CS 3330 Exam 3 Fall 2018

Name: ________________________________ Computing ID: __________

Letters go in the boxes unless otherwise specified (e.g., for **C** 8 write "C" not "8").
Write Letters clearly: if we are unsure of what you wrote you will get a zero on that problem.
Bubble and Pledge the exam or you will lose points.
Assume unless otherwise specified:

- little-endian 64-bit architecture
- `%rsp` points to the most recently pushed value, not to the next unused stack address.
- questions are single-selection unless identified as select-all

Variable Weight: point values per question are marked in square brackets.
Mark clarifications: If you need to clarify an answer, do so, and also add a `*` to the top right corner of your answer box.

Question 1 [2 pt]: Which of the following code snippets correctly transforms an `x` value of `0x2345` into `0x3452`? **Place a ✓in each box corresponding to a correct answer and leave other boxes blank.**

A ☐ `(x & 0x0FFF) | (x & 0x0FF0 << 4)`

B ☐ `(x<<4) | (x>>12)`

C ☐ `(x << 5)| (x << 2)`

D ☐ `(x & 0x0FFF) | (x & 0xF000)`

Question 2 [2 pt]: Which of the snippets have the best temporal locality? (Assume `n` is large.)

A `for(int i = 0; i < n; i++) { array[i] += i; }`
B
`for(int i = 0; i < n; i++) { array[i] += array[n-i]; }`
C `for(int i = 0; i < n; i++) { array[i%3] = i; }`
D `for(int i = 0; i < n; i++) { array[i+n] = n; }`
E `for(int i = 0; i < n; i++) { array[n-i] = i; }`

Answer:

Question 3 [2.0 pt]: Consider the following cache configuration:

- 2-way
- 16 block cache
- 32 byte cache blocks

When performing cache-blocked for a matrix multiplication of two large matrices of doubles using K by K blocks, the choice of value for K would on the system would probably fall in what range? (Assume that doubles are 8 bytes.)

A less than 3
B between 3-5 (inclusive)
C between 6-7 (inclusive)
D between 8-9 (inclusive)
E 10 or greater

Answer:

Question 4 [2 pt]: The TLB caches ________________.

A page table base registers
B page table entries for each level
C last-level page table entries
D data recently accessed by a process
E none of the above

Answer:

Information for questions 5–9
The following questions ask about running the following assembly snippet on the processor. Unless otherwise stated, assume that the processor uses branch prediction and forwarding and is five stages, as described in the textbook. The assembly snippet is shown along with its machine code. Each line starts with the memory address of the machine code, followed by a list of bytes at that location, with the byte at the smallest address listed first. Each byte is written in hexadecimal. (Remember that the sub a, b subtracts a from b and stores the result in b.) (We have provide scrap paper use it for this question)

```
0x000: 30 f4 00 02 00 00 00 00 00 00 |     irmovq $0x200, %rsp
0x00a: 30 f0 02 00 00 00 00 00 00 00 |     irmovq $0x2, %rax
0x014: 20 07                         |     rrmovq %rax, %rdi
0x016:                               | partA:
0x016: 80 20 00 00 00 00 00 00 00    |     call partB
0x01f: 00                            |     halt
0x020:                               | partB:
0x020: 61 07                         |     subq %rax, %rdi
0x022: 74 20 00 00 00 00 00 00 00    |     jne partB
0x02b: 90                            |     ret
```

Question 5 [2.5 pt]: (see above) Suppose this code were run on a buggy version of the five-stage pipelined processor. If the code results in an infinite loop and does not terminate, which of the following could be the issue? **Place a ✓in each box corresponding to a correct answer and leave other boxes blank.**

A ☐ push, pop, call, and ret instructions modify `%rsp` incorrectly

B ☐ The logic dealing with the condition codes for the sign flag and zero flag is incorrect

C ☐ For call instructions, the processor pushes an incorrect memory address to the stack

D ☐ Instructions are not properly squashed for mispredicted jumps

E ☐ The subq instruction performs subtraction incorrectly (e.g. by writing the result to `%rax` or by subtracting `%rdi` from `%rax`)

Question 6 [2 pt]: (see above) When this code terminates, what value will be in `%rsp`?

A `0x1f8`
B `0x208`
C `0x2`
D `0x200`
E none of the above

Answer:

Question 7 [2.0 pt]: (see above) Which of the following is true about how this code would execute on this processor if it were changed? **Place a ✓in each box corresponding to a correct answer and leave other boxes blank.**

A ☐ The memory address `0x200` contains the value `0x1F`

B ☐ The value `0x2` in the instruction `irmovq $0x2, %rax` could be replaced by any valid, 8-byte value, and it would not alter the number of cycles that it takes to execute

C ☐ When this code terminates, `%rax` contains the value `0x0`

D ☐ If the halt instruction were removed, the code would result in an infinite loop and would not terminate

Question 8 [2 pt]: (see above) What is the value of the condition codes SF and ZF when this code halts?

A SF = 0 and ZF = 1
B SF = 0 and ZF = 0
C SF = 1 and ZF = 1
D SF = 1 and ZF = 0
E not enough information is provided; or the code does not halt

Answer:

Question 9 [2 pt]: (see above) If the first irmovq instruction is fetched during cycle 1, during what cycle will the final halt instruction complete its writeback stage?

Answer:

Question 10 [2 pt]: Consider the following code:

```
char west[13] ="I saw the map";
char *point = west;
point += 2;
point[1] = 'e';
*(west + 1) = ' ';
```

After this executes, what is the value of the array `west`?

A `"I saw the map"`
B `"I sew the map"`
C `"I ew the map"`
D `"I s w the map"`
E none of the above

Answer:

Information for questions 11–14
For these questions, consider a system with

- 64KB (2^{16} byte) pages
- 40-bit virtual addresses
- 36-bit physical addresses
- two-level page tables, with 16 byte page table entries and equal number of page table entries at each level
- a 1024-entry, 8-way TLB with a random replacement policy

Question 11 [2 pt]: (see above) If the second-level page table for virtual address `0x13 4815 1234` is located at physical byte address `0x10 0000`, then what is the address of the second-level page table entry for `0x13 4815 1234`? (Write your answer as a hexadecimal number.)

Answer:

Question 12 [2 pt]: (see above) Accessing the virtual address `0x13 4815 1234` will access the same TLB set as accessing the virtual address ________________. **Place a ✓in each box corresponding to a correct answer and leave other boxes blank.**

A ☐ `0x13 AA15 FFFF`

B ☐ `0x12 3415 6789`

C ☐ `0x81 5134 5678`

D ☐ `0x13 4816 1234`

Question 13 [2 pt]: (see above) With which of the following data cache designs could this system overlap its TLB access with the set lookup for its data cache access? **Place a ✓in each box corresponding to a correct answer and leave other boxes blank.**

A ☐ a 4-way, 128KB data cache with 256B blocks with an LRU replacement policy

B ☐ a direct-mapped, 64KB data cache with 64B blocks

C ☐ an 8-way, 1MB data cache with 1024B blocks with a random replacement policy

D ☐ a 2-way, 512KB data cache with 64B blocks with an LRU replacement policy

Question 14 [2 pt]: (see above) What is the size of physical page numbers on this system? (Write your answer as a base-10 number of bits.)

Answer:

Question 15 [2 pt]: Compared to normal instructions, to be useful, vector instructions require ________________. **Place a ✓in each box corresponding to a correct answer and leave other boxes blank.**

A ☐ extra space in an out-of-order processor's instruction queue

B ☐ loops that use a result from a prior iteration to compute the result of each iteration

C ☐ extra wide registers in the processor

D ☐ a cache with a small hit time

Question 16 [3.0 pt]: Consider the following C function that deallocates a circular singly-linked list:

```
struct node {
    int value;
    struct node *next;
};
void freeLinkedList(node *root) {
    node *current = malloc(sizeof(struct node));
    current = root->next;
    while (current != root) {
        node *temp = current->next;
        free(current);
        current = temp;
    }
    free(root);

}
```

Which of the following statements about this code is true? **Place a ✓in each box corresponding to a correct answer and leave other boxes blank.**

A ☐ `current->value` should be free'd

B ☐ The `current` node should not be malloc'd

C ☐ The `temp` node should be malloc'd

D ☐ `root->value` should be free'd

E ☐ The `current` node should not be free'd inside the while loop

F ☐ The `temp` node should be free'd

Information for questions 17–18
Consider the folllowing C function:

```
double squared_diffs(double *x, double *y, int N) {
    double result = 0.0;
    for (int i = 0; i < N; ++i) {
        double temp = x[i] - y[i];
        result += temp * temp;
    }
    return result;
}
```

Question 17 [2 pt]: (see above) Which of the following optimizations (each of which might be performed by modifying the C code or by the compiler itself) would be useful on the above function? **Place a ✓in each box corresponding to a correct answer and leave other boxes blank.**

A ☐ loop unrolling

B ☐ using pointers to track the current location in `x` and `y` rather than the index `i`

C ☐ cache blocking

D ☐ using vector instructions

Question 18 [2 pt]: (see above) Suppose one transforms the above C code into a version that uses multiple accumulators to optimize the above function, but discovers that,with one particular processor and compiler, the function is no faster even though it was faster on some other combinations of processors and compilers. Which are possible reasons for this? **Place a ✓in each box corresponding to a correct answer and leave other boxes blank.**

A ☐ the processor can only perform one multiplication at a time

B ☐ the processor has a larger TLB

C ☐ the processor has larger cache

D ☐ the compiler ran out of registers for the accumulators

Question 19 [2 pt]: Which of the following are true about linking? **Place a ✓in each box corresponding to a correct answer and leave other boxes blank.**

A ☐ Statically linked files must contain a symbol table to run properly.

B ☐ Linked files contain assembly.

C ☐ Linking occurs after the object file is generated.

D ☐ The input to the linker is an executable file.

Question 20 [2 pt]: Assume an initially empty 16 entry 8-way TLB with an LRU replacement policy on a system with 16-byte pages and 1 byte page table entries and a page table base regitster value of `0x100`. Given the following list of virtual addresses which are accessed, the TLB entry last evicted from the TLB is the one that was stored in the TLB because of what prior access?
`0x124`, `0x144`, `0x154`, `0x174`, `0x134`

A `0x154`
B `0x144`
C `0x174`
D `0x124`
E none of the above

Answer:

Question 21 [2 pt]: Exceptions are triggered by ________________. **Place a ✓in each box corresponding to a correct answer and leave other boxes blank.**

A ☐ running a `ret` instruction without having run a corresponding `call` instruction previously

B ☐ using a privileged instruction in kernel mode

C ☐ running a `ret` instruction to return to an address that is not part of the process's address space

D ☐ input from an external device

Information for questions 22–24
Consider a 3-level page table on a sytem where pages are 256 bytes, page table entries are 2 bytes, and

- first level page tables contain 16 entries
- second level page tables contains 128 entries
- third level page tables contain 128 entries

Question 22 [2 pt]: (see above) What is the maximum possible space (in bytes) that the page tables for a single process can occupy? You may leave your answer as unsimplified arithmetic expression.

Answer:

Question 23 [2 pt]: (see above) If the page containing address `0x100` and containing the address `0x10000` are valid for a process, and no other pages are valid, how much spcae (in bytes) do the page tables for this process occupy? You may leave your answer as unsimplified arithmetic expression.

Answer:

Question 24 [2 pt]: (see above) During a page table lookup, what is the value of the next address that is accessed given an initial virtual address of `0x248567` just after reading a second level page table entry containing physical page number `0x34`?

A `0x340A`
B `0x3405`
C `0x3402`
D `0x3441`
E `0x1534`
F `0x4168`
G none of the above; or there is not enough information to answer

Answer:

Information for questions 25–26
Consider a system with:

- 2 level cache with 16 bytes pages
- page tables that are 1 page in size
- page entries are 1 byte in size, containing, from most significant bit to least:
  - 4 bit physical page numbers,
  - a valid bit,
  - an execute bit,
  - a write bit and
  - a kernel mode bit

Question 25 [2 pt]: (see above) How many bits are used to index into the second level page table?

Answer:

Question 26 [2 pt]: (see above) How much meta data is stored in the first level page table?

Answer:

Question 27 [2.0 pt]: Cache blocking can improve system performance by ______________.
Place a ✓in each box corresponding to a correct answer and leave other boxes blank.

A ☐ improving the temporal locality of program

B ☐ reducing the cache hit time

C ☐ reducing the cache miss rate

D ☐ improving the spacial locality of the program

Question 28 [2 pt]: When an exception occurs, ______________ will use the exception table in order to determine ______________

A the processor / whether the processor was already in kernel mode when the exception occured
B the operating system / the address to set the program counter to
C the operating system / the address the program counter was set to before the exception handler was started
D the processor / the address to set the program counter to
E none of the above

Answer:

Question 29 [2 pt]: Which of the following C code snippets is likely to experience fewest cache misses assuming `n` is large?

Snippet A:

```
for (i=0; i<n; i++) {
  for (j=0; j<n; j++) {
   sum = 0.0;
   for (k=0; k<n; k++) {
     sum += a[i][k]*b[k][j];
   }
   c[i][j] = sum;
 }
}
```

Snippet B:

```
for (k=0; k<n; k++) {
 for (i=0; i<n; i++) {
  r = a[i][k];
  for (j=0; j<n; j++)
   c[i][j] += r*b[k][j];
 }
}
```

Snippet C:

```
for (j=0; j<n; j++) {
 for (k=0; k<n; k++) {
  r = b[k][j];
  for (i=0; i<n; i++)
   c[i][j] += a[i][k]*r;
 }
}
```

Snippet D:

```
for (j=n-1; j>=0; j--) {
 for (k=n-1; k>=0; k--) {
   r = b[k][j];
   for (i=0; i<n; i++)
    c[i][j] += a[i][k]*r;
 }
}
```

Answer:

Information for questions 30–31
Consider the following two assembly snippets:
Snippet A:

```
addq $1, %rax
mulq %rax, %r8
addq $1, %rax
mulq %rax, %r8
```

Snippet B:

```
addq $2, %rax
mulq %rax, %r8
addq $2, %rbx
mulq %rbx, %r9
```

Question 30 [2 pt]: (see above) Consider an (in-order) pipelined processor with the following pipeline stages:

- Fetch
- Decode
- Execute 1
- Execute 2
- Execute 3
- Memory
- Writeback

Assume that the results of ALU operations for `addq` and `mulq` instructions are not available until near the end of the execute 3 stage and any operands for those instructions need to be available near the beginning of the execute 1 stage. The processor uses forwarding to resolve data hazards when possible. On this processor, snippet A executes ____________ than snippet B.

A three or more cycles slower
B two cycles slower
C one cycle slower
D as fast as
E one cycle faster
F two cycles faster
G three or more cycles faster

Answer:

Question 31 [2 pt]: (see above) Consider an out-of-order processor with four pipelined three-cycle multipliers and four pipelined one-cycle adders. On this processor, snippet A executes ____________ than snippet B. (Assume the results of a multiplication or addition can be forwarded to another multiplication or addition with no additional delay and ignore the costs of instruction fetching, etc.)

A three or more cycles slower
B two cycles slower
C one cycle slower
D as fast as
E one cycle faster
F two cycles faster
G three or more cycles faster

Answer:

Question 32 [2 pt]: Which of the following operations are likely to require a system call? **Place a ✓in each box corresponding to a correct answer and leave other boxes blank.**

A ☐ changing the contents of a file

B ☐ replacing values in the processor's cache

C ☐ allocating additional memory

D ☐ displaying a string on the screen

Question 33 [2 pt]: Consider the code below. What does it print out?

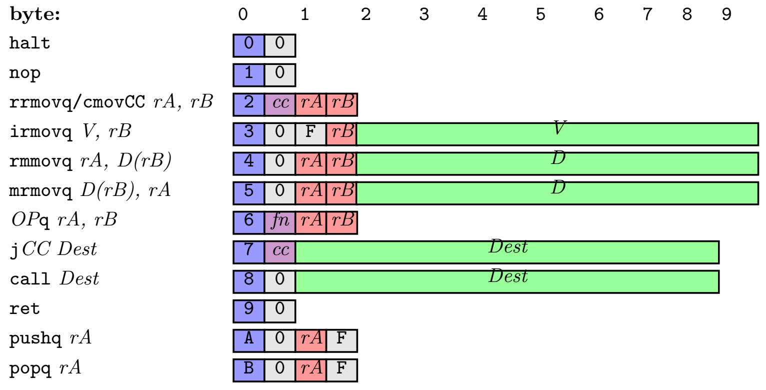
```
int left(){ printf("Frog left "); return 0; }
int right() {printf("Frog right "); return 1;}

int main(){
       printf(" %d ", right() || left());
}
```

A Frog left 1
B Frog right Frog left 1
C Frog right Frog left 0
D Frog right 1
E none of the above

Answer:

Question 34 [2 pt]: Given the executeable below (written as a sequence of hexadecimal byte values) what is the value `%rax` when the program exits? The Y86-64 encoding is provided for reference. Assume that all registers start out with the value `0`, and the `icode` value for the `subq` instruction is 1.



```
00 30 F1 01 00 00 00 00 00 00 00 61 11 00
```

A 2
B 4
C 1
D 0

Answer:

Question 35 [2 pt]: If the following code is run on a big endian machine where `int`s are 4 bytes, what will it print out given the memory layout below?

```
int *x = (int*) 0x12;
printf("%d\n", *x);
```

address	value
0x15	0x00
0x14	0x01
0x13	0x00
0x12	0x00
0x11	0x00
0x10	0x02
0x09	0x00

A 512
B 131072 (2^{17})
C 65536 (2^{16})
D 128
E 256
F 32768 (2^{15})
G none of the above

Answer:

Question 36 [2 pt]: Consider the following loops:
Loop A:

```
for (int i = 0; i < N; ++i)
    A[i] *= B[i-1] + B[i] + B[i+1] - C[i];
```

Loop B:

```
for (int i = 0; i < N; ++i)
    A[i] *= A[i-1];
```

Loop C:

```
for (int i = 0; i < N; ++i)
    A[i] -= A[i+1] + B[i];
```

Loop D:

```
for (int i = 0; i < N; ++i) {
    if (B[i] > C[i]) {
        A[i] *= B[i] + C[i];
    } else {
        A[i] -= B[i] - C[i];
    }
}
```

Which of the above loops are best suited to being optimized with vector instructions assuming the arrays `A`, `B`, and `C` do not overlap?

A Loop D
B Loop A
C Loop C
D Loop B

Answer:

Question 37 [2 pt]: Consider the following program

```
int n = 1024;
int array[1024*1024];

for( i = 0; i < n; i++){
       array[i] += array[n*i];
}
```

Answer:

Assuming that the program is running on machine with an initially empty 16KB (2^{14} byte), 2-way set-associative cache with 32B blocks and LRU replacement policy, and `array[0]`'s address is a multiple of 2^{14}. How many cache misses occur?

Information for questions 38–39
Consider a cache that has 2 ways, 8 sets, 8 bytes per cache block, and an LRU replacement policy. Assume that the cache is initially empty. Suppose 2 bytes at each of the following addresses are accessed (in the order written):

- `0xe28`
- `0xe2a`
- `0xd28`
- `0xb2e`
- `0x868`

Question 38 [2 pt]: (see above) Which of the following statements are true? **Place a ✓in each box corresponding to a correct answer and leave other boxes blank.**

A ☐ If address `0xb29` were accessed next, it would be a hit

B ☐ The cache would have a better hit rate if it used a random replacement policy

C ☐ The cache would evict fewer values if it was direct mapped

D ☐ If address `0xd2b` were accessed next, it would be a hit

Question 39 [2 pt]: (see above) How many cache misses will there be?

Answer:

Question 40 [2 pt]: When switching between two processes, an operating system will save and/or restore________________. **Place a ✓in each box corresponding to a correct answer and leave other boxes blank.**

A ☐ the page table base register

B ☐ the program counter

C ☐ the TLB's contents

D ☐ the exception table base register

Question 41 [2 pt]: Consider a 512B, 8-way set associative TLB, on a system with 8 byte page table entries and 4KB (2^{12} bytes) pages. When the virtual address `0x112345` is looked up in this TLB, what is the tag compared to? (Write your answer as a hexadecimal number.)

Answer:

Question 42 [2 pt]: Pipelining improves upon a single cycle processor by ____________________.
Place a ✓in each box corresponding to a correct answer and leave other boxes blank.

A ☐ Decreasing the amount of time required for arithmetic when executing an instruction

B ☐ Preventing data dependencies from making programs slower

C ☐ Increasing the critical path length

D ☐ Increasing the clock cycle length

E ☐ Increasing the number of instructions that can execute at the same time

Question 43 [2 pt]: Which of the following are true about kernel mode? **Place a ✓in each box corresponding to a correct answer and leave other boxes blank.**

A ☐ more virtual pages can be accessed in kernel mode

B ☐ exception handlers always run in kernel mode

C ☐ executing the `ret` instruction leaves kernel mode

D ☐ vector instructions always run in kernel mode

Question 44 [2 pt]: Which of the following are true about virtual memory? **Place a ✓in each box corresponding to a correct answer and leave other boxes blank.**

A ☐ The translation used in virtual memory is managed by the operating system

B ☐ Virtual page numbers are stored in the TLB

C ☐ Virtual memory provides the illusion that a process has access to all of memory

D ☐ Virtual memory allows two processes to share physical pages

....

Pledge:

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

____

Your signature here