CS 3330 Exam 2 Fall 2018

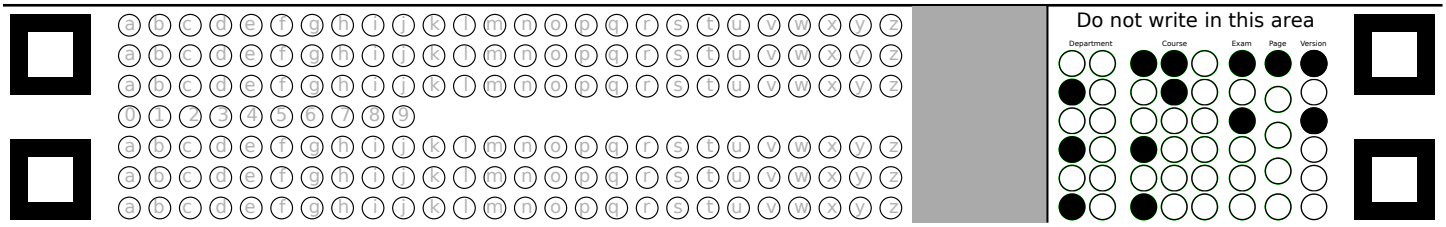
Name: _____ Computing ID: _____

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 [3.0 pt]: Suppose one increases the associativity of a cache while keeping its total size (number of data bytes stored) and block size the same. This change will most likely also increase: **Place a ✓in each box corresponding to a correct answer and leave other boxes blank.**

A ☐ the hit time of the cache

B ☐ the number of sets of the cache

C ☐ the amount of metadata (bookkeeping information like tag and valid bits) stored by the cache

D ☐ the number of compulsory misses a typical program using the cache experiences

E ☐ the number of conflict misses a typical program using the cache experiences

F ☐ the number of offset bits the cache uses

Question 2 [2 pt]: Consider the following implementation of the Caesar Cipher algorithm which is used to encrypt messages using a single key. (Note this is not a secure encryption algorithm)

```
const char* message = "h rdd sgd lzo"
int key = 1
for(i = 0; i < strlen(message) ; ++i){
    ch = message[i];

    if(ch >= 'a' && ch <= 'z'){
        ch = ch + key;
        message[i] = ch;
    }
}
```

How could the run time of the function be improved?

A Move the `if (ch>'z')` check outside of the loop
B Move the string length function outside of the loop
C No optimizations can be made
D Do loop unrolling by computing
`ch = message[i]; ch=message[i+1]`

Answer:

Question 3 [2 pt]: Which is of the following is true about function inlining? **Place a ✓in each box corresponding to a correct answer and leave other boxes blank.**

A ☐ Optimizing compilers will usually perform function inlining across files.

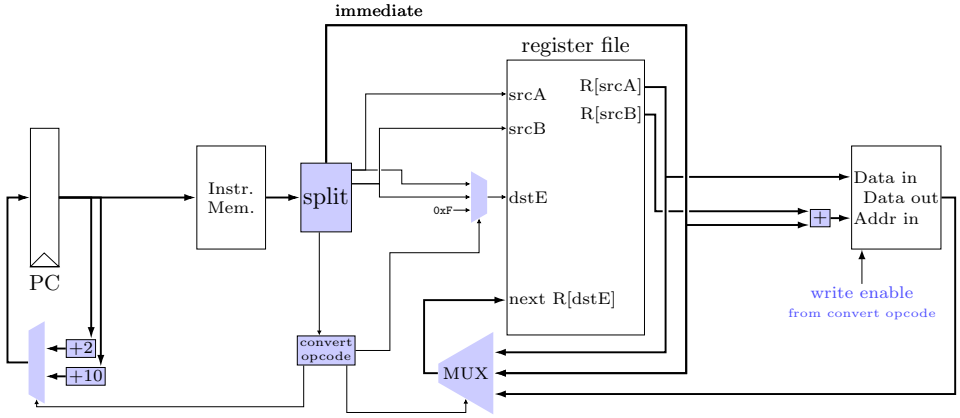
B ☐ Function inlining can increase the number accesses to the stack

C ☐ Function inlining can increase the size of the code stored in the instruction memory

D ☐ Optimizing compilers will perform function inlining for small functions.

Information for questions 4–5

Consider the processor below that only implements mov instructions:



Suppose the components shown above have the following timing:

component	time
register (e.g. PC)	10ps
MUX (each)	10ps
register file read	30ps
register file write	30ps
instruction memory (read)	50ps
data memory (read or write)	50ps
split	0ps
opcode convert	0ps
add (each)	15ps

Assume all components not listed above require negligible time.

Question 4 [2 pt]: (see above) How long should we make clock cycle for the processor above? (Write your answer as a base-10 number of picoseconds.)

Answer:

Question 5 [2 pt]: (see above) Suppose we add pipeline registers which have a 10 picosecond reigster delay to divide this processor into a pipelined processor with a fetch, decode, execute, memory, and writeback stages similar to processor we implemented in HCL. How long should the cycle time of this pipelined processor be? Assume that the MUXes for "dstE" and "next R[dstE]" are part of decode and writeback stages respectively and ignore the costs of any control logic that must be added to handle hazards. (Write your answer as a base-10 number of picoseconds.)

Answer:

Information for questions 6–7

Suppose:

- A benchmark program has a data cache miss rate of 1% on a particular processor;
- 20% of the instructions in this benchmark program use the data cache;
- The data cache's miss penalty (the extra time it requires on a miss) is 200 cycles;
- The data cache's hit time is 1 cycle;
- The processor is: pipelined, normally starts one instruction per cycle, executes instructions in order, and must stall during a data cache miss;
- The pipelined processor *only* stalls because of data cache misses

Question 6 [2 pt]: (see above) What is the average memory access time of the data cache when running this benchmark? (The average memory access time is the average time required for each data cache access.) Write your answer as a base-10 number of cycles. If necessary, you may leave your answer as an unsimplified arithmetic expression.

Answer:

Question 7 [2 pt]: (see above) What is the average number of cycles between when instructions complete when running this benchmark?

A 1 cycle or less
B more than 1.15 cycles and less than or equal to 1.25 cycles
C more than 1.75 cycles and less than or equal to 2 cycles
D more than 2 cycles
E more than 1.25 cycles and less than or equal to 1.45 cycles
F more than 1.45 cycles and less than or equal to 1.75 cycles
G more than 1 cycle and less than or equal to 1.15 cycles

Answer:

Information for questions 8–9

Consider an 8KB, 2-way set associative cache with 16 byte blocks with a write-back, write-allocate policy and a random replacement policy on a machine with 32-bit addresses.

Question 8 [2 pt]: (see above) When splitting up an address into tag, index, and offset bits for this cache, how many tag bits will there be?

Answer:

Question 9 [2 pt]: (see above) The byte at the address `0x12345` will map to the same set as which of the following addresses? **Place a ✓in each box corresponding to a correct answer and leave other boxes blank.**

A ☐ `0x0034F`
B ☐ `0x1234`
C ☐ `0x12456`
D ☐ `0x12346`

Question 10 [2 pt]: Consider the following memory access pattern, given as a list of byte addresses:

0x0 0x8 0x2 0x1 0x4 0x3

Given that you have a byte addressable 2-way associative cache with 2 sets and a block size of 2 bytes with an LRU replacement policy and a write-through, write-no-allocate policy, what bytes from which addresses are stored within index 0 in the cache?

A 0x0, 0x1, 0x2
B 0x2, 0x5, 0x0, 0x1
C 0x0, 0x1, 0x8
D 0x2, 0x3, 0x4
E 0x0, 0x1, 0x8, 0x9
F 0x0, 0x1, 0x4, 0x5
G 0x0, 0x1, 0x4
H 0x4, 0x5, 0x2, 0x3

Answer:

Information for questions 11–12
Consider the following assembly snippet:

```
addq %rcx, %rax
popq %rbx
xorq %rbx, %rax
subq %rbx, %rax
pushq %rax
```

Suppose this assembly snippet is executed on a six-stage pipelined processor with the following stages:

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

Assume this processor implements **forwarding**, using forwarding whenever it would improve performance without substantially increasing the cycle time (an example of what would substantial increase the cycle time is performing the work normally performed by two different stages sequentially within one clock cycle).

For this two part memory stage, the address to read or write must be computed by the beginning of the Memory part 1 stage and any value read will only be available near the end of the Memory part 2 stage.

Question 11 [2 pt]: (see above) On the six-stage processor described above, which of the following forwarding operations will be performed? **Place a ✓in each box corresponding to a correct answer and leave other boxes blank.**

A ☐ of `%rsp` from `popq` to `pushq`

B ☐ of `%rax` from `xorq` to `pushq`

C ☐ of `%rbx` from `popq` to `subq`

D ☐ of `%rax` from `addq` to `xorq`

Question 12 [2 pt]: (see above) On the six-stage processor described above, when the `addq` instruction is in its writeback stage, which instruction is being fetched?

A `xorq %rbx, %rax`
B `subq %rcx, %rax`
C `popq %rax`
D `pushq %rax`
E none of the above

Answer:

Question 13 [2 pt]: Consider the following C function:

```
void scaleArray(char *array, int size, char *ptr) {
    for(int i = 0; i < size; ++i) {
        array[i] *= *ptr;
    }
}
```

Under what conditions can an optimizing compiler's generated code load `*ptr` exactly once into a register rather than loading it again on each iteration of the loop? **Place a ✓in each box corresponding to a correct answer and leave other boxes blank.**

A ☐ if `array != ptr`

B ☐ if `array > ptr` or `ptr >= array + size`

C ☐ if `array[i] != ptr[i]` for all `i`

D ☐ if `array > ptr` and `ptr >= array + size`

Information for questions 14–15

Consider the following two C snippets where `array` is an array of `unsigned char`s.

Version A:

```
for (int i = 0; i < 4; ++i) {
    for (int j = 0; j < 256; ++j) {
        array[i * 256] += array[i * 256 + j];
    }
}
```

Version B:

```
for (int j = 0; j < 256; ++j) {
    for (int i = 0; i < 4; ++i) {
        array[i * 256] += array[i * 256 + j];
    }
}
```

For the below questions assume:

- we run these snippets on a system with a 512 byte, direct-mapped data cache with 16-byte cache blocks
- before each snippet runs, the data cache is completely empty
- the snippets are compiled so only accesses to the `array` use the data cache
- the address of `array[0]` is a multiple of 1024

Question 14 [2 pt]: (see above) How many data cache misses will version A experience given the scenario described above? Write your answer as a base-10 number.

Answer:

Question 15 [2 pt]: (see above) How many data cache misses will version B experience given the scenario described above? Write your answer as a base-10 number.

Answer:

Information for questions 16–18
Consider the following Y86-64 assembly code:

```
    subq %rdx, %rcx
    jl forward
    addq %rcx, %rdx
    xorq %rcx, %rdx
    andq %rcx, %rdx
forward:
    irmovq $100, %rdx
    pushq %rcx
    popq %rax
```

Assume that when this code is executed the value of registers and condition codes are set such that the **jl instruction does not jump to `forward`**. For all the questions below consider the code executing on the five-stage pipelined processor with forwarding and branch prediction we discussed in lecture. Recall that this processor predicts all branches as taken and corrects a misprediction by fetching the correct instruction during the conditional jump's memory stage.

Question 16 [2 pt]: (see above) When the `subq` instruction is in its writeback stage, what instruction will be in its fetch stage assuming the branch was not predicted correctly?

A `jle forward`
B `xorq %rbx, %rdx`
C `irmovq $100, %rdx`
D `addq %rcx, %rdx`
E `andq %rax, %rdx`
F none of the above or there is not enough information to answer

Answer:

Question 17 [2 pt]: (see above) The value of `%rdx` computed by `addq` will be forwarded to which of the following instructions? **Place a ✓in each box corresponding to a correct answer and leave other boxes blank.**

A ☐ `irmovq $100, %rdx`
B ☐ `andq %rcx, %rdx`
C ☐ `xorq %rcx, %rdx`
D ☐ `subq %rdx, %rcx`

Question 18 [2 pt]: (see above) The value of `%rcx` computed by `subq` will be forwarded to which of the following instructions? **Place a ✓in each box corresponding to a correct answer and leave other boxes blank.**

A ☐ `andq %rcx, %rdx`
B ☐ `addq %rcx, %rdx`
C ☐ `pushq %rcx`
D ☐ `xorq %rcx, %rdx`

Question 19 [2 pt]: Consider that the code below is running on a system with an 8B fully-associative data cache with 2-byte blocks and an LRU replacement policy, where both `message` and `key` are char arrays whose first byte is in the first byte of a cache block. Assume only accesses to `message` and `key` use the data cache, and the data cache is initially empty. How many cache misses does this code generate?

```
int n = 8;
for (i = 0; i < n; i += 2){
    for(j = 0; j < n; j += 2){
        for(iB = i; iB <= i+1; iB++){
            for(jB = j; jB <= j+1; jB++){
                message[jB*n + iB] ^= key[iB*n + jB];
            }
        }
    }
}
```

Answer:

Question 20 [2 pt]: Which of the following improves the performance of a program with high temporal locality?

A Decreasing the total size of cache by decreasing the block size
B Increasing the block size of cache but keeping the total cache size the same.
C Decreasing the total size of cache by decreasing the number of sets
D Increase the number of sets in the cache but keeping the total cache size the same

Answer:

.....

Pledge:

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

_____

Your signature here

This page intentionally left blank.