Final Exam

CSC 252

9 May 2014

Directions; PLEASE READ

This exam includes 44 multiple-choice questions. Each is worth 2 points, except for the ones labeled "*mark all that apply*": those are worth 4 points each. The grand total is 100 points. For the 2-point questions, please darken the circle next to the *single best answer*. Be sure to read all candidate answers before choosing.

There are also two essay-style questions that count for extra credit only. For these, please confine your answers to the space provided.

This is a *closed-book* exam. You must put away all books and notes, and *turn off* and put away all cellphones and other electronic devices.

In the interest of fairness, the proctor has been instructed not to answer questions during the exam. If you are unsure what a question is asking, make a reasonable assumption and state it as part of your answer.

You will have a maximum of three hours to complete the exam. The proctor will collect any remaining exams promptly at 11:30 am. Good luck!

1. (2 points) Put your name on every page (so if I lose a staple I won't lose your answers).

2. What explains the sudden move to multicore processors approximately 10 years ago?

◯ **a.** Architects had already exploited most of the instruction-level parallelism available in typical programs.

◯ **b.** Voltages could not easily be reduced any further and still make CMOS circuits function correctly.

◯ **c.** Clock rates could not be increased any further and still cool chips with air.

⊗ **d.** All of the above.

3. To a rough order of magnitude, how long does it take to service a miss in the last-level cache of a modern laptop computer?

⊗ **a.** 100ns

◯ **b.** 1ms

◯ **c.** 10ps

◯ **d.** 1ns

4. To a rough order of magnitude, how long does a typical timesharing operating system (like Linux) allow a thread to run before forcing a context switch?

◯ **a.** 100μs
⊗ **b.** 10ms
◯ **c.** 1s
◯ **d.** 10ns

5. To a rough order of magnitude, what is the access time of a typical modern hard drive?

◯ **a.** 0.1ms
◯ **b.** 1ms
⊗ **c.** 10ms
◯ **d.** 100ms

6. What value does the bit pattern 1111 0111 1000 1111 represent when interpreted as a 16-bit two's complement number?

◯ **a.** 63375
◯ **b.** −30607
⊗ **c.** −2161
◯ **d.** −4322

7. Which kind of dependence can cause data hazards in a single-core, pipelined, in-order processor? *Mark all that apply* (4 points).

⊗ **a.** read-after-write dependence
◯ **b.** write-after-read dependence
◯ **c.** write-after-write dependence
◯ **d.** read-after-read dependence

8. Which of the following techniques can improve cycles-per-instruction (CPI)?

◯ **a.** using a better compiler
◯ **b.** improving the technology and circuits
◯ **c.** using a CISC ISA instead of a RISC ISA
⊗ **d.** using a better organization (microarchitecture—implementation of the ISA)

9. Pipelining typically improves performance by . . .

◯ **a.** decreasing instruction latency

◯ **b.** eliminating data hazards

⊗ **c.** exploiting instruction-level parallelism

◯ **d.** decreasing the cache miss rate

10. Register renaming helps to eliminate . . .

◯ **a.** control hazards

◯ **b.** write-after-write data hazards

◯ **c.** read-after-write data hazards

⊗ **d.** write-after-read data hazards

11. Why does a typical out-of-order processor take pains to *retire* instructions in order (e.g., using a reorder buffer)?

⊗ **a.** to ensure precise exceptions

◯ **b.** to maintain multicore cache coherence

◯ **c.** to improve performance

◯ **d.** to ensure correct behavior in the event of mispredicted branches

12. Suppose we want to average the throughput (requests per second) of several web server implementations. Which kind of average should we use?

◯ **a.** arithmetic mean (1/n*th* of the sum)

⊗ **b.** harmonic mean (1 over 1/n*th* of the sum of the reciprocals)

◯ **c.** geometric mean (n*th* root of the product)

◯ **d.** The answer depends on information not provided in this question.

13. How can loop unrolling improve program performance? *Mark all that apply* (4 points).

◯ **a.** By eliminating data dependences.

⊗ **b.** By reducing bookkeeping operations that do not directly contribute to the result.

⊗ **c.** By exposing more opportunities for code optimization.

◯ **d.** By reducing the cache miss rate.

Questions 14 through 17 make use of the following C code, running on a 32-bit x86 machine.

```
typedef struct {
    union {
        unsigned char a;
        unsigned short b;
    } U;
    unsigned char c;
} S;
```

```
S B[10];
...
S *p = &B[4];
S *q = &B[5];
...
p->U.b = 0x1234;
```

14. What is the value of `q - p`?

⊗ **a.** 1

◯ **b.** 3

◯ **c.** 4

◯ **d.** none of the above

15. What is the value of `((int) q) - ((int) p)`?

◯ **a.** 1

◯ **b.** 3

⊗ **c.** 4

◯ **d.** none of the above

16. What is the value of `((int) &(p->c)) - ((int) p)`?

◯ **a.** 1

⊗ **b.** 2

◯ **c.** 4

◯ **d.** none of the above

17. What is the value of `p->U.a`?

◯ **a.** 0x12

⊗ **b.** 0x34

◯ **c.** 0x2

◯ **d.** 0x4

Questions 18 through 22 concern an L2 cache with 64-byte lines, 4 ways, and 4096 sets, for a machine with 32-bit virtual addresses and 40-bit physical addresses.

18. What will be the total capacity of the cache?

◯ **a.** 16KB
◯ **b.** 256KB
⊗ **c.** 1MB
◯ **d.** 32MB

19. How many bits of an address will be used to index within a given cache block?

◯ **a.** 4
⊗ **b.** 6
◯ **c.** 32
◯ **d.** 64

20. How many bits of a physical address will be used to identify the set in which a block might be cached?

⊗ **a.** 12
◯ **b.** 32
◯ **c.** 64
◯ **d.** 4096

21. How many bits will be used to represent the tag in each cache line?

⊗ **a.** 22
◯ **b.** 24
◯ **c.** 34
◯ **d.** 40

22. Among all the physical blocks in the system, how many will be candidates for caching in a given set?

◯ **a.** 4
◯ **b.** 64
◯ **c.** 4096
⊗ **d.** about 4 million

23. The "memory mountain" diagram on the cover of the textbook allows us to visualize the impact of...

◯ **a.** cache replacement policy

◯ **b.** pipelining

⊗ **c.** locality

◯ **d.** virtual memory

24. Which of the following may lead to a reduction in cache capacity misses? *Mark all that apply* (4 points).

◯ **a.** Switching from write-back to write-through.

⊗ **b.** Increasing cache size.

◯ **c.** Increasing associativity (for a given capacity).

⊗ **d.** Decreasing block size (for a given capacity).

25. What is the principal advantage of a write-back cache (in comparison to a write-through cache)?

⊗ **a.** Reduced memory traffic.

◯ **b.** Lower average miss penalty.

◯ **c.** Lower average hit cost.

◯ **d.** Lower average miss rate.

26. What is the principal advantage of a write-through cache (in comparison to a write-back cache)?

◯ **a.** Reduced memory traffic.

⊗ **b.** Lower average miss penalty.

◯ **c.** Lower average hit cost.

◯ **d.** Lower average miss rate.

27. Which of the following are true of a virtually indexed L1 cache?

◯ **a.** It allows address translation and L1 lookup to proceed in parallel.

◯ **b.** It may force the operating system to allow a given process to use only one page number at a time for a given physical frame.

◯ **c.** It may force the operating system to flush the cache on a context switch.

⊗ **d.** All of the above.

28. Which of the following memories technologies requires periodic refresh?

◯ **a.** SRAM

⊗ **b.** DRAM

◯ **c.** Flash memory

◯ **d.** Phase change memory (PCM)

29. What are the principal tasks of the linker? *Mark all that apply* (4 points).

⊗ **a.** Resolve external references among separately compiled program units.

◯ **b.** Translate assembly language into machine code.

⊗ **c.** Relocate code and data relative to the beginning of the program.

◯ **d.** Enforce access-control restrictions on system libraries.

30. Which of the following is *not* typically a benefit of dynamic linking?

⊗ **a.** Reduction in overall program execution time.

◯ **b.** Reduction in overall space consumption in memory.

◯ **c.** Reduction in overall space consumption on disk.

◯ **d.** Reduction in the cost of software updates.

31. Which of the following is most important to the efficiency of the `fork()` system call?

◯ **a.** Dynamically loaded shared libraries.

◯ **b.** Low-overhead traps from user to kernel space.

◯ **c.** Kernel-level caching of disk locations of commonly executed programs.

⊗ **d.** Copy-on-write memory sharing.

32. Virtual memory serves to . . .

◯ **a.** give every application the illusion that it has the memory to itself.

◯ **b.** give applications the illusion that there's more memory than there really is.

◯ **c.** protect applications from each other.

⊗ **d.** all of the above.

33. Which of the following is *not* typically stored in a page table entry?

◯ **a.** frame number

◯ **b.** user/system read/write/execute permissions

◯ **c.** valid (present) bit

⊗ **d.** owning process id

34. Which of the following is guaranteed in a call to malloc()?

◯ **a.** The returned pointer will always be cache-block aligned.

◯ **b.** The returned pointer will always be non-null.

◯ **c.** The allocated space will be initialized to all zeros.

⊗ **d.** None of the above.

35. Consider an implementation of malloc() whose first four requests are satisfied as follows:

bytes requested	bytes allocated
1	8
5	16
12	16
13	24

What factors likely account for the difference in requested and allocated sizes? *Mark all that apply* (4 points).

◯ **a.** the sequence of previously allocated blocks

⊗ **b.** alignment constraints

⊗ **c.** block header size

⊗ **d.** amount of memory requested

36. Which of the following does *not* correctly characterize the distinction between "system" I/O (read(), write(), etc.) and "standard" I/O (putc(), fgets(), printf(), etc.) in Unix?

⊗ **a.** Standard I/O is more portable than system I/O.

◯ **b.** Standard I/O is buffered in user space; system I/O is not.

◯ **c.** Standard I/O works only on files of characters; system I/O works on binary data as well.

◯ **d.** Standard I/O streams are represented by `FILE*` objects, rather than integer file descriptors.

37. Which of the following does *not* correctly characterize the distinction between TCP and UDP communication?

◯ **a.** TCP is connection-based; UDP is connectionless.

◯ **b.** TCP packets are always delivered successfully in the absence of a lost connection; UDP packets can be dropped.

◯ **c.** TCP packets always arrive in order; UDP packets can be reordered.

⊗ **d.** TCP works across the Internet; UDP is used only on local-area (e.g., machine-room) networks.

38. In Linux terminology, the main difference between processes and threads is

◯ **a.** Processes are managed by the OS kernel; threads are managed by the user-space scheduler.

◯ **b.** Processes are time-sliced; threads run until they block for synchronization or I/O.

◯ **c.** Threads are supported by special hardware on modern processors; processes are implemented entirely in software.

⊗ **d.** Threads share an address space; processes have separate address spaces.

39. Process groups in Linux are useful because . . .

⊗ **a.** you can send a signal to all the processes in a process group at the same time.

◯ **b.** all processes in a process group share the same address space.

◯ **c.** the kernel divides the processor fairly among the various process groups, not the individual processes.

◯ **d.** the kernel will automatically clean up all processes in a given process group when the lead process of the group terminates.

40. A semaphore used as a mutual exclusion lock by a pair of processes should be initialized to

◯ **a.** 0

⊗ **b.** 1

◯ **c.** 2

◯ **d.** The initial value doesn't matter.

41. In the commonly used MESI cache coherence protocol, what is the principal purpose of the E state?

⊗ **a.** To avoid a second bus transaction when a single-threaded program writes a line it previously read.

◯ **b.** To identify the core responsible for responding to BusRd requests from other cores.

◯ **c.** To avoid the need for write-back when evicting an exclusive line whose value is still correct in memory.

◯ **d.** To help identify the oldest line in a set when a victim must be chosen for replacement.

Questions 42 through 44 address the *hardware lock elision* (HLE) mechanism of the latest Intel processors, which we discussed in class. This mechanism redefines two previously unused op-codes (formerly no-ops) as special prefixes, which can be placed before lock acquire and release instructions.

42. What are the potential advantages of using HLE? *Mark all that apply* (4 points).

⊗ **a.** Critical sections that touch disjoint data can execute concurrently.

⊗ **b.** Successful speculation will avoid incurring a cache miss on the lock itself.

◯ **c.** The processor can make an optimal choice between spinning and blocking while waiting for the lock.

◯ **d.** The coherence protocol can avoid the need for BusRdX transactions during elided critical sections.

43. What happens when core B tries to write a location that core A is reading in a critical section whose lock has been elided?

◯ **a.** Core A receives a signal, allowing it to recover from the conflict.

◯ **b.** Core A's speculation fails; it goes back and speculates again.

⊗ **c.** Core A's speculation fails; it goes back and acquires the lock "for real."

◯ **d.** Core B is forced to wait until core A finishes its critical section.

44. What happens when a program using HLE is executed on an older x86 machine?

◯ **a.** It may crash due to an invalid instruction exception.

◯ **b.** It may compute incorrect results.

⊗ **c.** It will perform correctly, but may take more cycles than it would on the most recent machines.

◯ **d.** It may trap into the operating system, which must emulate the missing instructions.

45. (Extra Credit—up to 9 points.) The ISAs of several processors of the early 1980s, including the Berkeley RISC (ancestor of the modern SPARC) and the Stanford MIPS (ancestor of the modern MIPS) had *architecturally visible load delays*. The result of a `load` instruction was not guaranteed to be visible to the immediately following instruction. The compiler or programmer needed to fill that instruction slot with an unrelated instruction (an explicit `nop`, if nothing else); otherwise, incorrect execution could occur.

    (a) Suggest why this design decision might have made sense at the time.

    Answer: These machines had simple in-order, single-issue, 5-stage pipelines, like the one we studied for the Y86. In the technology of the day, an L1 cache hit could be serviced in 2 cycles, making it available (via forwarding) to the instruction after the one in the "delay slot." By forbidding the programmer from putting a dependent instruction in the delay slot, the architects avoided the need for a data-hazard interlock in the common case (a hit). This saved a few transistors, which were a precious resource.

    (b) Explain why it no longer makes sense today.

    Answer: Modern pipelines are longer, they execute instructions out of order (with explicit dependence tracking), and the hit time of the L1 has gotten longer in relative terms (3–5 cycles instead of only 2), so a single-cycle architectural delay no longer suffices: we have to have interlocks even in the cache hit case. Also, we have orders of magnitude more transistors available, so providing the interlocks is easy.

    (c) Explain how it is that modern SPARC and MIPS machines, which no longer have architecturally visible load delays, nonetheless remain backward compatible.

    Answer: A modern SPARC or MIPS processor implements interlocks for all data hazards. If the programmer puts a dependent instruction immediately after a load, the dependence is handled correctly. The new program won't work right on old hardware, but old programs will work just fine on new hardware, which is what we need for backward compatibility.

46. (Extra Credit—up to 8 points.) A multicore processor is said to be *sequentially consistent* if all loads and stores appear to occur in some global total order that is consistent with program order in every core. Sadly, most modern processors are *not* sequentially consistent: memory accesses can appear to occur in different orders from the perspective of different cores—or even in circular order (where A appears to happen before B, B before C, and C before A).

    (a) Give a possible reason why different cores might see stores in different orders.

    Answer: There are many possible reasons. The simplest is the effect of the write buffer in each core, which holds stores that have yet to propagate to memory. Suppose core A writes X, core B writes Y, and then both cores read both locations. A is guaranteed to see the value it wrote to X, but may not see B's write to Y, if it hasn't propagated all the way to memory yet. Meanwhile B is guaranteed to see the value it wrote to Y, but may not see A's write to X, if it hasn't propagated all the way to memory yet.

    (b) All non-sequentially consistent machines provide special (expensive) instructions that can, when desired, be used to force a memory access to be seen everywhere at once, after all previous accesses of the same core, and before all subsequent accesses of the same core. Suggest how the programmer, language, and/or compiler might use such instructions to achieve the illusion of sequential consistency.

    Answer: The simplest solution is to use the special instructions on every load or store of a shared variable, but that will be very expensive. A better solution—and the one most commonly adopted in practice—is to protect every access to a shared variable with a lock or other form of synchronization, so that every pair of conflicting accesses to the same location by different cores is be separated by a synchronization operation. If the expensive instructions are used to implement the synchronization operations, then *all* accesses will appear to happen in consistent order.