University of California, Berkeley
College of Engineering
Computer Science Division — EECS

Spring 2003 John Kubiatowicz

Midterm II

May 7th, 2003
CS152 Computer Architecture and Engineering

Your Name:	
SID Number:	
Discussion Section:	

Problem	Possible	Score
1	20	
2	25	
3	30	
4	25	
Total		

[This page left for π]

3.14159265358979323846264338327950288419716939937510582097494 4

Problem 1: Short Answers

Problem 1a[2pts]: Give a simple definition of precise interrupts/exceptions. Why is this important?

Problem 1b[3pts]: Explain why exceptions can occur out of order (in time) in an in-order, 5-stage pipeline. Give an example and explain how to achieve a precise exception point anyway.

Problem 1c[3pts]: Name and *define* 3 types of pipeline data hazards. For each hazard, explain how it is prevented in the 5-stage pipeline:

Problem 1d[2pts]: How do you refresh a DRAM, and *why does this work (i.e. what is happening internally)?*

Problem 1e[2pts]: What are load-delay slots? Does the programmer need to know about them (explain carefully):

Problem 1f[2pts]: Why is it important for Tomasulo to issue instructions in-order?

Problem 1g[2pts]: What is a victim cache? What is it good for?

Problem 1h[2pts]: Name and describe the structure that permits an out-of-order processor to achieve precise interrupts. How does this work?

Problem 1i[2pts]: Suppose you have a processor with a 4K page size, 16K first-level data cache with 128-bit cache lines. Assume that you want to overlap TLB lookup with cache lookup. What is the required associativity of the first-level cache?

[This page intentionally left blank]

Problem 2: Memory Hierarchy

Problem 2a[2pts]: Assume that we have a byte-addressed 64-bit processor with 64-bit words. Suppose that this processor has a 48-word, three-way, set-associative cache (LRU replacement) with 2-word cache lines. Split the 64-bit address into "tag", "index", and "cache-line offset" pieces. Which address bits comprise each piece (one is given)?

tag:
index:
cache-line offset: bits 3 – 0

Problem 2b[2pts]: How many sets does this cache have? Explain.

Problem 2c[7pts]: Assume that the processor makes the following byte accesses. Label each reference address as a Hit (H) or a Miss (M). Also, identify each cache miss as a **compulsory, conflict,** or **capacity** miss.

Byte Address	Hit/Miss?	Miss Type
38(0x026)	Miss	Compulsory
172(0x0AC)		
144(0x090)		
85(0x055)		
424(0x1A8)		
111(0x06F)		
174(0x0AE)		
551(0x227)		
90(0x05A)		
32(0x020)		
428(0x1AC)		
544(0x220)		
96(0x060)		
422(0x1A6)		
170(0x0AA)		

Problem 2d[2pts]: Calculate the cache hit rate (you can leave as a fraction).

[This page intentionally left blank]

Problem 2e[7pts]: You have a 500 MHz processor with 2-levels of cache, 1 level of DRAM, and a DISK for virtual memory. Assume that it has a unified first-level cache. Assume that the memory system has the following parameters:

Component	Hit Time	Miss Rate	Block Size
First-Level Cache	1 cycle	5% Data 1% Instructions	64 bytes
Second-Level Cache	20 cycles + 1 cycle/64bits	2%	128 bytes
DRAM	50ns+ 25ns/8 bytes	0.1% (Page Fault)	16K bytes (Page Size)

Disk Parameters: 50 Mbytes/sec transfer, 10ms average seek, 6000 RPM, 5ms controller time. Finally, assume that there is a TLB that misses 0.1% of the time on data (doesn't miss on instructions) and which has a fill penalty of 100 cycles. What is the average memory access time (AMAT) for Instructions? For Data (assume all reads)? *[1M = 1K × 1K = 1024×1024]*

Problem 2f[5pts]: Suppose that we measure the following instruction mix for benchmark "X":

Loads: 20%, Stores: 10%, Integer: 30%, Floating-Point: 20% Branches: 20%

Assume we have a single-issue processor with minimum CPI of 1.0. Assume we have a branch predictor that is correct 90% of the time, and that an incorrect prediction costs 3 cycles. Finally, assume that data hazards cause an average penalty of 2 cycles for floating point operations. Integer operations run at maximum throughput. What is the average CPI of Benchmark X, including memory misses (from part a)? *[hint: don't forget structural memory hazards]*

Problem #3: Two-way superscalar processors

Consider a dual-issue, *in-order* pipeline with one fetch stage, one decode stage, multiple execution stages (which include memory access) and a single write-back stage. Assume that the execution stages are organized into two parallel execution pipelines (call them *even* and *odd*) that support all possible simultaneous combinations of two instructions. Instructions wait in the decode stage until all of their dependencies have been satisfied. Further, since this is an in-order pipeline, new instructions will be forced to wait behind stalled instructions.

On each cycle, the decode stage takes zero, one, or two ready instructions from the fetch stage, gathers operands from the register file or the forwarding network, then dispatch them to execution stages. If less than 2 instructions are dispatched on a particular cycle, then "NOPs" are sent to the execution stages. When two instructions are dispatched, the *even* pipeline receives the earlier instruction. When only one instruction is dispatched, it is placed in the *even* pipeline.

Assume that each of the execution pipelines consist of a *single linear sequence* of stages in which later stages serve as no-ops for shorter operations (or: every instruction takes the same number of stages to "execute", but results of shorter operations are available for forwarding sooner). All operations are fully pipelined and results are forwarded as soon as they are complete. Assume that the execution pipelines have the following execution latencies: **addf** (2 cycles), **multf** (3 cycles), **divf** (4 cycles), integer ops (1 cycle). Assume that memory instructions take 3 cycles of execution: one for address calculation – done by the integer execution stage, and two unbreakable cycles for the actual memory access. Finally, assume that branch-conditions are computed by integer execution units.

Problem 3a[2pts]: Explain why we would be unable to pick a single optimum number of branch delay slots for the above processor.

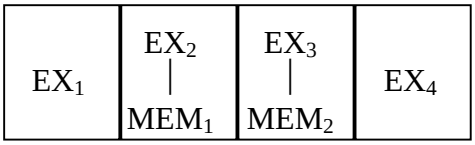
Problem 3b[3pts]: Can we tell the programmer that the number of branch delay slots varies by circumstances? If so, explain the programmer specification for branches. If not, explain why not and (1) indicate how we would "fix" the hardware to have only a specific number of branch delay slots and (2) indicate what that number would be.

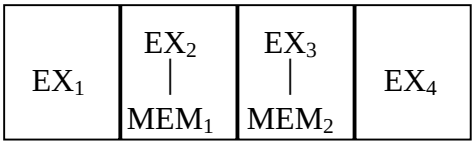
Problem 3c[15pts]: Below is a ***start*** at a simple diagram for the pipelines of this processor.

a)[3pts] Finish the diagram. Stages are boxes with letters inside: Use "F" for a fetch stage, "D" for a decode stage, EX_1 through EX_4 for the execution stages of each of the pipelines (including memory accesses), and "W" for a writeback stage. Clearly label which is the *even* pipeline. Include arrows for forward information flow if this is not obvious.

b)[2pts] Next, describe what is being computed in each EX stage (including partial results).

c)[10pts] Show all bypass paths (as arrows*). Your pipeline should never stall unless a value is not ready*. Label each bypass arrow with the types of instructions that will forward their results along that path (i.e. use "M" for **multf,** "D" for **divf,** "A" for **addf,** "I" for integer operations, and "Ld" for **load results**). *[Hint: think carefully about inputs to store instructions!]*





Problem 3d[2pts]: Does this processor have WAW hazards? Explain. If "yes", give an efficient way to fix the problem.

Problem 3e[2pts]: Does this processor have WAR hazards? Explain. If "yes", give an efficient way to fix the problem.

Problem 3f[3pts]: Assume that the fetch unit presents instructions to the decode stage for execution. The decode stage is free to dispatch zero, one, or two instructions every cycle. *Once instructions have passed decode, they execute to completion (no further blocking).* Assume that enough bypassing hardware has been included to handle every arrow given in (3c).

Suppose that we have the following instruction sequence:

```
ld   r1, 0(r2)
add  r4, r1, r2
```

How many cycles must be inserted between these two instructions by the decode stage to ensure correct execution? How does this translate to user-visible load-delay slots? Explain.

Problem 3g[3pts]: Suppose that we have the following instruction sequence:

```
multf f1, f2, f3
st    0(r1), f1
```

How many cycles will be inserted between these two instructions by the decode stage? How many lost instructions does this represent?

[This page intentionally left blank!]

Problem #4: Fixing the loops

Assume that we have a superpipelined architecture with the following use latencies:

Between a **multf** and an **addf:**	**3 insts**	Between a **load** and a **multf:**	**2 insts**
Between an **addf** and a **divf:**	**1 insts**	Between a **divf** and a **store:**	**7 insts**
Between an int op and a **store:**	**0 insts**	Between two integer ops:	**0 insts**
Number of branch delay slots:	**1 insts**		

Consider the following loop which performs a restricted rotation and projection operation. In this code, F0 and F1 contain **sin(**θ**)** and **cos(**θ**)** for rotation. The array based at register **r1** contains pairs of single-precision (32-bit) values which represent x,y coordinates. The array based at register **r2** receives a projected coordinate along the observer's horizontal direction:

```
project:   ldf      F3,0(r1)
           multf    F10,F3,F0
           ldf      F4,4(r1)
           multf    F11,F4,F1
           addf     F12,F10,F11
           divf     F13,F12,F2
           stf      0(r2),F13
           addi     r1,r1,#8
           addi     r2,r2,#4
           subi     r3,r3,#1
           bne      r3,zero,project
             nop
```

Problem 4a[3pts]: How many cycles does this loop take per iteration? Indicate stalls in the above code by labeling each of them with a number of cycles of stall:

Problem 4b[4pts]: Reschedule this code to run with as few cycles per iteration as possible. Do not unroll it or software pipeline it. How many cycles do you get per iteration of the loop now?

Problem 4c[7pts]: Unroll the loop once and schedule it to run with as few cycles as possible per iteration of the original loop. How many cycles do you get per iteration now?

Problem 4d[4pts]: Your loop in (4c) will not run without stalls. Without going to the trouble to unroll further, what is the minimum number of times that you would have to unroll this loop to avoid stalls? How many cycles would you get per iteration then?

Problem 4e[7pts]: Software pipeline this loop to avoid stalls. Overlap 5 different iterations. What is the average number of cycles per iteration? Your code should have no more than one copy of the original instructions. Ignore startup and exit code.

[This page intentionally left blank!]

[This is an extra page for scratch!]