalls. Integer instructions take one cycle to execute and the result can be used in the next cycle. For example, if `SEQ` is executed in cycle 1, `BNEZ` can be executed in cycle 2. We also assume that the processor has a perfect branch predictor with no penalty for both taken and not-taken branches.

Question 31 (4 points)

Assume that our system does not have a cache. Each memory operation directly accesses main memory and takes 100 CPU cycles. The load/store unit is fully pipelined, and non-blocking. After the processor issues a memory operation, it can continue executing instructions until it reaches an instruction that is dependent on an outstanding memory operation. How many cycles does it take to execute one iteration of the loop in steady state?

Question 32 (4 points)

Now we add zero-overhead multithreading to our pipeline. A processor executes multiple threads, each of which performs an independent search. Hardware mechanisms schedule a thread to execute each cycle.

In our first implementation, the processor switches to a different thread every cycle using fixed round robin scheduling (similar to CDC 6600 PPUs, L23-7). Each of the N threads executes one instruction every N cycles. What is the **minimum** number of threads that we need to fully utilize the processor, i.e., execute one instruction per cycle?

Name_____

Question 33 (4 points)

How does multithreading affect throughput (number of keys the processor can find within a given time) and latency (time processor takes to find an entry with a specific key)? Assume the processor switches to a different thread every cycle and is fully utilized. Check the correct boxes.

	Throughput	**Latency**
Better		
Same		
Worse		

Question 34 (5 points)

We change the processor to only switch to a different thread when an instruction cannot execute due to data dependency. What is the minimum number of threads to fully utilize the processor now? Note that the processor issues instructions in-order in each thread.

Question 35 (5 points)

We now investigate how caches can be used to improve performance for long memory access times. We add one level of data cache to the original processor from Question 31. A memory access takes one cycle if it hits in the cache. Otherwise, a memory access takes 100 cycles (including checking the cache).

The cache has 8-word blocks and the `'node'` structure is always aligned on 4 word boundaries. This ensures that the first and the second load instructions always access the same cache line due to spatial locality.

What is the **average** number of cycles that the processor with a data cache takes to execute one iteration of the loop in steady state? Assume that the hit-rate of the first load instruction is 50%. Remember there is no multithreading in this version of the machine.

Question 36 (4 points)

How does caching affect the average throughput and average latency of our benchmark as compared to the original processor from Question 31? Check the correct boxes.

	Throughput	**Latency**
Better		
Same		
Worse		

Question 37 (5 points)

Now consider combining multithreading with the data cache. If the cache hit-rate of the first load is 50%, what is the minimum number of threads to **guarantee** full utilization of the processor? In this question, assume that the processor switches on a cache miss.