CS 241: Computer Organization and Assembly Language
Practice Final Exam

Do not open until instructed to do so.

Name: ____________________

Sooner or later the world breaks everyone, and afterward many are strong in the broken places.
~Ernest Hemingway, *A Farewell to Arms*

Every problem is marked with a ▶ . When you see this symbol, it means that's a question which you can — and should — answer.

For grader use:

Score: ______________________________

Syscalls

`0`	`sys_read`
`1`	`sys_write`
`60`	`sys_exit`

Arguments in: `rdi, rsi, rdx, r10, r8, r9` in that order

Return value in: `rax`

Clobbers: `rcx, r11`

Common syscalls

	1	Output	Addr.	Length
`write`	`rax`	`rdi`	`rsi`	`rdx`

	0	Input	Addr	Length
`read`	`rax`	`rdi`	`rsi`	`rdx`

	60	Exit code
`exit`	`rax`	`rdi`

C-style functions

```
func:
  push rbp
  mov rbp, rsp

  ...

  pop rbp
  ret
```

Arguments in: `rdi, rsi, rdx, rcx, r8, r9` in that order

Return value in: `rax`

Callee-saved regs.: `rbx, rbp, r12-r15`

Clobbers: `rax, r10, r11`, argument registers

`rsp` must be a multiple of 16, plus 8, before any `call`. `rsp` is a multiple of 16 on function entry.

Memory operands

`size [displacement + base + m * offset]`

`size` byte, word, dword, etc.

`displacement` Constant address of array

`base` Array base register

`m` 1, 2, 4, or 8

`offset` Array offset register

Instructions

`mov rm, rmi`	Move
`xchng rm, rm`	Swap
`lea r, m`	Load Effective Address
`xor r, r`	Set *r* to `0`
`add rm, rmi`	Addition
`sub rm, rmi`	Subtraction
`mul rmi`	Unsigned multiply (by/into `rax`)
`div rmi`	Unsigned divide (into `rax`)
`imul rmi`	Signed multiply
`idiv rmi`	Signed divide
`cmp rm, rmi`	Compare, update flags
`text rm, rmi`	Test, update flags
`jmp target`	Jump to target
`j`*`CC`* `target`	Jump if condition *CC*
`loop target`	Decrement `rcx`, jump if not `0`
`call func`	Push `rip`, jump to `func`
`ret`	Pop `rip` and jump to
`push rm`	Push onto stack
`pop rm`	Pop from stack

`r`: register, `m`: memory operand, `i`: immediate

Condition codes

CC	Meaning
a	Unsigned $>$
ae	Unsigned $\geq$
b	Unsigned $<$
be	Unsigned $\leq$
g	Signed $>$
ge	Signed $\geq$
l	Signed $<$
le	Signed $\leq$
e	$=$
ne	$\neq$
s,c,z,...	If flag is set

5 points each

► Perform the following binary addition:
10110100 + 00111111
Show your work (all carries).

```
      1 1 1 1
    1 0 1 1 0 1 0 0
 +  0 0 1 1 1 1 1 1
---------------------
    1 1 1 1 0 0 1 1
```

► Suppose we want to swap the (byte) values in the registers al and ah. Write assembly code to do the swap.

You can do this with bswap (byte swap), but also manually, using another register:

```
mov bl, al
mov al, ah
mov ah, bl
```

► Perform the addition 01110100 + 10111111, show your work, write the final sum, as well as the state of the flags after the addition is complete.

```
       1 1 1 1 1
       0 1 1 1 0 1 0 0
   +   1 0 1 1 1 1 1 1
----------------------
 1     0 0 1 1 0 0 1 1
```

```
CF = 1

OF = 0

SF = 0

ZF = 0
```

▶ Perform the comparison `cmp 0b01110100, 0b10111111` and show the state of the flags after the comparison. (You can't actually do an immediate-immediate comparison, but just pretend.)

This is basically just 116 - 191 (or 116 - -65, signed) = `0b10110101` with an extra borrow.

```
CF = 1
```

```
OF = 1
```

```
SF = 1
```

```
ZF = 0
```

▶ Write assembly equivalent to the following C code:

```
int rax, rdi, rbx;

if(rax > 0)
  if(rdi < 10)
    rbx = 0;
```

```
cmp rax, 0
jle .done
cmp rdi, 10
jge .done
mov rbx, 0

.done:
```

Suppose we have the following structure definition:

```
struct S {
  int a;
  long b;
  char c;
  char* d;
};
```

▶ What is the size of this structure in bytes?

The structure will be laid out like this:

Offset	Member	Size (bytes)
0	a	4
4	padding	4
8	b	8
16	c	1
17	padding	7
24	d	8
32	Total size	

▶ What are the offsets, in bytes, of each of the structure members from the beginning of the structure?

```
S::a
```

```
S::b
```

```
S::c
```

```
S::d
```

▶ Write assembly code using string instructions to copy a 100 byte array from the address in `rax` to the address in `rbx`.

```
mov rcx, 100
mov rsi, rax
mov rdi, rbx
rep movsb
```

You could do this even faster by using larger movs:

```
mov rcx, 100 / 4
mov rsi, rax
mov rdi, rbx
rep movsd
```

`100` is not evenly divisible by 8, so if we used qwords we’d have to manually copy the remaining 4 bytes.

Coding problems

You should create a directory on the server called `~/cs241/final/` for your answers to these problems.

The first two problems will replace the equivalent section from the midterm, if you do better here than there. If you are happy with your grade on the midterm, *you do not need to do these problems*.

► Complete the following `syscall`-style function so that it will print out a triangle made of * characters. E.g., if the function's parameter in `rsi` is 5, it should print out

```
*
**
***
****
*****
```

```
section .data

newline:    db    10
star:       db    '*'

section .text

print_stars:
  mov r12, rsi    ; Save count

.while1:
  cmp r12, 0
  je .done

  ; Print r12 many stars
  mov r13, r12
  .do2:

    mov rax, 1
    mov rdi, 1
    mov rsi, star
    mov rdx, 1
    syscall

    dec r13
    cmp r13, 0
    jne .do2

  ; Print newline
  mov rax, 1
  mov rdi, 1
  mov rsi, newline
  mov rdx, 1
  syscall

  dec r12
  jmp .while1

.done:
ret
```

► Complete the following function so that it returns 1 if the qword array pointed to by `rdi` (array length in `rsi`) contains any duplicates, or `0` if it does not.

This is easier if you write a helper function, in fact, the `find` function from the next page.

```
has_duplicates:
  push rbp
  mov rbp, rsp

  mov r12, rdi           ; Array start addr
  lea r13, [8*rsi + rdi] ; Array end addr (+ 1)

.while:
  cmp r12, r13
  je .returnFalse

  mov rdi, r12
  mov rsi, r13
  sub rsi, r12 ; Length
  mov rdx, qword [r12]
  call find

  cmp rax, 0
  jne .returnTrue

  add r12, 8
  jmp .while

.returnTrue:
  mov rax, 1
  pop rbp
  ret

.returnFalse:
  mov rax, 0
  pop rbp
```

```
ret
```

These problems are new to the final; you must work them to pass.

25 points each

► Complete the definition of a C-style function

```
void capitalize(char* s);
```

which converts all lower-case characters (those in ASCII range 97-122) to upper case (65-89) in the (nul-terminated) string s

```
capitalize:

.while:
  cmp byte [rdi], 0
  je .done

  cmp byte [rdi], 'a'
  jb .continue
  cmp byte [rdi], 'z'
  ja .continue

  sub byte [rdi], ('a' - 'A')

.continue:
  inc rdi
  jmp .while

.done
  ret
```

► Complete the definition of a C-style function

```
long* find(long* array, unsigned long length, long value);
```

which takes a pointer to a signed qword array, a (qword) length, and a signed qword value, and returns either a pointer to the array element containing the value, or the null pointer if the value does not exist in the array.

```
find:
  push r14

  ; This assumes the length is given in the number of bytes in the array, not
  ; the number of qwords. This is more common in assembly code.

  lea r14, [rdi + rsi] ; Ending address
.while:
  cmp rdi, r14
  je .returnNull:

  cmp qword [rdi], rdx
  jne .continue

  ; Found
  pop r14
  mov rax, rdi
  ret

.continue:
  add rdi, 8
  jmp .while

.returnNull:
  pop r14
  mov rax, 0
  ret
```