CS 224 Winter 2023
Sections 1, 2, 3, & 13
Archibald, Judd, & Clement

Name: ______________________________
Section: ______________________________

CS 224 Midterm 2
Sections 001, 002, 003, & 013

Instructions:

- Treat it like an exam in the testing center
- No time limit
- Open book (any of the versions of the text for the course) and Y86-64 quick reference guide
- One page of hand-written notes (both sides) allowed
- Closed everything else
- Enter your answers in the corresponding Canvas quiz

1 (23):	
2 (5):	
3 (5):	
4 (5):	
5 (15):	
6 (15):	
7 (10):	
8 (10):	
EC (8):	
TOTAL (88):	

Problem 1. (23 points):

Consider the following Y86-64 program. The object code is shown on the left, and the assembly code on the right. Your task is to determine what the program state will be after this program is run to completion.

```
0x0000: 30f41001000000000000   | irmovq stack, %rsp
0x000a: 30f00400000000000000   | irmovq $4, %rax
0x0014: 30f36800000000000000   | irmovq data0, %rbx
0x001e: 30f10100000000000000   | irmovq $1, %rcx
0x0028: 50230000000000000000   | mrmovq (%rbx), %rdx
0x0032: 6000                   | addq %rax, %rax
0x0034:                        | L1:
0x0034: 6222                   | andq %rdx, %rdx
0x0036: 200a                   | rrmovq %rax, %r10
0x0038: 735a00000000000000     | je L2
0x0041: 50730800000000000000   | mrmovq 8(%rbx), %rdi
0x004b: 607a                   | addq %rdi, %r10
0x004d: 6003                   | addq %rax, %rbx
0x004f: 6112                   | subq %rcx, %rdx
0x0051: 743400000000000000     | jne L1
0x005a:                        | L2:
0x005a: 40a3f0ffffffffffffff   | rmmovq %r10, -16(%rbx)
0x0064: a00f                   | pushq %rax
0x0066: 00                     | halt
                               |
0x0067:                        | .align 8
0x0068:                        | data0:
0x0068: 02                     |     .quad 0x2
0x0070:                        | data1:
0x0070: 02                     |     .quad 0x2
0x0078:                        | data2:
0x0078: 04                     |     .quad 0x4
                               |
0x0080:                        | .pos 0x110
0x0110:                        | stack:
```

Please fill in the following details about the program state **after the program has run to completion**.

A. Memory - Indicate the value of the quadword at each of the following memory locations:

data0	0x0c	data1	0x02	data2	0x04

B. Condition Codes - Indicate the value of the three condition codes:

SF	0	ZF	1	OF	0

C. Registers - Indicate the value in each of the following registers:

%rax	0x008	%rdx	0x000	%rcx	0x001
%rsi	0x000	%rdi	0x004	%rbx	0x078
%rsp	0x108	%r8	0x000	%r10	0x00c

D. What is the value of the program counter PC?

0x0067

Problem 2. (5 points):

Consider the following C functions and assembly code where ap is in register %rdi and bp is in register %rsi, and the caller looks in %rax for the return value:

```
int fun0(long int *ap, long int *bp)
{
    long int a = *ap;
    long int b = *bp;
    *ap = a+b;
    return a+b;
}

int fun1(long int *ap, long int *bp)
{
    long int b = *bp;
    *bp = *ap;
    *ap = b;
    return b+b;
}

int fun2(long int *ap, long int *bp)
{
    long int a = *ap;
    long int b = *bp;
    *ap = b;
    *bp = a;
    return a+b;
}
```

```
mrmovq    (%rsi), %rax
mrmovq    (%rdi), %rdx
rmmovq   %rax, (%rdi)
rmmovq    %rdx, (%rsi)
addq      %rdx, %rax
ret
```

Which of the three functions is implemented in the assembly code shown?

Problem 3. (5 points):

Consider the following C functions and assembly code where `x` is in register `%rdi`, `y` is in register `%rsi`, `z` is in register `%rdx`, and the caller looks in `%rax` for the return value:

```
long fun0(long x, long y, long z) {
  long val = 0;
  if (x < y)  {
    if (x < z) {
      val = x;
    } else {
      val = y;
    }
  } else if (z < y) {
      val = z;
  }
  return val;
}

long fun1(long x, long y, long z) {
  long val = 0;
  if (x <= y)  {
    if (x <= z) {
      val = x;
    } else {
      val = y;
    }
  } else if (z <= y) {
      val = z;
  }
  return val;
}

long fun2(long x, long y, long z) {
  long val = 0;
  if (x > y)  {
    if (x > z) {
      val = x;
    } else {
      val = y;
    }
  } else if (z > y) {
      val = z;
  }
  return val;
}
```

```
  rrmovq   %rdi, %r10
  subq     %rsi, %r10
  jge      L2
  rrmovq   %rdi, %r10
  subq     %rdx, %r10
  rrmovq   %rsi, %rax
  cmovl    %rdi, %rax
  ret
L2:
  xorq     %rax, %rax
  rrmovq   %rsi, %r10
  subq     %rdx, %r10
  cmovg    %rdx, %rax
  ret
```

Which of the functions compiled into the assembly code shown?

Problem 4. (5 points):

Consider the following y86-64 code for `long int loopyX(long int x, long int y)`:

```
  irmovq  $0x1, %r10
  rrmovq  %rsi, %rcx
  subq    %rdi, %rcx
  jle     L4
  irmovq  0x0, %rax
L3:
  addq    %rsi, %rax
  subq    %r10, %rsi
  rrmovq  %rdi, %rcx
  subq    %rsi, %rcx
  jl      L3
  ret
L4:
  irmovq  0x0, %rax
  ret
```

Register `%rdi` holds parameter `x` and register `%rsi` holds parameter `y`. The return value, `result`, is left in `%rax` at return. Which of the following C functions compiled into the given assembly code?

```
long int loopy0(long int x, long int y) {
  long int i, result = 0;
  for( i = 0 ; i < x ; i++ ){ result += x; }
  return result;
}
```

```
long int loopy1(long int x, long int y) {
  long int i, result = 0;
  for( i = y ; i > x ; i-- ){ result += i; }
  return result;
}
```

```
long int loopy2(long int x, long int y) {
  long int i, result = 0;
  for( i = y ; i > 0 ; i-- ){ result += 1; }
  return result;
}
```

Problem 5. (15 points):

Consider the following sequence of bytes, shown in memory starting at address `0x000`. Your task is to determine the Y86-64 instruction sequence that is encoded in these bytes. Memory is displayed as in the Y86-64 online simulator, that is, memory addresses increase as we move to the right and down. The address of the first byte shown on each row is given to the left of the row. Each cell of the memory table contains a single byte.

Address	**Memory**							
0x000	30	f3	04	00	00	00	00	00
0x008	00	00	20	36	30	f0	ff	ff
0x010	ff	ff	ff	ff	ff	ff	10	60
0x018	36	62	60	10	73	16	00	00
0x020	00	00	00	00	00	00		

A. What is the sequence of Y86-64 instructions that is encoded in these bytes?

```
    irmovq 0x4, %rbx
    rrmovq %rbx, %rsi
    irmovq 0xffffffffffffffff, %rax
L1:
    nop
    addq %rbx, %rsi
    andq %rsi, %rax
    nop
    je L1
    halt
```

B. After this sequence of instructions has been run to completion, what is the value in register `%rax`?

0x8

Problem 6. (15 points):

Consider the following sequence of Y86-64 instructions. You will be determining the byte-level encoding for these instructions, assuming that they are placed in memory starting at memory location `0x000`.

```
irmovq $15, %rax
rrmovq %rax, %rbx
addq %rax, %rbx
andq %rax, %rbx
halt
```

Please fill in following memory table, placing the encoded byte in each memory cell. Leave any cells blank that are not used by the given instruction sequence. Memory is displayed as in the Y86-64 online simulator, that is, memory addresses increase as we move to the right and down. The address of the first byte shown on each row is given to the left of the row. Each cell of the memory table should contain a single byte (if used in the encoding), or be blank (if not used in the encoding).

Address	**Memory**							
0x000	30	f0	0f	00	00	00	00	00
0x008	00	00	20	03	60	03	62	03
0x010	00	00	00	00	00	00	00	00

Problem 7. (10 points):

Create a computation table similar to those used to implement **Lab 3: Y86-64** that defines what needs to happen in each stage of the sequential architecture to implement a `testq rA, rB` instruction that bitwise ands (`&`) `R[rA]` and `R[rB]` but does **not** store the result of the bitwise and (`&`) in `R[rB]`; it only updates the condition codes.

```
rrmovq  %rdi, %rcx
andq    %rsi, %rcx
jle     L4
```

With the new instruction, the above code is simplified as below.

```
testq  %rsi, %rdi
jle    L4
```

The byte format for `testq rA, rB`, where each box is a nibble, is below.

C	2	**rA**	**rB**

Stage	`testq rA, rB`
Fetch	icode:ifun <- M1[PC] rA:rB <- M1[PC+1] valP <-PC + 2
Decode	valA <- R[rA] valB <- R[rB]
Execute	valE <- valB & valA set CC
Memory	
Write back	
PC update	PC <- valP

Problem 8. (10 points):

Create a computation table similar to those used to implement **Lab 3: Y86-64** that defines what needs to happen in each stage of the sequential architecture to implement a `leaq D(rB), rA` instruction that loads the effective address given by `D(rB)` (e.g., computes the address `D + R[rB]` and puts that address in `R[rA]`, but doesn't set the condition codes).

```
irmovq  0xa, %rax
rrmovq  %rdi, %rdx
addq    %rax, %rdx
```

With the new instruction, the above code is simplified as below.

```
leaq 0xa(%rdi), %rdx
```

The byte format for the `leaq D(rB), rA`, where each small box is a nibble and the long box is 8 bytes, is below.

E	0	**rA**	**rB**	D

Stage	`leaq D(rB), rA`
Fetch	icode:ifun <- M1[PC] rA:rB <- M1[PC + 1] valC <- M8[PC + 2] valP <- PC + 10
Decode	valB <- R[rB]
Execute	valE <- valB + valC
Memory	—
Write back	R[rA] <- valE
PC update	PC <- valP

Problem 9. (8 points):

Extra Credit

Consider a C function having the general structure:

```
long int rfun(long int x, long int y) {
  long int nx = _________________________;
  long int ny = _________________________;
  long int rv = 0;

  if (_________________________) {
    return _________________________;
  }

  rv = rfun(nx, ny);
  return _________________________;
}
```

This C code yields the following y86-64 machine code:

```
rfun:
  irmovq   $0, %rax        set rv to 0
  rrmovq   %rdi, %rdx      move x to rdx
  andq     %rdx, %rdx
  jg       L8               if x <= 0, return
  ret
L8:
  pushq    %rbx
  rrmovq   %rsi, %rbx       rbx = y
  irmovq   $-1, %rdx
  addq     %rdx, %rsi       y = y-2
  addq     %rdx, %rsi
  addq     %rdx, %rdi       x = x-1
  call     rfun
  addq     %rbx, %rax       rv = rv + y
  popq     %rbx
  ret
```

The questions on the next page are about these pieces of code.

A. What value does `rfun` store in the callee-saved register `%rbx`?

the value of the second argument y

moved into %rbx at the instruction "rrmovq %rsi, %rbx"

B. Fill in the missing expressions in the C code shown above.

```
long int nx = x - 1;

long int ny = y - 2;

long int rv = 0;

if (nx <= 0) {
    return y;
}

rv = rfun(nx, ny);

return rv + y;
```

C. What does `rfun(3,6)` return in `%rax`?

12