CS 3330 Exam 1 Spring 2017

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

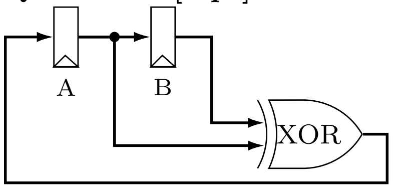
Letters go in the boxes unless otherwise specified (e.g., for **C** 8 write "C" not "8").
Write Letters clearly: if we are unsure of what you wrote you will get a zero on that problem.
Bubble and Pledge the exam or you will lose points.
Assume unless otherwise specified:

- 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.

. . . .

Question 1 [2 pt]:



Consider the circuit above with contains two one-bit registers labelled A and B (represented by rectangles with a triangular notch at the bottom), and combinatorial logic that performs a 1-bit XOR operation (labelled XOR) on the outputs of the two registers. If A and B initially output the value 1, what value will A and B output after 2 rising edges of the clock signal?

A A will output 1; B will output 1
B A will output 0; B will output 0
C A will output 1; B will output 0
D A will output 0; B will output 1

Answer: C

Information for questions 2–4
In the single-cycle Y86-64 processor design discussed in our book and lecture, the data memory has an *address* input and a *data* input.

Question 2 [2 pt]: (see above) When this processor is executing `mrmovq 4(%rax), %rbx` instruction, what is the *address* input to the data memory equal to?

A part of the output of the instruction memory
B the output of some combinatorial logic that performs arithmetic on an output of the register file
C the next value of the program counter
D irrelevant, since the data memory is not used by this instruction
E one of the outputs of the register file
F none of the above

Answer: B

Question 3 [2 pt]: (see above) When this processor is executing `popq %rax` instruction, what is the *address* input to the data memory equal to?

A the `rA` or `rB` field from the instruction
B one of the outputs of the register file
C the output of some combinatorial logic that performs arithmetic on an output of the register file
D the next value of the program counter
E irrelevant, since the data memory is not used by this instruction
F none of the above

Answer: B

Question 4 [2 pt]: (see above) When this processor is executing a `pushq %rax` instruction, what is the *value* input to the data memory equal to?

A one of the outputs of the register file
B irrelevant, since the data memory is not used by this instruction
C the output of some combinatorial logic that performs arithmetic on an output of the register file
D the next value of the program counter
E part of the output of the instruction memory
F none of the above

Answer: A

Question 5 [3 pt]: Which of the following are likely to be benefits of the Reduced Instruction Set Computer (RISC) design philosophy over a Complex Instruction Set Computer (CISC) design philosophy? **Select all that apply.**

A more specialized support for particular target applications;
B lower power consumption for a processor;
C lower time-to-market for a processor;
D making it simpler to write a functioning compiler;
E reducing the size in memory of a typical program;
F not needing to implement instructions that perform many memory accesses;

Answer: B C F

Question 6 [2 pt]: In the Y86-64 ISA, which instruction(s) change the program counter? **Select all that apply.**

- **A** `irmovq`
- **B** `call`
- **C** `addq`
- **D** all of the instructions
- **E** none of the instructions

Answer: D or ABCD or ABC

Question 7 [2 pt]: Indicate which of the following design choices in a processor is a feature of the ISA. **Select all that apply.**

- **A** The frequency of the processor
- **B** The number of ports in memories
- **C** The number of registers
- **D** The latency of the ADD instruction

Answer: C

Question 8 [2 pt]: Suppose `x` is a `long *` variable which is stored in the register `%rdi`. Which of the following X86-64 assembly snippets is equivalent to the C code `*x += 1;`?

- **A** `addq $1, 0(%rdi)`
- **B** `addq $1, 8(%rdi)`
- **C** `addq $1, 1(%rdi)`
- **D** `movq %rdi, 1(%rdi)`
- **E** `addq $1, %rdi`

Answer: A

Question 9 [2 pt]: Consider the following Y86-64 program and its corresponding machine code.

```
0x000: 30 f2 01 00 00 00 00 00 00 00 | irmovq $1, %rdx
0x00a: 50 02 00 00 00 00 00 00 00 00 | mrmovq 0(%rdx), %rax
0x014: 00                            | halt
```

(The value before the colon is the memory address of the instruction listed after the `|`, the hexadecimal values after the colon are the bytes of the instruction, in the order they appear in memory.)

After this assembly program runs, what is the value of `%rax`?

- **A** 0x5002
- **B** 0x30F2010000000000 (ends with 10 zeroes)
- **C** 0xF201000000000000 (ends with 12 zeroes)
- **D** 0x0
- **E** 0x1F230
- **F** 0x250
- **G** 0xF201
- **H** 0x1
- **I** 0x1F2
- **J** 0x30F201

Answer: I

Question 10 [2 pt]: Suppose you want to add a new instruction to the Y86-64 ISA. The instruction is called MADD (Multiply and Accumulate).

```
MADD rA, rB:
     R[rB] <- (R[rA]*R[rB])+R[rA]
```

Which statements are true for a **single-cycle processor**? **Select all that apply.**

A MADD will increase the time required for 'execute' stage
B MADD will increase the number of register accesses compared to using separate instructions for multiply and add
C MADD will increase the number of memory accesses
D MADD will be faster than using two separate instructions for multiply and add

Answer: A D

Question 11 [2 pt]: Suppose one added an `immovq $Immed, D(rB)` instruction to Y86-64, which moves a 64-bit immediate value to a memory location. Which of the following statements about adding this instruction in our single-cycle processor design are true? **Select all that apply.**

A it would require adding an additional input to the MUX in front of the data memory address input;
B it would require adding an additional input to the MUX in front of the data memory value input;
C it would require adding an additional port to the data memory;
D the instruction could have a 10-byte instruction encoding similar to `rmmovq`'s

Answer: B

Question 12 [2 pt]: What is $\log_2(64M)$ (where M represents the power of two equal closest to one million)?

Answer: 26

Question 13 [6 pt]: If `x` and `y` are **positive** signed integers between 1 and 1000, then which of the following are **always** true? **Select all that apply.**

A [ ] `(x & y) | ((~x) & (~y)) == x ^ y`

B [ ] `x ^ (x | y) >= y`

C [✓] `(x >> 2) <= (x >> 1)`

D [ ] `(x >> 2) << 2 == x`

E [ ] `x ^ (x & y) >= y`

F [✓] `(x & y) <= (x | y)`

Question 14 [2 pt]: Recall that Y86-64 has the flags `SF` (sign flag; 1 if negative) and `ZF` (zero flag; 1 if zero). After the Y86-64 assembly snippet `xorq %rax, %rbx` runs, which are possible values of these flags? **Select all that apply.**

A `SF = 1` and `ZF = 1`
B `SF = 0` and `ZF = 1`
C `SF = 1` and `ZF = 0`
D `SF = 0` and `ZF = 0`

Answer: B C D

Question 15 [2 pt]: Suppose you are designing an ISA for sensors which will be deployed in the deep forest of Amazon. The sensors need to be power efficient and must not use too much power for decoding instructions. The goal is to design an ISA that will make the fetch and decode stages as simple as possible. Which of the following design choices are appropriate? **Select all that apply.**

A A large number of addressing modes
B Variable-length instructions
C A large number of registers
D Uniform decode

Answer: D

Question 16 [2 pt]: In the single-cycle processor described in lecture and our textbook, when is the earliest that the output by the program counter register can change while the processor is executing a `andq` instruction?

A immediately after the `andq` instruction sends the result of the AND to the register file
B at the next rising edge of the clock signal
C as soon as the new value is computed
D immediately after the fields of the instruction are extracted

Answer: B

Question 17 [2 pt]: Consider the following C code:

```
int array[3] = { 1, 2, 3 };
int *ptr = array;
ptr += 1;
*ptr += 1;
```

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

A unknown; the code invokes undefined behavior
B `{1, 3, 3}`
C `{2, 2, 3}`
D `{1, 2, 3}`
E `{257, 2, 3}`
F none of the above

Answer: B

Information for questions 18–20

For these questions assume the following declaration:

```
unsigned char *instr;
```

and that `instr` is a pointer to the **first byte** of a Y86-64 instruction.

Recall that in a Y86-64 instruction:

- the `opcode` (also known as `icode`) field an is in the most significant 4 bits of the first byte of an instruction;
- the `rA` and `rB` fields (when present) are in the most significant and least significant 4 bits, respectively, of the second byte of the instruction.

In the questions below, you may assume that all bitshifts are logical shifts.

Question 18 [2 pt]: (see above) Which of the following is valid C which results in a true value if the `rA` and `rB` fields of the instruction are equal? (Assume the fields are present in the instruction.)

A `(instr[1] | (instr[1] >> 4)) == 0`
B `((instr[1] >> 4) - instr[1]) == 0`
C `instr[3] == instr[4]`
D `(instr[1] & 0xF) == (instr[1] & 0xF0)`
E `((instr[1] ^ (instr[1] << 4)) & (0xF << 4)) == 0`
F none of the above

Answer: E

Question 19 [2 pt]: (see above) Which of the following is valid C code to extract the opcode of the instruction?

A `(instr[0] & 0xF0) >> 4`
B `(instr[0] << 4) ^ 0xF`
C `(instr[0] >> 4) | 0xF`
D `(instr[3]<<3)|(instr[2]<<2)|(instr[1]<<1)|instr[1]`
E none of the above

Answer: A

Question 20 [2 pt]: (see above) Which of the following is valid C code to extract the `rB` field of the instruction? (Assume the field is present in the instruction.)

A `instr[4]`
B `(instr[0] & 0xF00) >> 8`
C `(instr[1] << 4) & 0xF`
D `instr[1] & 0xF`
E none of the above

Answer: D

Question 21 [2 pt]: Consider the following C function:

```
long foo(long x) {
    while (x < 10000) {
        x = x + x;
    }
    return x;
}
```

Which of the following X86-64 assembly snippets is equivalent to this C code? (Recall that, in the X86-64 calling convention, the first argument uses `%rdi` and the return value uses `%rax`. Also, `cmpq A, B` sets condition codes based on the subtraction B - A.)

A
```
 foo: popq %rax
loop: cmpq $10000, %rax
      jge end
      addq %rax, %rax
      jmp loop
 end: ret
```

B
```
 foo: addq %rdi, %rdi
      cmpq $10000, %rdi
      jl foo
 end: movq %rdi, %rax
      ret
```

C
```
 foo: cmpq $9999, %rdi
      jg end
      addq %rdi, %rdi
      jmp foo
 end: movq %rdi, %rax
      ret
```

D none of the above

Answer: C

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.