CS 3330 Exam 2 Fall 2019

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]: What is the total amount of data (in bytes) that can be stored in a two way set associative cache with 16 sets and 256 byte blocks?

Answer: 8K (or accept anything around 8000)

Question 2 [2 pt]: Which assembly instructions **can** be issued (sent to execution units) out-of-order before the mrmovq instruction below completes?

```
1| mrmovq 0(%r10), %r8
2| addq %r10, %r11
3| xorq %r8, %r9
4| subq %r9, %r10
5| xorq %r11, %r12
```

Write the numbers of the lines that get executed. (For example, if you think lines 2 and 3 can be executed with the mrmovq, then write "2 and 3".)

Answer: 2, 5

Information for questions 3–3

Consider the following C code:

```
unsigned char array[512 * 512];
/* ... */
int sum = 0;
for (int x = 0; x < 5; ++x) {
        for (int i = 0; i < 128 + 64; ++i) {
            sum += array[i];
        }
}
```

Assume that:

- only `array` is kept in memory (all other variables are stored in registers)
- array is stored at an address that is a multiple of 4096 (2^{12})
- that the compiler does not remove or reorder any accesses to `array`
- unsigned chars are 1 byte
- the cache is empty upon entry to the outer for loop above

Question 3 [2 pt]: (see above) If the loops above are run on a 128B direct-mapped cache with 1-byte cache blocks, how many data cache misses will occur? (You may write your answer as unsimplified arithmetic expression.)

Answer: $128 * 5 + 64 = 704$

Information for questions 4–5

Consider the following access pattern where W represents writes and R represents reads:

W	`0xABC`
R	`0xBAD`
W	`0xBAC`
R	`0x00D`
W	`0xBAE`

Assume that the access pattern is **repeated 4 times** on processor with a 48 byte fully associative cache with 16 byte blocks. Initially the cache is empty. The cache is using a write-allocate, write-back policy and LRU replacement policy.

Question 4 [2 pt]: (see above) How many total cache misses will occur (during those 4 repeats)?

Answer: 3

Question 5 [2 pt]: (see above) After all 4 repeats of the access pattern above occur, the cache may contain some blocks which must be written to memory if they are evicted anytime later. Write the tags of each of these blocks. (If there are no such blocks, write "none".)

Answer: 0xAB and 0xBA

Question 6 [2 pt]: Which of the following is/are true? **Place a ✓in each box corresponding to a correct answer and leave other boxes blank.**

A [✓] increasing the cache size generally decreases the miss rate

B [ ] increasing the associativity of the cache while keeping the cache size the same decreases capacity misses.

C [ ] increasing the cache size generally decreases the hit time

D [ ] increasing the associativity of the cache while keeping the cache size the same generally increases the miss rate

Information for questions 7–8
Consider the following assembly code

```
mrmovq 8(%rsp), %r10
subq %r10, %r12
rmmovq %r12, (%rsp)
xorq %r12, %r11
addq %r10, %rsp
```

Question 7 [2 pt]: (see above) Assume the above code is run on the five-stage pipelined processor described in lecture **using only stalling to resolve data hazards**. How many stalls are needed to correctly execute the above code? Count each cycle in which a new instruction isn't fetched as a single stall. (Write your answer as an integer.)

Answer: 6

Question 8 [2 pt]: (see above) Assume the above code is run on the five-stage pipelined processor described in lecture which uses **forwarding** to avoid stalling when executing the above code. Write the minimum set of registers that would must be forwarded. Example answer: `%r13,%r14`. If no forwarding is helpful, write `none`. should have said "avoid stalling (when possible) when executing the above code"

Answer: %r12,%r10 (half-credit for adding %rsp or omitting any one)

Question 9 [3.0 pt]: Consider the following C function:

```
void outerProduct(int n, int *a, int *b, int *result) {
    for (int i = 0; i < n; ++i) {
        for (int j = 0; j < n; ++j) {
            result[i * n + j] += a[i] * b[j];
        }
    }
}
```

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

A [ ] on some processors, the function could benefit from using multiple accumulators for the additions to the elements of the `result` array no repeated chains of addition, etc.

B [✓] unrolling the inner loop in this function would likely increase its size in the executable

C [✓] a compiler cannot store `a[i]` in a register between iterations of the innermost loop without adding checks that certain pointers refer to different areas of memory

D [ ] a compiler cannot unroll the loops without adding checks that certain pointers refer to different areas of memory

E [ ] the locality of the function would be improved by swapping the loop over `i` and the loop over `j`

F [✓] a compiler cannot reorder the loops without adding checks that certain pointers refer to different areas of memory

Question 10 [2 pt]: Consider the code below running on a system with a 256B fully-associative data cache with 16-byte blocks and an LRU replacement policy. Assume `matrix` is a very large array of 4-byte integers, and is aligned (so the first byte of `matrix[0]` is the first byte of a cache block). Assume only accesses to `matrix` use the data cache and that cache is initially empty.

```
int n = 160;
for (i = 0; i < n; i += 4){
    for (j = 0; j < n; j += 4){
        for (iB = i; iB <= i + 3; iB++){
            for (jB = j; jB <= j + 3; jB++){
                matrix[jB * n + iB] *= i*j;
            }
        }
    }
}
```

How many cache misses occur when the code is run?

Answer: 4 misses per block * $40 \cdot 40$ blocks $= 1600 \cdot 4 = 6400$

Information for questions 11–12
Consider the following assembly snippet. Initially, `%r11` contains `0x1`; `%r12` contains `0x1`; and `%rsp` contains `0x13`.

Assuming a 5 stage pipelined processor like the one described in lecture, with forwarding and a branch prediction scheme that assumes all branches are taken.

```
     subq %r11, %r12
     jne  bar
     call foo
     jmp end
bar: add %rsp, %r11
     jmp end
foo: add %r11, %r11
     ret
end: nop
```

Question 11 [2 pt]: (see above) If the `subq` is fetched during cycle 0, then during what cycle is the `nop` fetched? subq (0), jne (1), mispredict (2), mispredict (3), call (4), add (5), ret (6), stall (7,8,9), jmp (10), nop (11)

Answer: 11

Question 12 [2 pt]: (see above) What value is stored for register `%r11` **in the register file** when the branch misprediction is discovered for the `jne` instruction?

Answer: (answer `0x1`. Unchanged)

Information for questions 13–14

Suppose a six-stage pipelined processor with forwarding and branch prediction has the following pipeline stages:

- fetch 1
- fetch 2
- decode
- execute
- memory
- writeback

Question 13 [2 pt]: (see above) Suppose this processor experiences a branch misprediction. It detects and handles that misprediction during the **conditional jump instruction's decode stage**. In the following cycle, as a result of correcting this misprediction, which of the following pipeline registers should output values corresponding to a nop? half credit if off-by-one; 0 otherwise **Place a ✓in each box corresponding to a correct answer and leave other boxes blank.**

A [✓] the ones between fetch 1 and fetch 2

B [✓] between fetch 2 and decode

C [ ] between decode and execute

D [ ] between execute and memory

E [ ] between memory and writeback

Question 14 [2 pt]: (see above) Suppose this processor is executing an instruction in the memory stage and discovers that it must stall due to a data cache miss, keeping that instruction in the memory stage. As a result of this stall, which of the following pipeline registers should output values corresponding to a nop in the next cycle? half credit if off-by-one; 0 otherwise **Place a ✓in each box corresponding to a correct answer and leave other boxes blank.**

A [ ] the ones between fetch 1 and fetch 2

B [ ] between fetch 2 and decode

C [ ] between decode and execute

D [ ] between execute and memory

E [✓] between memory and writeback

Question 15 [2 pt]: Which of the following is/are differences between a processor that starts multiple instructions per cycle and executes them out of order and an in-order pipelined processor (like the one we built in the homeworks)? **Select all that apply.** drop question because we didn't specify the direction of comparison and because we forget to mark it as select-all-that-apply

A the number of data dependency increases
B the register file is likely to have more read and write ports
C resolving branch mispredictions is likely to require more cycles
D the number of hazards increases

Answer: B C D

Question 16 [2 pt]: Consider the code the below. How many version of the of the register `%r10` will be generated for these instructions by an out-of-order processor that resolve hazards using register renaming?

```
mrmovq (%rsp), %r10
addq %r10, %r10
addq %r10, %r11
addq %r12, %r10
rmmovq %r11, (%r10)
```

Answer: 3

Question 17 [2 pt]: Assume that the code below is run on a 5 stage pipeline, with forwarding.

```
    xorq %r10, %r9
    subq %r10, %r11
    andq %r11, %r11
```

It is possible to solve the data dependences above using forwarding and no stalling. To do this, which of the following forwarding operations would be useful? dropped; for variant K, this is done outside of TPEGS **Place a ✓in each box corresponding to a correct answer and leave other boxes blank.**

A ☐ `%r11` from the end of the `xorq`'s execute stage to the end of the `andq`'s decode stage met to be from subq's, but isn't...

B ☐ `%r10` from the end of the `xorq`'s memory stage to the end of the `subq`'s decode stage

C ☐ `%r11` from the end of the `xorq`'s memory stage to the end of the `andq`'s decode stage meant to be from subq's, but isn't...

D ☐ `%r10` from the end of the `xorq`'s execute stage to the end of the `subq`'s decode stage

Question 18 [2 pt]: Consider a 4-way set associative 32KB cache with 64 byte cache blocks, an LRU replacement policy, a write-back policy, and a write-allocate policy. On a system with 64-bit addresses, how many bits of an address is used for the tag?

Answer: 51

Question 19 [2 pt]: Consider a processor with an 8 cycle non-pipelined divider and 3 cycle pipelined multiplier. After values are produced by an execution unit (like the divider or multiplier), they can be used as inputs to an execution unit in the following cycle. What is the **minimum** number of cycles required to execute $(a \times (b \times c)) \div (d \times e)$ using these execution units?

Answer: 14 (half-credit for 15)

Question 20 [2 pt]: Suppose we have L1 cache with a 2 cycle hit time and 90% hit rate, and an L2 cache with a 80% hit rate (assume all L2 accesses come via this L1 cache), and a main memory with a 100 cycle access time. What hit time, in cycles, does the L2 cache need into have to result in an overall average memory access time of 5 cycles for the L1 cache?

Answer: 10

Recall the formula for average memory access time: hit time + miss penalty × miss rate. $5 = 2 + 0.1 \cdot (x + 0.2(100))$

Pledge:

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

Your signature here