# CS 3330 Exam 2 Spring 2018

**Name:** EXAM KEY **Computing ID:** KEY

**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]:** Suppose one increased the size of cache blocks but kept the total data size and associativity of a cache the same. Which of the following are likely results? **Place a ✓in each box corresponding to a correct answer and leave other boxes blank.**

**A** [✓] increasing the number of conflict misses

**B** [ ] decreasing the number of conflict misses

**C** [✓] decreasing the number of compulsory misses

**D** [ ] increasing the number of compulsory misses

**Information for questions 2–3**

The following questions ask about running the following assembly snippet on the five-stage pipelined processor which uses branch prediction and forwarding as described in our textbook, which predicts all branches as taken and corrects misprediction by fetching the correct instruction during conditional jump's memory stage.

```
part1:
    irmovq $1, %rax
    addq %rax, %rax
    je part2
    halt
part2:
    xorq %rax, %rax
    je part1
    halt
```

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

Answer: 10

**Question 3 [2 pt]:** (see above) When the instruction `je part2` is in the execute stage, which instruction is in the fetch stage?

**A** `addq %rax, %rax`
**B** `je part1`
**C** `xorq %rax, %rax`
**D** `irmovq $1, %rax`
**E** none of the above

Answer: B

**Information for questions 4–6**
Consider the following Y86 assembly code:

```
mrmovq 8(%r8), %r11
addq %r11, %r8
subq %r8, %r11
popq %r8
```

**Question 4 [2 pt]:** (see above) Suppose this snippet is executed on a four-stage pipelined processor with the following stages:

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

The split execute stage requires the operands for an addition or subtraction to be available near the beginmning of the execute 1 stage, and only has results available near the end of the execute 2 stage. Assume this processor implements forwarding and branch prediction to resolve hazards when possible. When this processor is executing the execute 1 stage of `subq`, it is executing the _____ stage of the `addq` instruction?

**A** fetch and decode
**B** execute 1
**C** execute 2
**D** memory and writeback
**E** none of the above — the instruction is not in the pipeline

Answer: D

**Question 5 [2 pt]:** (see above) When this snippet is executing a five-stage pipelined processor, with forwarding and branch prediction as described in our textbook, if the `mrmovq` instruction is fetched during cycle number 1, then the `popq` instruction does its writeback stage in what cycle number? Write your answer as a base 10 number.

Answer: 9

**Question 6 [2 pt]:** (see above) When this snippet is executing a five-stage pipelined processor, with forwarding and branch prediction as described in our textbook, in which of the following ways are values forwarded between instructions? **Place a ✓in each box corresponding to a correct answer and leave other boxes blank.**

**A** [✓] `%r8` from `addq` to `subq`
**B** [✓] `%r11` from `mrmovq` to `addq`
**C** [ ] `%r8` from `addq` to `popq`
**D** [ ] `%r8` from `mrmovq` to `addq`

**Information for questions 7–8**

For these questions, consider the following cache access pattern, where each access is to one byte at the specified address:

- read `0x10`
- read `0x12`
- write `0x13`
- write `0x14`
- write `0x20`
- read `0x13`
- read `0x15`

For this question, consider an access a write miss if the address being written is not stored in the cache, *regardless of whether it will be eventually brought into the cache.*

**Question 7 [2 pt]:** (see above) With an initially empty, 16-byte direct-mapped cache with a 4-byte blocks, and a write-back, write-allocate policy, how many **read and write misses** will there be?

Answer: 4 (0x10, 0x14, 0x20, 0x13)

**Question 8 [2 pt]:** (see above) With an initially empty, 16-byte direct-mapped cache with a 8-byte blocks, and a write-through, write-no-allocate policy, how many **read and write misses** will there be?

Answer: 2 (0x10, 0x20)

**Information for questions 9–11**

Consider a pipelined processor with the following pipeline stages, which takes the indicated amount of time to complete its computations (or data writes) **excluding a 10 picosecond pipeline register delay**:

- fetch — 200 ps
- decode — 100 ps
- execute — 100 ps
- memory — 200 ps
- writeback — 100 ps

For each of the questions below, **write your answer as a base-10 number of picoseconds**. *If you think there is not enough information to tell, write "unknown".*

**Question 9 [2 pt]:** (see above) What is the cycle time of this processor? Write your answer as a base-10 number of picoseconds.

Answer: 210 ps (half-credit for 200 ps)

**Question 10 [2 pt]:** (see above) When there are no hazards, what is the time from when the processor starts the fetch stage of an instruction until when it completes the writeback stage of that instruction? different answers are for different interpretations of when the writeback stage is completed (end of clock cycle or end of work for writeback)

Answer: 1050 ps or 940 ps or 950 ps (half-credit for 1000 ps or 1040 ps or 900 ps)

**Question 11 [2 pt]:** (see above) Suppose every 10th instruction requires one cycle of stalling to handle hazards and there are no other performance costs to handling hazards. What would be the resulting *mean* time between when instructions complete their writeback stage? also accept: 21 ps (half for 20ps; mean time from end of one writeback stage to beginning of next) or 121 or 126 or 131 ps (half for 120 ps or 125 ps or 130 ps; mean time from end of work for one writeback stage to beginning of next)

Answer: 231 ps (half-credit for 220 ps) plus see alternate accepted answers

**Information for questions 12–13**
Consider a 4-way, 16KB set associative cache with 16-byte cache blocks with a random replacement policy and a write-back, write-allocate policy, and a 16-bit address size. (1KB = $2^{10}$ bytes.)

**Question 12 [2 pt]:** (see above) Which of the following forms of metadata does this cache need to store alongside every cache block? **Place a ✓in each box corresponding to a correct answer and leave other boxes blank.**

**A** [✓] a dirty bit

**B** [✓] the tag bits

**C** [ ] the index bits

**D** [ ] information to implement the replacement policy

**Question 13 [2 pt]:** (see above) Which addresses would map to the **same set** as `0x1234`? **Place a ✓in each box corresponding to a correct answer and leave other boxes blank.**

**A** [ ] `0x1A34`

**B** [✓] `0x1235`

**C** [ ] `0x1004`

**D** [✓] `0xF23F`

**Information for questions 14–15**
The following questions concern the following C code:

```
void computeSums(int *array, int *results, int result_count, int array_size) {
    for (int i = 0; i < result_count; i += 1) {
        for (int j = 0; j < array_size / result_count; j += 1) {
            result[i] += array[j * result_count + i];
        }
    }
}
```

**Question 14 [2 pt]:** (see above) Because of the potential for the arrays `results` and `array` to overlap, a compiler can **not** apply which of the following optimizations **unless it adds an extra check for overlap** (or similar) in the generated assembly? **Place a ✓in each box corresponding to a correct answer and leave other boxes blank.**

**A** [✓] keeping `result[i]` in a register during the inner loop

**B** [ ] replacing multiplications by `result_count` in the inner loop with addition of `result_count * sizeof(int)` to a pointer computed in a prior iteration

**C** [✓] changing the loop order by swapping the inner and outer for loops to improve locality in `array`

**D** [ ] computing `array_size / result_count` once rather than during each iteration of the inner loop

**Question 15 [2 pt]:** (see above) Which of the following would be effects of unrolling the inner loop four times in the above C code? **Place a ✓in each box corresponding to a correct answer and leave other boxes blank.**

**A** [ ] for large `result_count`s, the function would have fewer data cache misses

**B** [ ] the function would take up less space in memory

**C** [✓] for large `result_count`s, the function would execute less instructions comparing and/or manipulating `j`

**D** [ ] the function would use more registers

**Information for questions 16–18**

The following question ask about running the following assembly snippet on the five-stage pipelined processor which uses branch prediction and forwarding as described in our 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.

```
0x000: 30 f4 00 01 00 00 00 00 00 00 |    irmovq $0x100, %rsp
0x00a: 30 f7 02 00 00 00 00 00 00 00 |    irmovq $2, %rdi
0x014: 30 f6 01 00 00 00 00 00 00 00 |    irmovq $1, %rsi
0x01e: 30 f2 3e 00 00 00 00 00 00 00 |    irmovq $0x3e, %rdx
0x028: a0 2f                         |    pushq %rdx
0x02a:                               | start:
0x02a: 61 67                         |    subq %rsi, %rdi
0x02c: 76 2a 00 00 00 00 00 00 00    |    jg start
0x035: 73 3f 00 00 00 00 00 00 00    |    je finish
0x03e: 00                            |    halt
0x03f:                               | finish:
0x03f: 90                            |    ret
0x040: 00                            |    halt
```

**Question 16 [2 pt]:** (see above) When this code terminates, what value will be in `%rdi`?

**A** -1
**B** 0
**C** 2
**D** 1
**E** none of the above

Answer: B

**Question 17 [2 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** [✓] If `ret` were replaced by `call 0x03e`, it would decrease the number of cycles the program takes to run.

**B** [ ] If `ret` were replaced by a `halt` instruction, it would not change the number of cycles the program takes to run.

**C** [ ] The halt instruction at address `0x03e` could be replaced with any valid one-byte instruction without changing the result of the program, since it never gets executed

**D** [✓] If `irmovq $1, %rsi` were replaced by `irmovq $10, %rsi`, it would decrease the number of cycles the program takes to run.

**Question 18 [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: 21

**Information for questions 19–21**
Consider the following C code:

```
unsigned char A[1024 * 8];
...
for (int i = 0; i < 2; ++i) {
    for (int j = 0; j < 8; ++j) {
        A[j * 1024] += j + A[j * 256 + 4];
    }
}
```

Assume `A` is located at a memory address which is a multiple of $2^{20}$ and this code is compiled so that no memory accesses to `A` are reordered or omitted and all other values are kept in registers.

Note that $1024 = 2^{10}$ and $256 = 2^8$ and `unsigned char`s are 1 byte.

**Question 19 [2 pt]:** (see above) If this code is run with a 4-way, 4 KB set-associative cache with four-byte cache blocks and an LRU replacement policy, how many cache misses will the `for` loop over `i` experience? Assume the cache is empty when the loop is started.

Answer: 24

**Question 20 [2 pt]:** (see above) If this code is run with a 512-byte direct-mapped data cache with four-byte cache blocks, how many cache misses will the `for` loop over `i` experience? Assume the cache is empty when the loop is started.

Answer: 32

**Question 21 [2 pt]:** (see above) If this code is run with a 12-byte fully-associative cache with 1 byte cache blocks, which replacement policy for that cache would result in the highest hit rate?

**A** first-in, first-out
**B** LRU
**C** random
**D** multiple of the above policies are tied for highest hit rate

Answer: C

## Pledge:

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

Your signature here