INSTRUCTIONS

Answer questions 1 through 4 for the Breadth Exam and answer questions 1 through 7 for the Depth Exam. We RECOMMEND that students taking the Depth Exam spend the first two hours on questions 1 through 4, and the remaining two hours on questions 5 through 7. We have tried to make the questions quite specific. If, however, some confusion should arise, be sure to state all your assumptions explicitly.

BREADTH EXAM

1. BCD Arithmetic

Binary coded decimal (BCD) encodes decimal numbers with four bits per decimal digit, where `0000` represents 0, `0001` represents 1, `0010` represents 2, ..., `1001` represents 9, and bit patterns `1010` to `1111` are not used. Assume that you are given gates (NOT, AND, OR, NAND, NOR, etc.) and a one-bit full adder with three inputs (`x`, `y`, `carry-in`) and two outputs (`sum` and `carry-out`). Your answers will be critiqued more for correctness and clarity, than for speed or gate count. Clearly separate the interface specification from the implementation.

(a) Specify an interface to a one-digit decimal adder, ignoring carry-lookahead. Implement it (with gates and one-bit full adders), ignoring carry-lookahead.

(b) Specify an interface for eight-digit decimal adder. Implement it using ripple carry between decimal digits.

(c) Specify an interface to a one-digit decimal adder that allows a multi-digit adder to use carry-lookahead between digits. Implement the new one-digit decimal adder using gates and one-bit full adders.

2. Register Set Alternatives

Some architectures have a unified register set accessed by all instructions (e.g., VAX), some have separate register sets for integer and floating-point instructions (e.g., DLX and MIPS), and some have more than two register sets (e.g., Cray 1).

(a) Discuss the advantages and disadvantages of these alternatives, including the impact on implementation cost, instruction set design, and performance.

(b) Describe how the trend toward superscalar implementations will affect this tradeoff. Explain your predictions.

3. Impact of New Technology on Virtual Memory

Consider the following hypothetical situation. A small Silicon Valley start-up company has just invented a new technology called *3D storage*. For $500 you can purchase one terabyte of non-volatile storage with an access time (to the first byte) of 10 microseconds (NOT milliseconds) with a transfer rate for contiguous data of 1 megabyte per second (one additional byte every microsecond).

How would this technology affect the use and implementation of demand-paged virtual memory? Specifically,

(a) What effect, if any, would this have on the way page faults are handled? Why?

(b) What effect, if any, might this have on the page size? Why?

(c) In most virtual memory systems, main memory is organized as a fully-associative cache of pages. Why might we consider a set-associative organization with this new storage device? Why not?

4. Branches and Instruction Scheduling

You are in the process of implementing the DLX instruction set with delayed branches (from Patterson and Hennessy). Your microarchitecture has a 7-stage pipeline:

ITLB	IF	ID	EX	DTLB	MEM	WB
			FPEX1	FPEX2	FPEX3	FPWB

The functions of each pipeline stage are:

ITLB	Translate the PC into a physical address
IF	Fetch an instruction from the instruction cache
ID	Decode instruction, read registers, evaluate branch conditions
EX	Execute integer ALU instructions or compute effective address
FPEX*	Execute floating point instruction.
DTLB	Translate the effective address into a physical address.
MEM	Access the data cache
WB	Write ALU or load result to integer register file
FPWB	Write FP execution or load result to FP register file

The branch condition is not available until the *end* of the ID stage, therefore branches have a 3-cycle branch delay. Code *must* be scheduled to fill these delay slots, inserting NOPs if necessary. Similarly, load instructions do not return a value until the end of the MEM stage, therefore loads have a delay of 2 cycles. A hardware interlock will cause a stall if a later instruction uses the destination register of a load that has not completed. Finally, floating-point execution occurs in a 3-stage pipelined execution unit. The FP execution unit supports bypassing from the end of the FPEX3 stage, to the beginning of the FPEX1 stage. A hardware interlock will stall a dependent instruction if necessary.

(a) Schedule the following code to minimize the total execution time. Eliminate unnecessary NOP instructions. You may NOT unroll the loop.

```
     do 10 i = 1, 64
        Y(i) = a * X(i) * X(i) + Y(i)
 10  continue

       LD    F0, a
       ADDI  R4, R1, #512      ;last address to load
loop:  LD    F2, 0(R1)   ;load X(i)
       MULTD F2, F0, F2  ;a*X(i)
       MULTD F2, F2, F2  ;a*X(i)*X(i)
       LD    F4, 0(R2)   ;load Y(i)
       ADDD  F4, F2, F4  ;a*X(I) + Y(i)
       SD    F4, 0(R2)   ;store into Y(i)
       ADDI  R1, R1, #8  ;increment index to X
       ADDI  R2, R2, #8  ;increment index to Y
       SUB   R20, R4, R1 ;compute bound
       BNZD  R20, loop   ;branch if non-zero
       NOP               ;delay slot 1
       NOP               ;delay slot 2
       NOP               ;delay slot 3
```

Assuming no additional memory system delays (e.g., cache misses), how many cycles are required per iteration?

(b) Consider extending the instruction set with *canceling* (a.k.a *squashing*) delayed branches. Explain what these are, and how they might help. What additional complications do these branches introduce?

(c) Reschedule the loop assuming a canceling delayed branch. Assuming no additional memory system delays (e.g., cache misses), how many cycles are now required per iteration?

DEPTH EXAM

Answer questions 5 through 7 for the Depth Exam (in addition to questions 1 through 4 which you should have already answered). The questions are quite specific. If, however, some confusion should arise, be sure to state all your assumptions explicitly.

5. Fetch-and-Add and Combining

The NYU Ultracomputer people advocate *fetch-and-add* and *combining*.

(a) Assume that `fetch-and-add(X,n)` denotes a fetch-and-add of the memory location `X` with the value `n`. Assuming `X` is initially 100, what are the possible results returned by `fetch-and-add(X,n)`, `fetch-and-add(X,m)` and `load(X)` executed by three processors at approximately the same time? What are the possible final values of location `X`?

(b) What is combining? What is its potential benefit? How does combining affect you answer to part (a)?

(c) Assume that `compare-and-swap(old,new,X)` stores the value `new` in the memory location `X` only if `X`'s current value is `old`, and then returns a bit that's set if the store was done. Can two or more compare-and-swaps be combined? Why or why not?

6. Compilers and Aggressive Memory Systems

A uniprocessor load-store machine is being designed for very high performance. The memory system contains two levels of cache memory in front of a large main memory that is heavily interleaved. The caches are write-through and use first-in-first-out write buffers to reduce or eliminate stalls on a write. The plan is for the hardware to guarantee correct handling of all memory WAW and WAR hazards, but to ignore memory RAW hazards; it is up to the software to make sure that memory RAW hazards are not violated, and to ensure the correct execution of programs. The software does so by inserting special synchronization operations in the program. The synchronization operation is essentially a "fence" that guarantees that all memory operations initiated before the fence appear to complete before any following operations are initiated.

(a) Explain how the fence operation might be implemented in the memory system described above. How can this fence operation reduce the cost or complexity of the memory system hardware?

(b) A simple-minded compiler could achieve correct execution by inserting a fence before each load instruction. An optimizing compiler would attempt to eliminate redundant fence instructions. What data types (e.g., scalars, arrays, pointers, and floating point) and what control flow constructs (e.g., loops, conditional branches, and procedure calls) pose special problems? Give examples.

(c) Under what conditions would you expect this memory system to be a good idea? Do you predict that this will be used in the future? Why or why not?

7. Memory Latency

Over the years, processor speed has increased much more rapidly than main memory speeds, resulting in a situation where a main memory access requires many processor clock cycles (perhaps several tens or hundreds).

(a) How does the fact that main memory is typically much slower than the processor impact the design of the *processor*?

(b) Discuss several ways to overcome the problem of long main memory latency, detailing the advantages and disadvantages of each approach.