CS3330 Exam 3 – Spring 2015

Name: ______________________ CompID: ____________

Letters: Put the letter of your selection or the short answer requested in the box.
Single-select: All multiple-select questions are marked as such.
Write clearly: if we are unsure what you wrote you will get a zero on that problem.
Mark calrifications: If you need to clarify an answer, do so, and also add a `*` to the top right corner of your answer box.
Little-endian is assumed unless otherwise specified.
Pledge the exam or you will get a zero on the entire exam.

. . . .

Question 1: Given `int` variables `a` and `b`, which of the following implements `if (!a) b == 0;`?

A `b &= !!a;`
B `b &= !-a;`
C `b &= !a;`
D `b &= a;`
E `b &= -a;`
F `b &= -!a;`
G none of the above

Answer:

Question 2: The `switch` statement in C and the `int` instruction in assembly both make use of what conceptual data structure?

A a binary decision diagram
B an array of addresses of code blocks
C an array of jump statements
D a bitmask of condition codes
E an array of addresses of functions
F an array of addresses of addresses of code blocks

Answer:

Question 3: We saw in lab that the operating system occasionally stops your code for a few thousand cycles. Users typically want the OS to interrupt your code often (many times a second) so that the computer reacts quickly to their requests. This is because users

A value high throughput
B value low latency
C value low throughput
D value high latency

Answer:

Question 4: Multi-level page tables keep each page table to the size of a single page, which lets us swap out some of the page table to disk. For a C program that use a few dozen megabytes of memory, multi-level page tables also

A improve TLB performance
B reduce the total number of page table entries that exist
C improve data cache performance
D make page lookups require fewer memory accesses
E all of the above
F none of the above

Answer:

Question 5: The assembly instruction `call foo` pushes how many values onto the stack?

A 0
B a variable number dependant on `foo`
C a fixed number > 2
D 2
E 1
F a variable number dependant on both `foo` and the code preceding and/or following the `call`
G a variable number dependant on the code preceding and/or following the `call`

Answer:

Question 6: Register files, caches, memory, and disks each take in a number representing which data value to read (an address, a register ID, or a sector and track number) and either change or return the value of the data stored at that location. What other feature do they all share?

A each of them is missing in some common computer (such as those in cars, phones, etc)
B you can tell by looking at C (or C++ or Java) source code which one(s) will be accessed
C none of them are total functions (there can be "addresses" that do not correspond to any value)
D each of them can cause hazards
E they are each accessed via a bus
F they are all total functions (for every "address" they return a value)
G each of them can cause faults
H all common computers (including those in cars, phones, etc) have all four
I there are assembly instructions to access each of them
J none of the above

Answer:

Information for questions 7–13
Consider the following code:

```
    double *f(int **a, char *b, int c) {
1.      double *ans = malloc(c * sizeof(double*));
        int i, j;
        for(i = 0; i < c; i += 1) {
2.          double *tmp = malloc(sizeof(double) * c);
3.          a[i][i] = b[i] + sizeof(int);
            for(j = 0; j <= i; j += 1) {
4.              tmp[j] += a[i][j];
            }
6.          ans[i] += tmp[i-1];
        }
7.      return &ans;
    }
```

For each of the memory errors listed below, list the line number(s) on which the error occurs, or write “none” if the memory error is not in the code above.
Ignore any unnumbered lines; if the error occurs only on unnumbered lines, write “none”.

Question 7: (see above) Reading uninitialized memory

Answer:

Question 8: (see above) Memory leak

Answer:

Question 9: (see above) Off-by-one indexing error

Answer:

Question 10: (see above) Confusing pointer and value sizes

Answer:

Question 11: (see above) Dereferencing a non-pointer

Answer:

Question 12: (see above) Referencing non-existent variables

Answer:

Question 13: (see above) Incorrect use of `sizeof` in pointer arithmetic

Answer:

Information for questions 14–15
The translation lookaside buffer (TLB) is a cache, but instead of mapping from address to values stored in memory, it maps

Question 14: (see above) from

A physical addresses
B virtual page offsets
C physical page numbers
D physical page offsets
E virtual page numbers
F virtual addresses

Answer:

Question 15: (see above) to

A virtual page numbers
B physical page offsets
C virtual page offsets
D virtual addresses
E physical addresses
F physical page numbers

Answer:

Question 16: The exception number for an interrupt is found on a bus; for a trap it is an operand to the `int` assembly instruction; for a fault it is

A read from a program register
B read from a non-program register
C the "fault occurred" exception number (14 (`0x0e`) in IA32)
D found on the same bus as it would be for an interrupt
E an argument of the faulting instruction
F chosen by the operating system
G found on a different bus than it would be for an interrupt
H none of the above

Answer:

Question 17: You have a set-associative cache and want to decrease the tag size without changing the size of the incoming addresses. You should

A put fewer lines in each set
B put fewer bytes in each block
C put more bytes in each block
D put more lines in each set
E none of the above would make the tag smaller

Answer:

Information for questions 18–19
Suppose a write-back set-associative cache accepts 32-bit (4-byte) addresses; its block size is 128 bits (16 bytes); it has 32 sets of 4 lines each. Assume that the sets implement least-recently-used replacement policy by storing a count in each line representing how many other lines in the set have been accessed since this line was last accessed.

Question 18: (see above) Besides the tag and the stored data, how many more bits does each line need? If nothing else needs to be stored per line, answer "0".

Answer:

Question 19: (see above) How many bits long is the tag? If there is no tag needed, answer "0".

Answer:

Question 20: Graphics Processing Units (GPUs) achieve massive parallelism by having hundreds of processors all execute the same instructions at the same time, but each manipulate their own data values. Conceptually, each processor has its own register file but all share a single program counter. Which of the following Y86 instructions would pose difficulties for a shared PC? Select all that apply.

A `cmovge`
B `irmovl`
C `call`
D `andl`
E `rrmovl`
F `jge`
G `pushl`
H `jmp`
I `xorl`
J `mrmovl`

Answer:

Question 21: We talked about the C library function `setjmp` while were were talking about hardware exceptions because

A it mimics what the hardware does before invoking an exception handler
B it is used to implement the exception table
C it is implemented using exceptions
D it is commonly used inside of exception handlers
E it mimics what the operating system does at the beginning of each exception handler

Answer:

Question 22: Common processor chips uses two voltage states (low and high). If we built a processor chip that used three voltage states instead of two, which of the following would be true?

A Both could perform the same computations in similar numbers of steps
B The three-state chip could perform computations that the two-state chip cannot
C The two-state chip can perform computations that the three-state chip could not
D Both could perform the same computations, but in some cases the three-state chip would need more than twice as many steps
E Both could perform the same computations, but in some cases the three-state chip would need less than half as many steps

Answer:

Question 23: A common trick to swap to integer variables without using a temporary is `x ^= y; y ^= x; x ^= y;` Assemble a similar three-step no-temporary swap out of the following options. Your answer should be exactly three letters in order of operation; for example, if you want to do the first option twice, then the third, answer A A C.

Note: there is at least one correct answer starting from each of the six options.

A `x = x + y;`
B `y = y - x;`
C `x = y - x;`
D `x = x - y;`
E `y = x + y;`
F `y = x - y;`

Answer:

Information for questions 24–25
Assume the following:

- each page table fits on a single page
- you have 6 bits of flags (like "executable", "on-disk", etc)
- page table entries are a power-of-two bytes long (1B or 2B or 4B or 8B or 16B or ...)
- you are minimizing wasted space in the page table(s)

Question 24: (see above) If each virtual address is 28 bits long and each physical address is 24 bits long, how large should pages be?

A 16KB
B 8KB
C 256B
D 2KB
E 512B
F 1KB
G 32KB
H 4KB

Answer:

Question 25: (see above) If your page table entries are 2 bytes (16 bits) long, which of the following is true? Pick the most constrained true statement; for example, if $x \leq 10$ is true, pick $x \leq 10$ and not $x \leq 16$.

A virtual addresses are no more than 22 bits long
B virtual addresses are no more than 16 bits long
C physical addresses are no more than 10 bits long
D physical addresses are no more than 16 bits long
E virtual addresses are no more than 10 bits long
F physical addresses are no more than 22 bits long

Answer:

Question 26: Given `unsigned int` variables `a` and `b`, if `a + b == a ^ b` is true, then which of the following must also be true?

A `a + b == a * b`
B `a + b == a | b`
C `a + b == a - b`
D `a + b == a & b`
E `a == 0 || b == 0`

Answer:

Question 27: Set-associative caches store tags, but page tables do not. Why not?

A tags are the part of the address unused in the rest of the cache lookup process; there are no unused parts of an address in page table lookups
B page tables don't have sets of multiple elements like caches do, so they don't need tags
C tags are used to see if an address is in the cache; every address is in the page tables
D we associate a tag with an entire block of data in a cache; page tables refer to individual bytes so they don't need tags
E while there is nothing called a "tag," page table entries do have something that is functionally identical to the tag

Answer:

Information for questions 28–30
Let

- F be the time needed for the fetch stage to complete (including instruction memory delay).
- D the time needed for the decode stage to complete (including register file delay).
- E the time needed for the execute stage to complete (including condition code setting delay).
- M_r be the time required for a data-memory read.
- M_w the time required for a data-memory write.
- W be the time needed for the writeback state to complete (including register file delay).
- R be the time needed for a register not listed above to store a value.

Question 28: (see above) Which of the following is the best approximation of the minimum clock cycle duration for the five-stage Y86 sequential architecture?

A $F + D + E + \max(M_r, M_w) + R$
B $F + D + E + M_r + M_w + R$
C $\max(F, D, E, M_r, M_w, W, 5R)$
D $F + D + E + M_r + M_w + 5R$
E $\max(F, D, E, M_r, M_w, W, R)$
F $\max(F, D, E, M_r, M_w, W) + 5R$
G $F + D + E + \max(M_r, M_w) + 5R$
H $\max(F, D, E, M_r, M_w, W) + R$

Answer:

Question 29: (see above) Suppose the memory stage is taking far longer than any other stage in our five-stage Y86 pipelined architecture. If we split memory into many stages, what could we expect to happen to the clock speed?

A no change to clock speed
B it depends on how forwarding and stalling are handled
C achieve the same clock speed as if M_r and M_w were zero
D reduce the impact of M_r and M_w but never remove it completely

Answer:

Question 30: (see above) Which of the following is the best approximation of the minimum clock cycle duration for the five-stage Y86 pipelined architecture?

A $\max(F, D, E, M_r, M_w, W) + R$
B $\max(F, D, E, M_r, M_w, W) + 5R$
C $F + D + E + \max(M_r, M_w) + 5R$
D $\max(F, D, E, M_r, M_w, W, R)$
E $F + D + E + \max(M_r, M_w) + R$
F $F + D + E + M_r + M_w + 5R$
G $\max(F, D, E, M_r, M_w, W, 5R)$
H $F + D + E + M_r + M_w + R$

Answer:

Question 31: Consider the following code snippet:

```
for(i = 0; i < n - 0x800000; i += 1) a[i+0x800000] += a[i];
```

Which kind of data cache would make this code run the fastest? Assume that `a` is a `double[]`.
Note: 0x800000 = 8,388,608

A a 4MB direct-mapped cache with 32B lines
B a 1KB full-associative cache with 8B lines
C a 1MB 2-way set-associative cache with 8B lines
D a 16MB direct-mapped cache with 8B lines
E a 128KB 4-way set-associative cache with 8B lines
F a 128KB 2-way set-associative cache with 32B lines

Answer:

Question 32: Clock speed is a direct measure of

A latency
B throughput
C both
D neither

Answer:

Question 33: We saw in lab that the operating system occasionally stops your code for a few thousand cycles; it does this at predictable intervals, every few-hundredths of the second. We later learned that the operating system is activated by means of exceptions. What type of exception would you use to enable this periodic OS control?

A abort
B trap
C fault
D interrupt
E none of the above

Answer:

Information for questions 34–38
For each of the following optimization techniques, select the source(s) of inefficiencies that the optimization is intended to remove. All of these are "select all that apply" questions.

Question 34: (see above) Using multiple accumulators
(e.g., adding alternately to `add1` and `add2` inside a loop, then adding the two together after the loop finishes)

A data cache misses
B redundant copies of the same instructions
C data-hazard based pipeline stalls
D instructions that change neither program registers nor memory
E bookkeeping instructions that move data without changing it
F code that causes the compiler not to perform some other optimizations
G comparison computations

Answer:

Question 35: (see above) Loop unrolling

A redundant copies of the same instructions
B comparison computations
C instructions that change neither program registers nor memory
D bookkeeping instructions that move data without changing it
E data cache misses
F code that causes the compiler not to perform some other optimizations
G data-hazard based pipeline stalls

Answer:

Question 36: (see above) Reordering which loop is inside a set of nested loops

A redundant copies of the same instructions
B data cache misses
C data-hazard based pipeline stalls
D comparison computations
E code that causes the compiler not to perform some other optimizations
F instructions that change neither program registers nor memory
G bookkeeping instructions that move data without changing it

Answer:

Question 37: (see above) Replacing expressions inside a loop with variables set outside of the loop.

A redundant copies of the same instructions
B data-hazard based pipeline stalls
C comparison computations
D instructions that change neither program registers nor memory
E data cache misses
F bookkeeping instructions that move data without changing it
G code that causes the compiler not to perform some other optimizations

Answer:

Question 38: (see above) Replacing functions with macros
(e.g., using `#define cube(x) (x*x*x)` instead of `int cube(int x) { return x*x*x; }`)

A instructions that change neither program registers nor memory
B data cache misses
C comparison computations
D redundant copies of the same instructions
E bookkeeping instructions that move data without changing it
F data-hazard based pipeline stalls
G code that causes the compiler not to perform some other optimizations

Answer:

Question 39: All of the following are possible with virtual memory; which one is also possible without virtual memory?

A hardware can utilize more memory than user code knows about
B user code can use more memory than hardware has
C processes can be prevented from seeing one another's memory
D the operating system can control what addresses user code may access
E multiple programs can share a single copy of each shared library's code
F different processes may use use the same address for different purposes

Answer:

Question 40: Consider a hypothetical computer that lets "bits" have any value between 0 and 1, such as 0.4414 or 0.1112, where the value represents a probability that the number be treated like a 1 instead of like a 0. For example

- `if (0.05) exit(0);` has a 5% chance of ending your program.
- `int x = ((0.5)<<1) | (0.1);` is a number that has a 45% chance of acting like a 3 (0b11), a 5% chance of acting like a 2 (0b10), etc., each time it is used.
- after running `int *a = 0.5, 0.7; int y a[0.8];`, `y` has an 80% chance of being 0.5 (a bit that acts like 1 and 0 with equal probability) and a 20

Which one of the of the following techniques that we discussed this semester would work for this kind of probabilistic analog computer?

A loop unrolling
B exception tables
C the translation lookaside buffer
D float-point numbers
E pipelining
F all of the above would work
G none of the above would work

Answer:

Information for questions 41–42
Common map structures tend to spend most of their time in memory accesses. Hash maps are designed to have random address access order, which reduces cache effectiveness, but can usually find the correct element after just a single memory lookup. Tree map accesses require following one pointer after another, which means a lot of memory accesses; $2\lg(n)$, where n is the number of elements in the map, is a reasonable estimate of the number of accesses needed.

Question 41: (see above) If your hash map is much larger than any of your caches, can code that uses hash maps benefit from the data cache existing?

A yes, because of temporal locality
B yes, but not because of locality
C yes, because of both spatial and temporal locality
D no
E yes, because of spatial locality

Answer:

Question 42: (see above) Assume that L1 cache accesses are 50× faster than main-memory accesses. If every hash map access has cache misses in all caches and every tree node access is an L1 cache hit, how large would your map have to be before the hash map was as fast as the tree map?

Answer:

A $2^{50} \approx 10^{15}$ elements (one quadrillion, or one peta-element)
B $2^{100} \approx 10^{30}$ elements (one nonillion)
C $2^{25} \approx 32$ million elements

Question 43: We discussed both sequential and pipelined architectures. Which of the following applies to pipelined architectures but not sequential architectures?

A hazards
B dependencies
C faults
D exceptions
E all of the above apply to both

Answer:

Information for questions 44–45
In the real world, caches mean that memory can take an unpredictable number of cycles to access. Suppose in our pipelined simulator we added two signals: `dmemBusy` that is `1` if the data memory needs more time to fulfill its last request and `imemBusy` that is `1` if the instruction memory needs more time to fulfill its last request.

Question 44: (see above) What should go in the blanks?

> When `imemBusy` is `1`, bubble __ pipeline registers and stall __ pipeline registers.

Answer with two numbers, like "`5 0`" if we bubble all five pipeline registers. If we don't need any bubbles or stalls, answer "`0 0`". If bubbling and stalling are not sufficient to react correctly to `imemBusy`, answer "`error`"

Answer:

Question 45: (see above) What should go in the blanks?

> When `dmemBusy` is `1`, bubble __ pipeline registers and stall __ pipeline registers.

Answer with two numbers, like "`5 0`" if we bubble all five pipeline registers. If we don't need any bubbles or stalls, answer "`0 0`". If bubbling and stalling are not sufficient to react correctly to `dmemBusy`, answer "`error`"

Answer:

Question 46: Labels in assembly are turned into the addresses of the labeled lines of code when assembled. Labels are used to implement control constructs, method definitions, and

A local variables
B `return` statements
C heap-allocated arrays
D `struct` definitions
E all of the above
F none of the above

Answer:

Pledge:

On my honor as a student, I have neither given nor received aid on this exam.

Your signature here

Computing ID

A