Computer System Architecture
6.823 Midterm Examination
Spring 2001

Name:____________________

This is an open book, open notes exam.
110 Minutes
16 Pages

Notes:
- Not all questions are of equal difficulty, so look over the entire exam and budget your time carefully.
- The last 2 pages of the exam are for your reference only. Feel free to tear them off to look at. We will not grade anything you write on these pages.
- Please write your name on every page in the midterm (you get 5 points for doing this).

	Name:	_________	5 Points
Part A	(Q1-Q4):	________	26 Points
Part B	(Q5-Q10):	________	31 Points
Part C	(Q11-Q12):	________	12 Points
Part D	(Q13-Q19):	________	27 Points
Part E	(Q20):	________	9 Points

Total: ________ 110 Points

Name________________________________

Part A: Microprogramming (26 points)

Ben Bitdiddle decided to add a separate PC register and a PC+4 adder to the original bus-based DLX implementation (Lecture slide 4-2) to improve performance. The modified bus-based DLX is shown in Figure 1. Assume that memory can perform a read or a write in one cycle.

Figure 1: Modified bus-based DLX machine

The original microcode for Fetch, BNEZ, ADD, JR, JAL, and JALR are as follows: (Note that the errors in lecture slide 4-25 are fixed here and that some further optimizations have been added)

Fetch0:	MA ← PC, A ← PC	ADD0:	A ← Reg[rf1]
Fetch1:	IR ← Mem	ADD1:	B ← Reg[rf2]
Fetch2:	PC ← A + 4; dispatch	ADD2:	Reg[rf3] ← A + B; fetch
BNEZ0:	A ← Reg[rf1]	JR0:	A ← Reg[rf1]
BNEZ1:	B ← sExt$_{16}$(Imm); feqz	JR1:	PC ← A; fetch
BNEZ2:	A ← PC		
BNEZ3:	PC ← A + B; fetch		

JAL0:	A ← PC	JALR0:	A ← PC
JAL1:	Reg[31] ← A	JALR1:	Reg[31] ← A
JAL2:	B ← sExt$_{26}$(Imm)	JALR2:	PC ← Reg[rf1] ; fetch
JAL3:	PC ← A + B; fetch		

Question 1 (12 points)

For each of the above microcode sequences, state whether it can be accelerated by making use of the separate PC register and PC+4 adder, and if so, give the new microcode sequence.

Ben is given the following pseudo-code:

```
if A is 0, B ← C
```

Ben converts this pseudo-code to the DLX code shown below. Assume that Ra, Rb, and Rc initially hold A, B, and C values respectively and that there are no delay slots.

```
    BNEZ Ra, done
    ADD  Rb, Rc, R0
done:
```

Question 2 (3 points)

How many cycles does this code sequence take on average for the original and modified bus-based DLX machines? Assume the probability that A = 0 is 0.5.

Alyssa P. Hacker observes that the pseudo-code that Ben was given is frequently used, and decides to implement a new instruction called CMOV (conditional move). This new instruction has the following form:

```
CMOV R1, R2, R3
; R1 gets the value of R3 if R2 has zero.
```

(Take a careful look at register numbers!)

Question 3 (8 points)

Write the microcode for this new instruction.

Question 4 (3 points)

Using the CMOV instruction, write the DLX code for the pseudo code. Now how many cycles does this DLX code take to execute on average? Again, assume the probability that A = 0 is 0.5.

Name__________________________________

Part B: Pipelining (31 points)

Ben Bitdiddle decides that it is a waste of resources to not use the register write port in the writeback stage of **`SW/SB/SH`** and **`BEQZ/BNEZ`** instructions. He comes up with the idea of adding auto-increment/decrement instructions to the DLX ISA.

The formats of the new instructions are as follows (others are similar):

```
SW   rd, (rs1 += offset)
```

whose effects are: *M[rs1 + offset] ← rd*
rs1 ← rs1 + offset **note: offset is a signed value*

```
BEQZ rs1++, offset
BEQZ rs1--, offset
```

which means to increment/decrement the value of rs1 **after** executing the branch.

Question 5 (5 points)

What does Ben have to change in the standard 5-stage pipeline presented in class (see attached diagram at the end of midterm) to get these new instructions to work? Do not worry about control logic and assume the pipeline is fully bypassed. Explain how the new instructions are executed.

Question 6 (4 points)

Are there any **extra** stall conditions required? If not, explain why not. If so, then for each hazard, write a code sequence that exhibits it and indicate how many stall cycles are caused.

Consider the following piece of C code (which is an array copy). **a** and **b** are **integer** (4 bytes) arrays. Assume that `n >= 0`

```
for (i = n; i >= 0; i--) {
  b[i] = a[i];
}
```

The assembly code for the original DLX ISA is as follows.

```
;; Ra = a, Rb = b, Rn = n

     SLLI R5, Rn, 2
     ADD  Ra, Ra, R5
     ADD  Rb, Rb, R5
loop:
     LW   R6, 0(Ra)
     ADDI Ra, Ra, -4
     SW   R6, 0(Rb)
     ADDI Rb, Rb, -4
     BNEZ Rn, loop
     ADDI Rn, Rn, -1
```

This DLX implementation has a single branch delay slot (without annulling).

Question 7 (8 points)

Using the new instructions, implement the above C code. Make the new code fairly efficient. We don't care about register values at the end of execution.

Question 8 (3 points)

How many cycles (issue slots) does each iteration of the loop take to execute (old vs new pipeline)?

Question 9 (4 points)

Do the auto-incrementing store instructions require any special handling for interrupts? Explain.

Question 10 (7 points)

In lecture slide L7-10, a solution is proposed to handle interrupts in delay slots. On an interrupt, if instruction I_i is in a delay slot then save PC_{i-1} instead of PC_i. The branch instruction is thus replayed. Would this scheme work with auto-increment/decrement branches? If so, explain how. If not, explain why not and propose another solution that does work.

Part C: Caches (12 points)

You have just accepted a position at Caches-R-Us as a research scientist. Your first task is to assess the pathological performance of various cache organizations. You decide to start by looking at two basic caches, both with a capacity of four words. The first is a direct-mapped cache with one word per cache line. The second is a fully-associative cache also with one word per cache line and an LRU replacement policy. For both of the following questions assume the caches are initially empty, i.e., all lines are invalid.

Question 11 (5 points)

Please specify a memory access pattern that will cause the fully-associative cache to incur fewer misses than the direct-mapped cache.

Question 12 (7 points)

Does there exist a memory access pattern that causes the direct-mapped cache to incur fewer misses than the fully-associative cache? If so, please give one such access pattern, or else explain why this is not possible.

Name________________________________

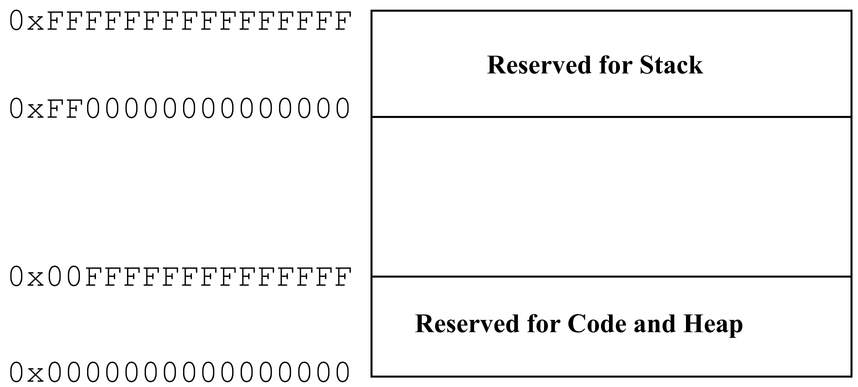
Part D: Hacking Virtual Memory (27 points)

Ben has hired you as a consultant for his newest microprocessor, the Bentium I™. He is experimenting with the idea of using a 64-bit virtual address.

Question 13 (2 points)

How large is the page table if he only uses a single-level page table? Assume that each page is 4KB, each page table entry is 4 bytes, and that the Bentium is a byte-addressable machine.

Ben read in a recent technical journal that many current implementations of 64-bit ISAs implement only part of the large virtual address space. They usually segment the virtual address space into three parts: one used for stack, one used for code and heap data, and the third one unused. For example, the Benhammer™ processor's virtual address space is shown below:



A special circuit is used to detect whether the top eight bits of an address are all zeros or all ones before the address is sent to the virtual memory system. If they are not all equal, an invalid virtual memory address trap is raised. This scheme in effect removes the top seven bits from the virtual memory address, but retains a memory layout that will be compatible with future designs that implement a larger virtual address space.

Question 14 (2 points)

Ben likes the Benhammer scheme but wants an even cheaper virtual memory system, so in the Bentium he decides to remove the top 22 bits and only use the lower 42 bits to index the virtual memory. How large is the single-level page table now?

Question 15 (5 points)

Ben is still unsatisfied about the page table size and asks you to use a three-level hierarchical page table that breaks the 42-bit address into three 10-bit page indices and a 12-bit page offset. If page table overhead is defined as (in bytes):

$$\frac{\text{PHYSICAL MEMORY USED BY PAGE TABLES FOR A USER PROCESS}}{\text{PHYSICAL MEMORY USED BY THE USER CODE, HEAP, AND STACK}}$$

What is the smallest possible page table overhead for the three-level hierarchical scheme? Remember that a complete page table page (1024 PTEs) is allocated even if only one PTE is used. Assume a large enough physical memory that no pages are ever swapped to disk.

Question 16 (5 points)

What is the largest possible page table overhead for the three-level hierarchical scheme? Assume that once a user page is allocated in memory, the whole page is considered to be useful.

Alyssa P. Hacker is unhappy with the large hole in the virtual address space given by the Benhammer scheme. She decides that a hashed page table is the way to go. Again, the machine has a 64-bit virtual address and 4KB pages. The hardware paging system has only one page table with 64 slots, each containing 8 PTEs. Alyssa decides to use ***X mod 64*** as the hash function to select a slot, where ***X*** is the VPN. The page table resides in memory and Alyssa's design has no TLB, so each PTE read requires one memory access. During a page table lookup, all PTEs in each slot are searched sequentially. If there is a miss in the page table, a trap is raised and a software handler will refill the page table, with each refill requiring 10 memory accesses on average.

Question 17 (6 points)

Alyssa is happy with her new modification and runs a very simple benchmark that repeatedly loops over an array of 2^{21} bytes, reading one byte at a time in sequential address order. On average in the steady state, how many memory accesses are performed for each byte read by the user program? Ignore the memory traffic for instruction fetch, assume that the array starts on a page boundary, and there are no other memory accesses in the user code apart from the single byte memory accesses.

Question 18 (2 points)

Alyssa now decides to add a one-entry TLB to the hashed paging system. What should the replacement policy for the TLB be? Circle the most appropriate of the following choices:

a) FIFO
b) LRU
c) Random
d) Doesn't matter, all of the above give the same performance

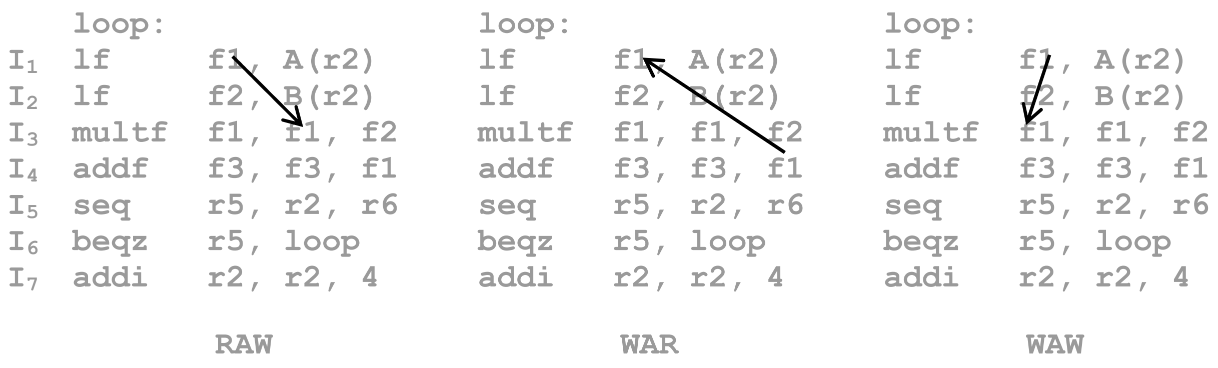
Question 19 (5 points)

Given your answer to the previous question, what is now the average number of memory accesses per user byte read for Alyssa's benchmark?

Name___________________________________

Part E: Scoreboarding (9 points)

The DLX floating-point unit (FPU) uses a separate register set ***f0-f15***.
Consider the following instruction loop. One branch delay slot is assumed.



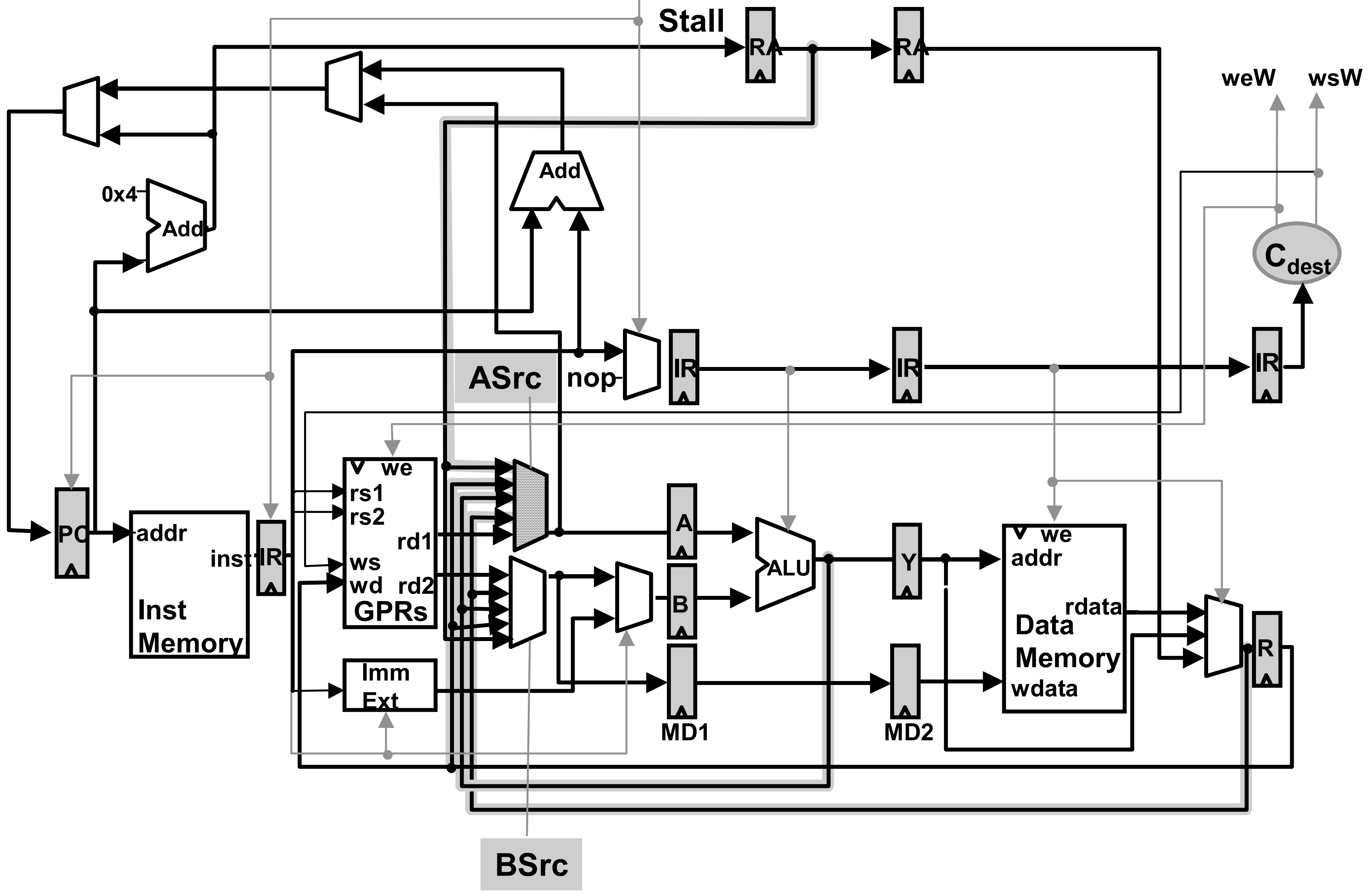
```
    loop:                    loop:                    loop:
I1  lf     f1, A(r2)         lf     f1, A(r2)         lf     f1, A(r2)
I2  lf     f2, B(r2)         lf     f2, B(r2)         lf     f2, B(r2)
I3  multf  f1, f1, f2        multf  f1, f1, f2        multf  f1, f1, f2
I4  addf   f3, f3, f1        addf   f3, f3, f1        addf   f3, f3, f1
I5  seq    r5, r2, r6        seq    r5, r2, r6        seq    r5, r2, r6
I6  beqz   r5, loop          beqz   r5, loop          beqz   r5, loop
I7  addi   r2, r2, 4         addi   r2, r2, 4         addi   r2, r2, 4

           RAW                      WAR                      WAW
```

Question 20 (9 points)

In the figures above, draw all the RAW, WAR, and WAW dependencies. Include dependencies across loop iterations. One example dependency is already given for each type of hazard. Please be neat so that we can clearly see the dependencies you draw.

Name_______________________________

Fully Bypassed Datapath



Name________________________________

Midterm Summary