Instructions

For the Breadth Exam, answer questions 1 through 4. For the Depth Exam, answer questions 1 through 7. The questions are quite specific. If, however, some confusion should arise, be sure to state all your assumptions explicitly.

Breadth Exam

(1) There are several design decisions that have to be made when designing a cache memory system. One of these decisions has to do with what should be done on a write operation that does not hit in the cache. With a *write-allocation* policy, the cache line to which the write miss occurred is brought into the cache, while with a *no-write-allocation* policy the write is propagated to memory without changing the contents of the cache.

Does the choice of a write-through or a write-back cache influence whether a write-allocation policy should be used? If a write-allocation policy has been chosen, does its choice influence the choice of a write-through vs. write-back cache?

(2) A conditional branch instruction may be expressed in the form

if A *op* B **goto** label

where *op* is a relation between A and B. The most general case allows A and B to have any (integer) value, while a restriction might be that B is zero. Likewise, *op* might be any of the six possible relations, or it might be restricted to as few as two.

Explain the trade-offs that would determine the set of branch instructions that might be present in a new architecture.

(3) Architectures have allowed instructions of (a) one size (e.g., MIPS, SPARC), (b) a few sizes (IBM 360/370) or (c) many sizes (DEC VAX). What are the advantages and disadvantages of these alternatives?

(4) The 16-bit microprocessors introduced in the late 70s had the capability to handle unaligned operands directly, while the 32-bit microprocessors introduced more recently with new architectures have much more restricted abilities to deal with unaligned variables.

Draw a block diagram of the logic necessary to implement arbitrary byte-alignment of 32-bit memory reads and writes from a memory that is 32-bits wide. What parts of this logic can eliminated by providing only primitives that require software support for unaligned data?

Depth Exam

(5) You are in the process of implementing a computer with a conventional RISC instruction set (e.g., MIPS, SPARC, or DLX). Your initial microarchitecture has a 5-stage pipeline:

The functions of each pipeline stage are:

IF	Fetch an instruction from the instruction cache
ID	Decode instruction
EX	Execute ALU instructions or compute effective address
MEM	Access the data cache and TLB
WB	Write ALU or load result to register file

However, after initial attempts to map this microarchitecture onto the given implementation technology, you discover that the data cache access time is the critical path. This critical path is seriously increasing the machine's cycle time. To compensate for this problem, you are considering changing the microarchitecture by pipelining cache accesses. Specifically, you are considering splitting the MEM stage into a TAG stage, during which the cache tags and TLB are accessed, and a DATA stage, during which the cache data are read or written.

This change will affect the cycle time, average number of cycles per instruction, effectiveness of some compiler optimizations, and the implementation complexity. Discuss the pros and cons of making this change. How will you decide whether or not to make this change?

(6) In addition to storing operands in the memory address space, several instruction set architectures also provide a much smaller storage space inside the CPU. In the past this storage space has consisted of as few as one element — an accumulator. Today, many architectures provide for 32 storage elements (or registers) for integer operands, and even 32 separate registers for floating-point operands.

What factors have led to an increase in the number of storage elements and what are their impact on the instruction-set architecture? What factors could cause a reversal of this trend?

(7) Some architectures have provided support for a fast context switch after every instruction (e.g, the HEP Computer System).

(a) How does one support a context switch that takes ‘‘zero time’’ (or at least less than one instruction time)?

(b) What are the advantages and disadvantages of this approach?

(c) How do longer memory latencies affect the design and merits of this method?