CS 3330 Exam 2 Fall 2017

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.

. . . .

Information for questions 1–2

For these questions consider a pipelined processor where the execute stage is split into three stages in order to support instructions with complex ALU operations. The resulting stages in this processor are:

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

The new split execute stages require the values for the ALU operation near the start of the execute 1 stage and only produces results near the end of end of the execute 3 stage, regardless of the complexity of the ALU operation. The other stages work as in the five-stage processor described in lecture.

Question 1 [2 pt]: (see above) To execute the following assembly snippet on this processor with **minimum stalling**:

```
addq %rax, %rbx
mrmovq 4(%rax), %r13
subq %rbx, %r13
```

the processor needs to stall for some number of cycles **in addition to forwarding** register values. How many cycles of stalling are required? Write your answer as a base-10 number.

Answer: 3

Question 2 [2 pt]: (see above) To execute the following assembly snippet on this processor without stalling:

```
addq %rax, %rbx
xorq %rcx, %rdx
rrmovq %r8, %r9
rmmovq %r11, 4(%r12)
subq %rbx, %r13
```

the processor needs to forward the value of `%rbx` from `addq` to `subq`. Which forwarding path can the processor use to do this?

A the end of `addq`'s execute 2 stage to the end of `subq`'s decode stage
B the end of `addq`'s memory stage to the end of `subq`'s execute 1 stage
C the end of `addq`'s execute 3 stage to the end of `subq`'s execute 1 stage
D the end of `addq`'s execute 3 stage to the end of `subq`'s decode stage
E the end of `addq`'s memory stage to the end of the `subq`'s decode stage
F the end of `addq`'s writeback stage to the end of `subq`'s execute 1 stage
G none of the above

Answer: E

Information for questions 3–4
Consider a 4 KB (2^{12} byte) 2-way set-associative write-back, write-allocate cache with 256-byte (2^8 byte) cache blocks, an LRU replacement policy.

Question 3 [2 pt]: (see above) How many sets does this cache have? Write your answer as a base-10 number.

Answer: 8

Question 4 [2 pt]: (see above) When accessing this cache, which of the following addresses will map to the **same cache set** as `0x12345`? **Place a ✓ in each box that will map to the same set. Leave other boxes blank.**

A [✓] `0x23BFF`

B [ ] `0x01234`

C [ ] `0x12000`

D [✓] `0x123FF`

Question 5 [2 pt]: Suppose a program reads every byte of a 2MB array starting with the lowest address and proceeding in order to the highest address, then repeats this process 1000 times. On which type of data cache will the program experience the **fewest** cache misses on average? (Assume memory accesses other than those to the array are negilible and that 1MB represents 2^{20} bytes.)

A a 1MB 4-way set associative cache with a random replacement policy and 64B cache blocks
B a 1.75MB fully associative cache with an FIFO (first-in, first-out) replacement policy and 64B cache blocks
C a 0.5MB direct-mapped cache with 32B cache blocks
D a 1MB 8-way set associative cache with an LRU (least recently used) replacement policy and 64B cache blocks

Answer: A

Information for questions 6–7

Suppose a program reads a single byte at each of the following addresses, in the following order:

- `0x06` DM: M (set 1); 2W: M (set 1)
- `0x07` DM: H (set 1); 2W: H (set 1)
- `0x16` DM: M (set 1); 2W: M (set 1)
- `0x00` DM: M (set 0); 2W: M (set 0)
- `0x17` DM: H (set 1); 2W: H (set 1)
- `0x03` DM: H (set 0); 2W: M (set 1)
- `0x07` DM: M (set 1; conflict with 0x14-7); 2W: M (conflict with 0x02-3 and 0x16-7)

Question 6 [2 pt]: (see above) Consider a 16 byte direct-mapped cache with a 4 byte block size. Assume the cache is initially empty. How many of these accesses will be hits? Write your answer as a base-10 number.

Answer: 3

Question 7 [2 pt]: (see above) Consider a 8 byte 2-way set asssociative cache with a 2 byte block size and an LRU replacement policy. Assume the cache is initially empty. How many of these accesses will be hits? Write your answer as a base-10 number.

Answer: 2

Question 8 [2 pt]: Suppose we have two processor designs, with different numbers of pipeline stages. On the first processor 10% of the instructions in a benchmark program stall for exactly 1 cycle and 5% stall for exactly 2 cycles. The second processor has more pipeline stages, so 40% of the instructions in the benchmark program stall for one more cycle than on the first processor, but it has half the cycle time.

If the benchmark program took 10 seconds to run on the first processor, then how long did take on the second processor? accepted all answers (outside of TPEGS) due to ambiguity

A more than 10 seconds
B more than 9 and less than or equal to 10 seconds if 40% stall for 3 cycles
C more than 8 and less than or equal to 9 seconds
D more than 7 and less than or equal to 8 seconds
E more than 6 and less than or equal to 7 seconds CPI(first) = 1.2, CPI(second) = 1.6
F 6 or less seconds if 40% of 15% stall more

Answer: B E

Information for questions 9–10

Consider the following C code:

```
unsigned char array[1024 * 1024];
/* ... */
int sum = 0;
for (int x = 0; x < 10; ++x) {
    for (int i = 0; i < 16; ++i) {
        sum += array[i * 128] * array[i * 128 + 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 the `array`;
- `unsigned char`s are 1 byte;
- the cache is empty upon entry to the outer for loop above

Note that $1024/8 = 128$.

Question 9 [2 pt]: (see above) If the above nested for loops are run on a system with a **1KB (1024 byte) direct-mappped cache** with **1-byte cache blocks**, how many data cache misses will it experience?

A 16
B 24
C 31
D 160
E 175 160 for `array[i*128]`, 15 for `array[i*128+i]` (array[0*128+0] is hit because `array[0*128]` just accessed; others only thing in cache set)
F 240
G 320
H none of the above

Answer: E

Question 10 [2 pt]: (see above) If the above nested for loops are run on a system with an **4KB (1024 × 4 byte) 16-way set associative cache** with **64-byte cache blocks** and an LRU replacement policy, how many data cache misses will it experience?

A 31
B 240
C 320
D 16 array[i*128] and array[i*128+i] always in same cache block; since there are only 16 cache blocks in total, they can all fit in the cache even if they're all in one set
E 175
F 160
G 24
H none of the above

Answer: D

Information for questions 11–13

Suppose we are running the following assembly snippet in the five-stage pipelined processor with forwarding and branch prediction discussed in the lecture and in the textbook:

```
    xorq %rax, %rax
    jne after_add
    addq %rax, %rsp
after_add:
    subq %rax, %rsp
    andq %rsp, %rax
    rmmovq %rsp, 0(%rax)
    popq %rax
```

Recall that this processor predicts all branches as taken and computes the actual result of branches by the end of the execute stage of the conditional jump instruction and fetches the corrected next instruction during the conditional jump's memory stage. **The conditional jump is not taken.** the rmmovq instruction was mistakenly an mrmovq when the exam was given (here and below); this key corrects this

Question 11 [3.0 pt]: (see above) In order to execute this assembly snippet correctly, which of the following forwarding operations will the processor need to perform? **Place a ✓ next to each correct answer. Leave all other boxes blank.**

A ☐ `%rax` from `xorq` to `addq` addq's decode after xorq's writeback

B ☑ `%rsp` from `addq` to `subq` ✓

C ☑ `%rax` from `andq` to `rmmovq` ✓ read for address

D ☑ `%rsp` from `subq` to `rmmovq` ✓

E ☑ `%rsp` from `subq` to `popq` ✓

F ☐ `%rax` from `andq` to `popq` popq only writes %rax

Question 12 [2.0 pt]: (see above) When the `andq` instruction is in the fetch stage, what pipeline stages will be running a pipeline "bubble" (nop inserted to handle a pipeline hazard)? **Place a ✓ in each box that will have a pipeline bubble. Leave all other boxes blank.** accept this or nothing checked for full credit (andq is fetched twice); half-credit for any two consecutive stages checked

A ☐ decode

B ☐ execute

C ☑ memory ✓

D ☑ writeback ✓

Question 13 [2 pt]: (see above) When the `subq` instruction is in the writeback stage, the `rmmovq` instruction is in what stage?

A fetch
B decode
C execute
D memory
E writeback
F none of the above; it is not in the pipeline

Answer: C

Information for questions 14–16
Consider the following two pieces C code:

```
/* Version A */
for (int i = 0; i < N; ++i)
    for (int j = 0; j < N; ++j)
        A[i * N + j] += B[i + j * 10] * C[j];
```

and:

```
/* Version B */
for (int j = 0; j < N; ++j)
    for (int i = 0; i < N; ++i)
        A[i * N + j] += B[i + j * 10] * C[j];
```

Assume that N is a large integer constant and that A, B, and C are large arrays of 4-byte ints. Assume that all values except the accesses to A, B and C are stored in registers.

Question 14 [1.0 pt]: (see above) Which version has better spatial locality in its accesses to B?

A version A
B version B
C they are about the same

Answer: B

Question 15 [1.0 pt]: (see above) Which version has better temporal locality in its accesses to C?

A version A
B version B
C they are about the same

Answer: B

Question 16 [1.0 pt]: (see above) Which version has better temporal locality in its accesses to A?

A version A
B version B
C they are about the same elements of A accessed at most once

Answer: C

Question 17 [2 pt]: Which of the following statements about loop unrolling are true? **Put a ✓ next to each box that is true. Leave the other boxes blank.**

A ✓ loop unrolling is likely to increase the size of an executable

B loop unrolling improves the temporal locality of instruction accesses more instructions in memory, each accessed less frequently

C ✓ as long as a loop has enough iterations, loop unrolling will decrease the number of instructions executed

D loop unrolling is not practical if the loop body manipulates two pointers that may alias would matter if loop condition/iteration variable changes, but can duplicate (and not reorder) loop body code that might involve pointer aliasing

Information for questions 18–20

Suppose the components of our Y86 processor take the following amounts of time to process their inputs and either produce a corresponding output or (in the case of register file or data memory writes) be ready to store a value if a rising clock edge happens.

component	time required
instruction memory	250 picoseconds
register file reading	200 picoseconds
register file writing	200 picoseconds
ALU	350 picoseconds
data memory (reading or writing)	300 picoseconds
PC increment	30 picoseconds

In addition, assume the **register delay for any pipeline registers** is **25 picoseconds**. Assume that other components of the processor take a negligible amount of time.

Question 18 [2 pt]: (see above) If we use these components to build a five-stage pipelined Y86 processor, like the one described in lecture and our textbook, what is the minimum cycle time this processor can have? Write your answer as a base-10 number of picoseconds.

Answer: 375 (half-credit for 350 or 1875 or 1750)

Question 19 [2 pt]: (see above) If we use these components to build a single-cycle Y86 processor, what is the minimum cycle time this processor can have? Do not include any register delay for the PC register. Write your answer as a base-10 number of picoseconds.

Answer: 1300 (half-credit for 1330 or 1325 or 1355)

Question 20 [2 pt]: (see above) Suppose we split the ALU into two halves, each of which takes 175 picoseconds (half of the original 350 picoseconds). Using this, we split the execute stage into two stages to modify our five-stage pipelined into a six-stage pipelined Y86 processor. By how much will this reduce the cycle time of our Y86 processor? Write your answer as a base-10 number of picoseconds.

Answer: 50 (half-credit for 250 or 325 or 75)

. . . .

Pledge:

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

____

Your signature here