CS33: Intro Computer Organization
Midterm, Form: A

Name: ____________________

ID: ____________________

Problem 1

What is the output of the following code?
Assume that int is 32 bits, short is 16 bits, and the representation is two's complement.

```
unsigned int x = 0xDEADBEEF;
unsigned short y = 0xFFFF;
signed int z = -1;
if (x > (signed short) y)
    printf("Hello");
if (x > z)
    printf("World");
```

(a) Prints nothing.

(b) Prints "Hello"

(c) Prints "World"

(d) Prints "HelloWorld"

Problem 2

Which of the following instructions read memory?

(a) movq %rbx, %rbp

(b) cvtsi2ssl %rdi,%xmm0

(c) leaq 4(%rax,%rbx,2), %rcx

(d) cmov %rbx, %rcx

(e) subq %rax, (%rbx)

Name: ____________________

ID: ____________________

Problem 3

. Which expression will evaluate to `0x1` if `x` is a multiple of 32 and `0x0` otherwise? Assume that `x` is an unsigned `int`.

(a) `!(x & 0x1f)`

(b) `!(x & 0x3f)`

(c) `(x & 0x1f)`

(d) `(x | 0x3f)`

(e) `!(x ^ 0x1f)`

Problem 4.

`%rsp` is `0xdeadbeefdeadd0d0`. What is the value in `%rsp` after the following instruction executes?

```
pushq %rbx
```

(a) `0xdeadbeefdeadd0d4`

(b) `0xdeadbeefdeadd0d8`

(c) `0xdeadbeefdeadd0cc`

(d) `0xdeadbeefdeadd0c8`

Name: ______________________

ID: ______________________

Problem 5.

Consider the C declaration

```
int array[10] = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9};
```

Suppose that the compiler has placed the variable `array` in the `%ecx` register. How do you move the value at `array[3]` into the `%eax` register? Assume that `%ebx` is 3.

(a) `leal 12(%ecx),%eax`

(b) `leal (%ecx,%ebx,4),%eax`

(c) `movl (%ecx,%ebx,4),%eax`

(d) `movl 8(%ecx,%ebx,2),%eax`

(e) `leal 4(%ecx,%ebx,1),%eax`

Problem 6.

. Consider the following code, what is the output of the printf?

```
int x = 0x15213F10 >> 4;
char y = (char) x;
unsigned char z = (unsigned char) x;
printf("%d, %u", y, z);
```

(a) -241, 15

(b) -15, 241

(c) -241, 241

(d) -15, 15

Name: ______________________

ID: ______________________

Problem 7.

Short answers.

a. What is the value returned by sizeof for **int (*B[3])[5]?**

b. What is the value returned by sizeof for **int *(A[3][5])?**

c. What is the value returned by sizeof for **int (*C)[3][5]?**

Problem 8.

In the following questions assume the variables `a` and `b` are signed integers and that the machine uses two's complement representation. Also assume that `MAX INT` is the maximum integer, `MIN INT` is the minimum integer, and `W` is one less than the word length (e.g., `W` = 31 for 32-bit integers).
Match each of the descriptions on the left with a line of code on the right (write in the letter). You will be given 2 points for each correct match.

1. One's complement of `a`

   ____

2. `a`.

   ____

3. `a & b`.

   ____

4. `a * 7`.

   ____

5. `a / 4` .

   ____

6. `(a < 0) ? 1 : -1` .

   ____

a. `~(~a | (b ^ (MIN_INT + MAX_INT)))`

b. `((a ^ b) & ~b) | (~(a ^ b) & b)`

c. `1 + (a << 3) + ~a`

d. `(a << 4) + (a << 2) + (a << 1)`

e. `((a < 0) ? (a + 3) : a) >> 2`

f. `a ^ (MIN_INT + MAX_INT)`

g. `~((a | (~a + 1)) >> W) & 1`

h. `~((a >> W) << 1)`

i. `a >> 2`

Name: ____________

ID: ____________

Problem 9.

a. First, deduce the following functions.

```
000000000000064a <func1>:
 64a:  48 83 ec 18          sub    $0x18,%rsp
 64e:  89 7c 24 0c          mov    %edi,0xc(%rsp)
 652:  83 7c 24 0c 00       cmpl   $0x0,0xc(%rsp)
 657:  75 07                jne    660 <func1+0x16>
 659:  b8 01 00 00 00       mov    $0x1,%eax
 65e:  eb 0e                jmp    66e <func1+0x24>
 660:  8b 44 24 0c          mov    0xc(%rsp),%eax
 664:  83 e8 01             sub    $0x1,%eax
 667:  89 c7                mov    %eax,%edi
 669:  e8 05 00 00 00       callq  673 <func2>
 66e:  48 83 c4 18          add    $0x18,%rsp
 672:  c3                   retq

0000000000000673 <func2>:
 673:  48 83 ec 18          sub    $0x18,%rsp
 677:  89 7c 24 0c          mov    %edi,0xc(%rsp)
 67b:  83 7c 24 0c 00       cmpl   $0x0,0xc(%rsp)
 680:  75 07                jne    689 <func2+0x16>
 682:  b8 00 00 00 00       mov    $0x0,%eax
 687:  eb 0e                jmp    697 <func2+0x24>
 689:  8b 44 24 0c          mov    0xc(%rsp),%eax
 68d:  83 e8 01             sub    $0x1,%eax
 690:  89 c7                mov    %eax,%edi
 692:  e8 b3 ff ff ff       callq  64a <func1>
 697:  48 83 c4 18          add    $0x18,%rsp
 69b:  c3                   retq
```

```
int func1(unsigned int n)
{
    if ( _______ )
        _____________;
    else
        _____________;
}

int func2(unsigned int n)
{
    if ( _______ )
        _____________;
    else
        _____________;
}
```

b) Suppose we call `func1(4)`, what is the return value?

Name: ____________________

ID: ____________________

c) Consider the case where `func1(2)` is called: Draw the stack at the point in the program execution when the stack is largest. To get full credit, you must show where the stack pointer is pointing, and indicate the names/locations of any unknown registers pushed to stack, any known values pushed to the stack, and any unused stack space.

Assume each line of the table represents 4 bytes.

caller return address <7-4>
caller return address <3-0>

Problem 10.

Solve the final bomb_lab phase:

phase_8:

```
0x6d8 <+0>:  sub    $0x8,%rsp
0x6dc <+4>:  mov    $0x1,%esi
0x6e1 <+9>:  mov    $0x1,%ecx
0x6e6 <+14>: mov    $0x0,%eax
0x6eb <+19>: jmp    0x715 <phase_8+61>
0x6ed <+21>: sub    $0x1,%esi
0x6f0 <+24>: mov    %ecx,%eax
0x6f2 <+26>: mov    %esi,%edx
0x6f4 <+28>: lea    0x200925(%rip),%r8        # 0x201020 <map>
0x6fb <+35>: lea    (%r8,%rdx,8),%rdx
0x6ff <+39>: movzbl (%rdx,%rax,1),%edx
0x703 <+43>: cmp    $0xff,%dl
0x706 <+46>: je     0x748 <phase_8+112>
0x708 <+48>: cmp    $0x3,%dl
0x70b <+51>: je     0x752 <phase_8+122>
0x70d <+53>: mov    %r9d,%eax
0x710 <+56>: cmp    $0x21,%dl
0x713 <+59>: je     0x75c <phase_8+132>
0x715 <+61>: lea    0x1(%rax),%r9d
0x719 <+65>: cltq
0x71b <+67>: movzbl (%rdi,%rax,1),%eax
0x71f <+71>: cmp    $0x77,%al
0x721 <+73>: je     0x6ed <phase_8+21>
0x723 <+75>: cmp    $0x61,%al
0x725 <+77>: je     0x734 <phase_8+92>
0x727 <+79>: cmp    $0x73,%al
0x729 <+81>: je     0x739 <phase_8+97>
0x72b <+83>: cmp    $0x64,%al
0x72d <+85>: jne    0x73e <phase_8+102>
0x72f <+87>: add    $0x1,%ecx
0x732 <+90>: jmp    0x6f0 <phase_8+24>
0x734 <+92>: sub    $0x1,%ecx
0x737 <+95>: jmp    0x6f0 <phase_8+24>
0x739 <+97>: add    $0x1,%esi
0x73c <+100>:jmp    0x6f0 <phase_8+24>
0x73e <+102>:mov    $0x0,%eax
0x743 <+107>:callq  0x6a4 <explode_bomb>
0x748 <+112>:mov    $0x0,%eax
0x74d <+117>:callq  0x6a4 <explode_bomb>
0x752 <+122>:mov    $0x0,%eax
0x757 <+127>:callq  0x68a <phase_defused>
0x75c <+132>:mov    $0x0,%eax
0x761 <+137>:callq  0x6be <s3cret_phase>
```

Possibly helpful GDB interaction: (x/64bx just means print 64 bytes of memory in hex format)

```
(gdb) x/64bx map
0x201020 <map>:     0xff   0xff   0x00   0x00   0x00   0xff   0x00   0xff
0x201028 <map+8>:   0xff   0x00   0xff   0x00   0xff   0x00   0xff   0xff
0x201030 <map+16>:  0xff   0x00   0x00   0xff   0xff   0x00   0xff   0x00
0x201038 <map+24>:  0xff   0xff   0x00   0x00   0x03   0xff   0x00   0x00
0x201040 <map+32>:  0xff   0x00   0x00   0xff   0xff   0x00   0xff   0xff
0x201048 <map+40>:  0xff   0x00   0xff   0xff   0x00   0x00   0xff   0xff
0x201050 <map+48>:  0xff   0x00   0x00   0x00   0x00   0xff   0x00   0xff
0x201058 <map+56>:  0xff   0xff   0xfe   0xff   0xff   0xff   0x21   0xff
(gdb) x/16bx unimportant_array
0x201060 <unimportant_array>:    0x00   0x00   0x00   0x00   0x00   0x00   0x00   0x00
0x201068 <unimportant_array+8>:  0x00   0x00   0x00   0x00   0x00   0x00   0x00   0x00
```

1. What string will defuse the bomb?

2. What string will activate the secret phase?

Name: ______________________

ID: ______________________

Problem 11.

The next problem concerns code generated by GCC for a function involving a switch statement. The code uses a jump to index into the jump table:

```
  400519: jmpq   *0x400640(,%rdi,8)
```

Using GDB, we extract the 8-entry jump table as:

```
0x400640:    0x0000000000400530
0x400648:    0x0000000000400529
0x400650:    0x0000000000400520
0x400658:    0x0000000000400529
0x400660:    0x0000000000400535
0x400668:    0x000000000040052a
0x400670:    0x0000000000400529
0x400678:    0x0000000000400530
```

The following block of disassembled code implements the branches of the switch statement:

```
# on entry: %rdi = a, %rsi = b, %rdx = c
  400510: mov    $0x5,%rax
  400513: cmp    $0x7,%rdi
  400517: ja     400529
  400519: jmpq   *0x400640(,%rdi,8)
  400520: mov    %rdx,%rax
  400523: add    %rsi,%rax
  400526: salq   $0x2,%rax
  400529: retq
  40052a: mov    %rsi,%rdx
  40052d: xor    $0xf,%rdx
  400530: lea    0x70(%rdx),%rax
  400534: retq
  400535: mov    $0xc,%rax
  400538: retq
```

Name: ____________________

ID: ______________________

Fill in the blank portions of C code below to reproduce the function corresponding to this object code. You can assume that the first entry in the jump table is for the case when `a` equals 0.

```
long test(long a, long b, long c)
{
  long answer = ______;
  switch(a)
  {
    case ___:
      c = ______;
      /* Fall through */
    case ___:
    case ___:
      answer = ______;
      break;
    case ___:
      answer = ______;
      break;
    case ___:
      answer = ______;
      break;
    default:
      answer = ______;
  }

  return answer;
}
```

The Midterm

Problem 1

prints nothing.

- x > (signed short) y
  - signed short is converted to unsigned long for this comparison
  - so it is false
- x > z
  - is also false because nothing is greater than unsigned -1

Problem 2

subq %rax, (%rbx)

- because the parentheses indicate a memory reference

Problem 3

!(x & 0x1f)

- x & 0x1f says "only let the last five bits through the gate"
- !() of that says 1 if those five bits are 00000, 0 if anything else
- that's equivalent to divisibility by 32

Problem 4

0xdeadbeefdeadd0c8

- because pushq pushes a 64-bit (8 byte) word onto the stack
- so it decrements %rsp by 0x8

Problem 5

movl (%ecx, %ebx, 4), %eax

- array[3] is indeed located at %ecx + 4 * %ebx

Problem 6

-15, 241

- y is 0xF1, z is also 0xF1
- y is signed char, so it is -15
- z is unsigned char, so it is 241

Problem 7

a. sizeof(B) = 24

- because in `int (*B[3])[5]`, `B` is an array of 3 pointers to [arrays of 5 ints]

b. sizeof(A) = 120

- because in `int (*A[3][5])`, `A`is a 3 by 5 array of pointers to ints

c. sizeof(C) = 8

- because in `int (*C)[3][5]`, `C` is a pointer to a 3 by 5 array of ints

Problem 8

one's complement of `a`: `a ^ (MIN_INT + MAX_INT)`

- because `a ^ (MIN_INT + MAX_INT) = a ^ -1 = ~a`
- one's complement is basically "flip the bits"

`a`: `((a ^ b) & ~b) | (~(a ^ b) & b)`

- because `(c & ~b) | (~c & b) = c ^ b`
- you can see that because XOR is basically "the first and not the second, or the second and not the first"
- then let `c = a ^ b`
- and `((a ^ b) & ~b) | (~(a ^ b) & b) = (a ^ b) ^ b = a`

`a & b`: `~(~a | (b ^ (MIN_INT + MAX_INT)))`

- because that's `~(~a | ~b)` because of the above
- and that's `a & b` by deMorgan

`a * 7`: `1 + (a << 3) + ~a`

- because `(a << 3) = a * 8`
- and `1 + ~a = -a`

`a / 4`: `((a < 0) ? (a + 3) : a) >> 2`

- because `/` always rounds toward 0 while `>>` always rounds toward `-inf` so you have to add a bias

`(a < 0) ? 1 : -1`: `~((a >> W) << 1)`

- because if `a < 0`, then `(a >> W)` is -1, `<< 1` is -2, and `~` is 1
- and if `a >= 0`, then `(a >> W)` is 0, `<< 1` is 0, and `~` is -1

Problem 9

(a)

```c
int func1(unsigned int n) {
 if ( n == 0 ) // mov %edi, 0xc(%rsp); cmpl $0x0, 0xc(%rsp); jne 660
  return 1; // mov $0x1, %eax
 else
  return func2 ( n - 1 ); // sub $0x1, %eax; mov %eax, %edi; callq 673
}

int func2(unsigned int n) {
 if ( n == 0 ) // mov %edi, 0xc(%rsp); cmpl $0x0, 0xc(%rsp); jne 689
  return 0; // mov $0x0, %eax
 else
  return func1 ( n - 1 ); // sub $0x1, %eax; mov %eax, %edi; callq 64a
}
```

(b)

```
func1(4) = func2(3) = func1(2) = func2(1) = func1(0) = 1
```

(c)

```
caller return address bytes 7 to 4
caller return address bytes 3 to 0
...
...
```

```
0x00 00 00 02
...
...
...
0x00 00 00 00
0x00 00 06 6e // return address of stack frame for func1(2)
...
...
0x00 00 00 01
...
...
...
0x00 00 00 00
0x00 00 06 97 // return address of stack frame for func2(1)
...
...
0x00 00 00 00
...
...
...
```

Problem 10

It's a maze!

So

`%rdi` contains the address of your input string

`%rsi` contains your `y` position

`%rcx` contains your `x` position

`%rdx` becomes the address you calculate for your position: `%rdi + 8*%rsi + %rcx`

You start in `(1, 1)` counting from the top left, zero-based

`a` is left, `d` is right, `s` is down, `w` is up

You're trying to get to `0x03` to finish the phase, or `0x21` to activate the secret phase

Defuse the bomb: `sdsdd`

Activate the secret phase: `sdssassdssddddw` (you have to go out of the `map`, through the `0xfe`, into the `unimportant_array` near the end of that path)

Problem 11

switch statement :)

```c
long test(long a, long b, long c) {
  long answer = 5;
  switch(a) {
    case 5:
      c = b ^ 0xf; // or 15
    case 0: // or 7
    case 7: // or 0
      answer = c + 112;
      break;
    case 2: // or 4
      answer = (c + b) << 2; // or 12
      break;
    case 4: // or 2
      answer = 12; // or (c + b) << 2
      break;
    default:
      answer = 5;
  }
}
```