

# Final Exam

CSC 252

8 May 2002

## Directions

This exam includes 45 multiple-choice questions, each of which is worth 2 points, for a total of 90 points. There are also two essay-style questions that count for extra credit only. **Please darken the circle next to the best answer.** Be sure to read all candidate answers before choosing.

Put your name on every page. Scratch paper is available if you need it, but the proctor will collect **only the exams**. No partial credit will be given on the multiple-choice questions.

In the interest of fairness, the proctor has been instructed not to answer questions during the exam.

You will have a maximum of 3 hours to work, but you shouldn't need it all. Good luck!

1. What is the two's complement 16-bit additive inverse of  $0x3a2c$ ?

- a.  $0xc5d3$
- b.  $0xba2c$
- c.  $0xc5d4$
- d. none of the above

2. What is  $0xed1c + 0x7959$  in 16-bit two's complement?

- a.  $0x6675$
- b.  $0x16675$
- c.  $0xfe7e$
- d. none of the above

3. Which of the following does *not* result in overflow when calculated in 16-bit 2's complement?

- a.  $0x3039 + 0x5ba0$
- b.  $0xe05c + 0xab39$
- c.  $0xb0b0 + 0x893b$
- d.  $0x8000 - 0x0001$

4. Why does the IEEE floating point standard include *denormal* numbers (numbers near zero with fewer than the normal number of bits of precision)?

- a. To ensure that numbers near zero are evenly spaced on the number line.
- b. To make use of bit patterns that would otherwise be wasted.
- c. To reduce the cost of the division operation.
- d. To ensure that every number has an exact multiplicative inverse.

```

int A[12] = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11};
int main()
{
    char *p = (char *) (&A[0]);
    int *q = (int *) (p + 8);
    printf("%d\n", *q);
}

```

Figure 1: A mysterious program.

```

void *first_free;
void *first_fit(int size)
{
    int *p = (int *) first_free;
    while (*p < size) {          /* size field is in 1st word */
        p = (int *) (*(p+1));   /* next ptr is in 2nd word */
        if (p == 0) return 0;
    }
    return (void *) p;
}

```

Figure 2: A buggy program.

5. Why does the IEEE floating point standard include *NaN* (not-a-number) values?
- a. To provide backward compatibility with the Cray floating point standard.
  - b. To reduce the cost of the division operation.
  - c. To reduce the need for error-checking in long calculations.
  - d. To facilitate the setting of floating-point condition codes.
6. What does the program in figure 1 print on a Pentium machine?
- a. 0
  - b. 2
  - c. 8
  - d. It doesn't print anything; it gets a segmentation error and dies.
7. The code in figure 2, taken from a non-working `malloc` package, is supposed to find an appropriate free block in an explicit free list, or return `nil` if there is no appropriate block. Instead it produces a segmentation error and dies. What needs to be done to fix it?
- a. Increment `p` by 4, not 1, to look at the 2nd word.
  - b. Dereference `p` *before* adding 1 to it, not after.
  - c. Mask out the `free` bit in the size word.
  - d. Check for `p == 0` before trying to check the size.

8. What is the difference between a “.o” object file and an executable file?
- a. The .o file contains assembly language; the executable file contains machine language.
  - b. The executable file has been run through the linker.
  - c. The .o file contains symbol-table information for gdb.
  - d. The executable file has been optimized for faster execution.
9. What does the Pentium pushl instruction do?
- a. Increment the stack pointer and then store its argument at the resulting address.
  - b. Decrement the stack pointer and then store its argument at the resulting address.
  - c. Increment the stack pointer after storing its argument at the previous address.
  - d. Decrement the stack pointer after storing its argument at the previous address.
10. If %eax contains 0x3ac4 and %ecx contains 0xa, what address is specified by the Pentium effective address expression 4(%eax, %ecx, 4)?
- a. 0xeb3c
  - b. 0xeb1e
  - c. 0x3af0
  - d. none of the above
11. What does 8(%ebp) probably access?
- a. a parameter
  - b. a local variable
  - c. a saved register
  - d. the return address
12. If array A is at address 1000 (decimal) and was declared as int A[10][10], what is the address of A[3][4] on a Pentium?
- a. 1136
  - b. 1172
  - c. 1048
  - d. 1034
13. Which of the following techniques would probably *not* stop the buffer overflow attacks you explored in assignment 3?
- a. Include array subscript checks in the code.
  - b. Have the operating system and hardware cooperate to prohibit the execution of instructions in the stack.
  - c. Have the compiler and hardware cooperate to insert a small, random amount of padding into every stack frame, so its address is unpredictable.
  - d. Redesign the stack to grow from low address to high addresses, rather than vice versa.

```

for (i = 0; i < fib(n); i++) {
    if (i % m) c++;
}

```

Figure 3: An inefficient program.

14. The loop in figure 3 is meant to iterate once for every positive integer less than the  $n$ th Fibonacci number. To minimize run-time cost, it makes sense to lift the call to `fib` out of the loop header and place its result in a temporary variable. Why can't the compiler do this for us automatically?
- a. Because it doesn't know whether `fib` modifies `m`.
  - b. Because it doesn't know whether `fib` produces output.
  - c. Because it doesn't know whether `fib` depends on `c`.
  - d. all of the above
15. Why didn't RISC machines appear before the early 1980s?
- a. Nobody thought of them until then.
  - b. Until then you couldn't fit enough transistors on a chip to build a pipelined machine.
  - c. That's when IBM's patent on pipelining expired.
  - d. That's when the gap between processor and memory speeds reached the point at which pipelining became profitable.
16. What is the cycle time of a 1.25GHz processor?
- a. 1.25  $\mu$ s
  - b. 0.8  $\mu$ s
  - c. 1.25 ns
  - d. 0.8 ns
17. Why do recent implementations of the Pentium incorporate a hardware translator that converts IA32 instructions into simpler internal instructions?
- a. To maintain backward compatibility with the 80286.
  - b. To facilitate branch prediction.
  - c. To facilitate pipelining.
  - d. To minimize memory bandwidth.
18. Why do most modern processors place alignment constraints on memory accesses?
- a. To simplify address translation.
  - b. To avoid the possibility that an access may span cache lines.
  - c. To simplify instruction encoding.
  - d. To avoid the need to scale address calculations by the operand size.

19. Why are the keyboard and main memory usually connected to different buses?
- a. To allow the keyboard to communicate with the processor at the same time that the disk is communicating with memory.
  - b. To reduce the cost of the keyboard interface.
  - c. To minimize the cost of handling keyboard interrupts.
  - d. all of the above
20. Which of the following is not generally considered a distinguishing characteristic of RISC machines?
- a. fixed-length instructions
  - b. load/store architecture (other instructions work only on registers)
  - c. limited addressing modes (often only displacement)
  - d. heavily microprogrammed implementations
21. Which of the following does not usually lead to pipeline stalls (bubbles)?
- a. branch mispredictions
  - b. page faults
  - c. consecutive floating point operations
  - d. immediate use of loaded values
22. Which of the following is not commonly used to increase IPC (instructions executed per cycle)?
- a. long pipelines
  - b. multiple pipelines (superscalar execution)
  - c. faster clocks
  - d. out-of-order execution
23. Branch prediction has become increasingly important on recent machines. Why?
- a. Pipelines have gotten longer and more complex.
  - b. Clock rates have increased.
  - c. Processor speeds have increased faster than memory speeds.
  - d. Program sizes have increased.
24. Hand-converting array-based code to pointer-based code is not as useful a technique as it used to be. Why?
- a. Modern compilers can usually do the conversion automatically.
  - b. Modern processors have faster multipliers.
  - c. neither (a) nor (b)
  - d. both (a) and (b)

25. What are the risks of premature hand optimization?
- a. You may make your code harder to understand and maintain.
  - b. You may waste time optimizing stuff that doesn't matter.
  - c. You may implement transformations that the compiler could have done in a safer and more general way.
  - d. all of the above
26. Which of the following limits the ultimate effectiveness of loop unrolling?
- a. Amdahl's Law with respect to loop overhead
  - b. I-cache size
  - c. limited register set
  - d. all of the above
27. Which of the following optimizations is performed by gcc by default (i.e. when you don't specify the `-O` command-line switch)?
- a. loop unrolling
  - b. strength reduction of loop index variables
  - c. moving invariant computations out of loops
  - d. none of the above
28. Register renaming, in which the hardware automatically maps uses of the architectural (instruction set) registers onto a larger set of physical registers, dynamically, serves to reduce the performance impact of which of the following?
- a. write-after-write dependences
  - b. write-after-read dependences
  - c. read-after-write dependences
  - d. (a) and (b)
29. Why do the authors of the text recommend taking a large number of timing measurements and picking the minimum from among them?
- a. To capture "best case" results.
  - b. To weed out "artifacts" like inopportune timer interrupts.
  - c. To warm up the cache.
  - d. all of the above
30. Which of the following is not generally a consequence of higher associativity in a cache design?
- a. lower likelihood of thrashing
  - b. higher hit time
  - c. lower miss time
  - d. less space for real data (as opposed to bookkeeping overhead)

	block size	associativity	capacity
L1 I-cache	128B	direct-map	64KB
L1 D-cache	128B	2-way	32KB
L2 cache	128B	8-way	1.5MB
L3 cache	512B	8-way	32MB

Figure 4: Cache architecture of the IBM pSeries 690 (Regatta) machine.

31. A *write-allocate* cache policy allocates a cache line on a write miss; a *write-no-allocate* policy allocates lines only on read misses. A *write-through* cache keeps memory up-to-date; a *write-back* cache sends dirty lines to memory only when they are evicted. Which of the following combinations makes the least sense?
- a. write-allocate, write-through
  - b. write-allocate, write-back
  - c. write-no-allocate, write-through
  - d. write-no-allocate, write-back

Questions 32 through 34 refer to figure 4, which summarizes the cache architecture of the CS department's new high-end IBM server.

32. How many sets are in the Regatta L1 D-cache?

- a. 128
- b. 256
- c. 512
- d. 1024

33. How many bits are required to identify an L3 cache block within an 8KB page?

- a. 4
- b. 5
- c. 6
- d. 7

34. If main memory consists of 32GB of RAM, how many 512B blocks map to the same set in the L3 cache?

- a. 1024
- b. 2048
- c. 4096
- d. 8192

35. The TLB on the Regatta has 1024 entries. Assuming an 8KB page size and 32GB of RAM, what fraction of memory can be covered by cached translations?
- a. 1/128
  - b. 1/1024
  - c. 1/4096
  - d. 1/8192
36. Which of the following is not usually a privileged (kernel-mode only) operation?
- a. change the mapping between virtual and physical addresses
  - b. change the condition codes
  - c. access the Ethernet
  - d. mask and unmask interrupts
37. Which of the following is not usually performed automatically by the hardware when an exception occurs?
- a. switch to kernel mode
  - b. save registers on the stack
  - c. disable (mask) further exceptions
  - d. jump to a pre-defined address
38. Why do Windows XP and Linux use `trap` instructions to invoke system calls, while Windows ME uses ordinary subroutine calls?
- a. Because the `trap` instructions are more efficient.
  - b. Because Windows XP and Linux were designed more recently.
  - c. Because Windows XP and Linux protect the kernel from user processes, while Windows ME does not.
  - d. Because Windows XP and Linux were designed to run on multiple hardware platforms, not just the IA32.
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. It is not possible for a process in Linux to count the number of signals it receives because
- a. some signals are delivered to a random member of the process group.
  - b. some signals are ignored by default.
  - c. signals are not queued if they arrive while signals are masked.
  - d. signals are discarded if the receiving processes is stopped.
41. Which of the following data structures is not commonly used for hardware-managed page tables?
- a. characteristic array
  - b. linked list
  - c. multi-level tree
  - d. hash table
42. A semaphore used as a mutual exclusion lock should be initialized to
- a. 0
  - b. 1
  - c. 2
  - d. depends on the application
43. Linux pthreads differ from processes in that they
- a. share the same address space
  - b. share the same open files
  - c. support faster context switches
  - d. all of the above
44. Consider a simple non-pipelined processor with a single level of cache. Suppose this processor has a cache miss penalty of 20 cycles and that it sustains a cache hit rate of 90% for data in our favorite application. Suppose further that 20% of the instructions in our application access data. If we could improve the cache hit rate for data to 95%, how much improvement could we expect in average cycles per instruction?
- a. 0.2 fewer cycles per instruction
  - b. 0.4 fewer cycles per instruction
  - c. 4 fewer cycles per instruction
  - d. none of the above

45. Which of the following does not need to be flushed and re-loaded when context-switching between processes on a Pentium?

- a. L1 I-cache
- b. register set
- c. page table root pointer
- d. TLB contents

46. (Extra Credit) Suppose you are writing a concurrent, multi-threaded implementation of the `smooth` function from assignment 5, for use on a 4-processor machine.

- (5 points) You want to give each thread 1/4 of the image to smooth. Would you give the first thread (i) every 4th row, (ii) the first quarter of the rows, (iii) the upper left quadrant of the array, or (iv) something else? Why?

**Answer:** Probably the upper left quadrant. This results in significantly fewer border elements than either bands or interleaved rows.

- (5 points) A *barrier* is a synchronization mechanism that is meant to be executed by *all* threads in a program. When a thread calls the `barrier` function it waits until all threads have called it, at which point they are all allowed to return. A barrier can be called multiple times, in which case a thread making its *n*th call waits until all other threads have made their *n*th call. Explain how you could use barriers to avoid undesirable race conditions in your multi-threaded version of `smooth`.

**Answer:** We need two barriers. Initially each thread makes a local copy of border elements from its neighbors. This is followed by a barrier to make sure all copies are done. Then each thread creates smoothed versions of its own elements. A second barrier ensures that all new versions have been created.

47. (Extra Credit)

- (5 points) Outline, briefly, how the `setjmp()` and `longjmp()` routines might be implemented on a Pentium.

**Answer:** The `jmp_buf` would contain space for the 8 integer registers, the program counter, and, optionally, the 8 floating point registers. The `setjmp` routine would save the registers into the buffer, copy the program counter from the stack into the buffer, and return a zero. The `longjmp` routine would restore the registers and jump indirectly to the saved address. Getting the frame pointer, stack pointer, and program counter right at the same time is slightly tricky. The `longjmp` routine must actually write the saved program counter into the stack at the same location it was stored in `setjmp`, point the stack pointer at it, and `return`.

- (3 points) Why would implementation be harder on a Sparc? Explain.

**Answer:** On a Sparc `setjmp` and `longjmp` must save and restore the state of register windows. In particular, when `longjmp` is executed, the stack may be deeper than it was when `setjmp` was executed. The register window must be popped as many times as the difference between the previous and current stack depth (in frames). This may result in underflow traps, which will be caught and interpreted by the operating system.