CS33: Intro Computer Organization
Fall 2019 Midterm

Name: Rayeeb Chung

UID: 605 131 578

IMPORTANT INSTRUCTIONS: You must write your name on both the FRONT AND BACK of the exam. You may do so now. Do not open the exam.

This is an open book, open notes exam, but you cannot share books/notes. Please follow the university guidelines in reporting academic misconduct.

Please wait until everyone has their exam to begin. We will let you know when to start.

Good luck!

6

Question 1. C Puzzles (8pts)
You are running the following program on the cs33.seas.ucla.edu machine (ISA is x86-64).

```
// Create some random values
int x = random();
int y = random();
int z = random();
/* convert to other forms */
unsigned ux = (unsigned) x;
unsigned uy = (unsigned) y;
double dx = (double) x;
double dy = (double) y;
double dz = (double) z;
```

8 -8
10 -6
0 - uy
0 - 5
= -5
= 5
5 - 3 = -(3-5)
2 = -(-2) = 2
1000 0000

For each of the following C-puzzles, in the column marked answer, either mark true if the expression always holds (ie. always yields 1), or give a counterexample (eg. TMIN) which breaks the rule.

x is positive or x < UX

1111 1111 1111
0001 0000 0001

Expression	Answer ("True" or describe a counterexample)
(x<y) == (-x>-y)	(example) x=Tmin, y=0
ux-uy == -(y-x)	(a) False, x=1, y=2, 1U-2U = UMAX -1, -(1-2)= 1
(x >= 0) || (x < ux)	(b) False, x = Umax, because it is negative then casts and isn't < ux
ux & (~(1 << 31)) < 0	(c) False, ux = 1 00...001 & 0111...111 <! 0
~x + ~y + 1 == ~(x+y)	(d) True
dx*dy == x*y	(e) True
dx*y == x*dy	(f) True
dx+dy+dz == dz+dy+dx	(g) True
((x >> 31) << 31) <= x	(h) True

any + # = 0
any - # = 1000

1111
1000
0111

ux & ~10000
ux & 0111
ux &

0001 & 01111 <! 0

0111 1111
0111 1111
1111 1110

0001 -> 1110
0010 -> 1101
1011
+ 0001
1100

~0011
1100

1000 -> 0111
-> 0111
1110

0111 -> 1000
+1
= 0000

4

Question 2. Multiple Choice (10pts)

For the following multiple choice questions, select all that apply.

✓ 1. Which of the following registers are guaranteed to have the same value before and after a call instruction in x86-64?

(a) rax X
[✓ (b)] rbx
(c) rdi
[✓ (d)] rbp
[✓ (e)] rsp)

✓ 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)

✓ 3. Assuming our ISA is x86-64, which of the following operations could we identify as modifying the ~~*values on*~~ the program stack? (content) of stack

[✓ (a)] call <func>
(b) addq $8, %rsp
[✓ (c)] movq %rax, (%rbp)
(d) movq 20(%rsp), %rax
[✓ (e)] pushq %rbp
[✓ (f)] addq %rax, 8(%rsp)

-536870912

-268435456

✓ 4. What hexadecimal bit pattern would be found in memory in an x86-64 machine, for the number negative 33, when the corresponding datatype is an "int" in C?

```
(a)    0x80 0x00 0x00 0x33
(b)    0x80 0x00 0x00 0x21
(c)    0x21 0x00 0x00 0x80
(d)    0x33 0x00 0x00 0x80
(e)    0xFF 0xFF 0xFF 0x21   33
(f)    0xDF 0xFF 0xFF 0xFF
(g)    0xFF 0x21
(h)    0xDF 0xFF
```

[✓ (e)]

1000 0000 0000 0000

FF FF FF DF

1111 1111 1111 1111111 1101 1111

1111 1111 1111 1111 1111 1111 1101 1111

5

0010 0001

1101

31302428

0001

1111 1111 (1111 1101)

654 221

1111 1111 1111 0010 0001

1000000 1001001

1111 1111 1111 1111 1111

31

5. If a,b,c are n-bit signed integers, and c is the result of a+b, under what conditions can we be *guaranteed* that c **is not** the true result under full precision arithmetic?

(a) a >= 2^{n-2} && b >= 2^n-2

(b) a <= -2^{n-2} && b <= -2^n-2

(c) a - b > 0

(d) a + b > 0

(e) a > 0 && b > 0 && c < 0

[✓ (f)] a < 0 && b < 0 && c > 0

a >= 2^1 ~~&~~ b >= 2^4

4 bit

a >= 2^2

b >= 2

```
 1000
+1000
 0000
```

```
 1
 0100
 0100
 1000
```

0101

```
 1 1
 0111
 0111
 1110
```

```
 1 1 1
 0101
 0111
 1110
```

3

Question 3. This Bytes (8pts)

Say if it's impossible

For this question, either interpret the value as a bit pattern, or write down the corresponding value.

For floating point questions, use the following 8-bit floating point representation based on the IEEE floating point format:

- There is a sign bit in the most significant bit.
- The next 4 bits are the exponent. The exponent bias is: $2^{4-1}-1=7$
- The last 3 bits are the fraction.
- The representation encodes numbers of the form: $V = (-1)^s \times M \times 2^E$, where M is the significand and E is the biased exponent.

E = exp - bias

s exp frac
1 4 3

$2^3-1=7$

Bit Pattern	Value Description
(a) 0111 1111	Negative of smallest possible signed integer (ie. -TMin)
✓ (b) 0100 0000	Largest signed integer that is a power-of-2
✓ (c) 1111 1111	TMin + Tmax
(d) 01100 000	Floating Point value: 32
0 0 1 0 0 0 0 1	✓ (e) 33 (interpret as "char"-sized integer)
1 1 0 1 1 1 1 0	(f) 0 (interpret as "char"-sized integer)
0 0 1 1 1 0 0 0	(g) 2 (interpret as "8-bit float")
1 0 1 1 1 1 1 1	(h) 2 × 1.111 (interpret as "8-bit float")

0110

M =

exp = 4

4-7 = -3

32 = 2^5 0010000 0,000

32/2 = 16 R0
16/2 = 8 R0
8/2 = 4 R0
4/2 = 2 R0
2/2 = 1 R0
1/2 =

1.00000×2^5

5+7 = 12 = 1100

12 - 7 = 5

$1.100 \times 2^5 = 32$

0111 = $\bigcirc 7$ = 2

7-7 = 0

2^0

1.111

2 × 1.000 = 2

Question 4. Be the compiler! (6 pts)

Suppose we have the following C code:

```
if(a>b) { a+=b;}
```

Also assume that a and b are "int", **a** is in %eax, **b** is in %ebx. You can use other registers as temporaries.

(a) Write an x86-64 assembly snippet that is equivalent to this statement in C, while making sure to use a jump (aka branch) instruction. Please use a label (.eg L1) as the target of the jump. **(4 pts)**

```
cmp %ebx, %eax
jg .L1

.L1:
    add %ebx, %eax
```

-1

(b) Write an x86-64 assembly snippet that is equivalent to this statement in C, while making sure ***NOT*** to use a jump (aka branch) instruction. **(2 pts)**

```
cmp %ebx, %eax
```

-2

Question 5: Interpreting Assembly (6 pts)

For each of the functions in x86-64 assembly below, convert them into a plausible version of the C code.

`Assembly of function`	Write a plausible C-code for the function

`movq %rdi,%rax`
`salq $4,  %rax` ·16 <<4 * 16
`addq %rdi,%rax`
`addq %rax,%rax`
`ret`
	int fun(int n) {
	int temp = n;
	temp = temp · 16 + n;
	return (2 · temp)
	}
	2

`movl (%rdi),%edx` edx = *p
`addl %edx,(%rsi)` *x += *p
`movl %edx,%eax` return *p
`ret`
	int fun(int *p , int *x) {
	int temp = *p;
	*x += temp;
	return temp;
	}
	3

Question 6: (6 pts)

The C code and assembly is given below for a function, but without values of M and N.

`#define M __`
`#define N __`
 	
`int array1[M][N];`
`int array2[N][M];`
`int copy(int i, int j) {`
`  array1[i][j] = array2[j][i];`
`}`
	`movslq %edi,%rdi`
	`movslq %esi,%rax`
	`lea    (%rax,%rax,4),%rdx`
	`add    %rdi,%rdx`
	`mov    array2(,%rdx,4),%edx`
	`lea    0x0(,%rdi,8),%rsi`
	`sub    %rdi,%rsi`
	`add    %rax,%rsi`
	`mov    %edx,array1(,%rsi,4)`
	`retq`

What are the values of M and N?

M = 5 ✓ 6

N = 7 ✓

rdx = 5N + M

edx : array2 + 4(5N+M)

rsi = 8M

rsi = 7M

7M + N

array1 + 4(7M + N) =

array2 + 4(5N + M)

28M + N

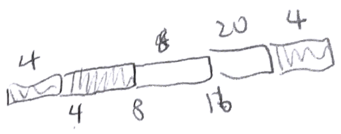
= 20N + M

2

Question 7. ISA Design (4 pts)

In one or two sentences only, why have 32-bit ISAs become less popular for personal computers (laptops/desktops/cell-phones) over the last two decades?

Now, there are 64-bit ISA, so each register can hold 64 bits. This allows for faster processing since each register has more space.



Question 8: Stack Structures (12pts)

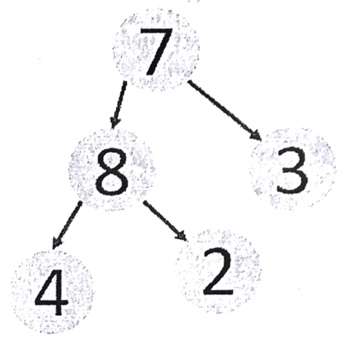
Considered an *unordered* tree represented with this struct. The function "smallest" will retrieve the smallest element from the tree.

```
struct node {
  int value;          4 bytes
  struct node* left, *right;
} node;                        16 bytes

int smallest(node * n) {
  int temp, ret = n->value;
  if(n->left) {
     temp=smallest(n->left);
     if(temp<ret) ret = temp;
  }
  if(n->right) {
     temp=smallest(n->right);
     if(temp<ret) ret = temp;
  }
  return ret;
}
```

smallest node 2

Assume this is the input data-structure stored using node structs:



```
smallest:
0x4004ed <+0>:   push   %rbp
0x4004ee <+1>:   push   %rbx
0x4004ef <+2>:   sub    $0x8,%rsp
0x4004f3 <+6>:   mov    %rdi,%rbp
0x4004f6 <+9>:   mov    (%rdi),%ebx
0x4004f8 <+11>:  mov    0x8(%rdi),%rdi
0x4004fc <+15>:  test   %rdi,%rdi
0x4004ff <+18>:  je     0x40050b <smallest+30>
0x400501 <+20>:  callq  0x4004ed <smallest>
0x400506 <+25>:  cmp    %eax,%ebx
0x400508 <+27>:  cmovg  %eax,%ebx
0x40050b <+30>:  mov    0x10(%rbp),%rdi
0x40050f <+34>:  test   %rdi,%rdi
0x400512 <+37>:  je     0x40051e <smallest+49>
0x400514 <+39>:  callq  0x4004ed <smallest>
0x400519 <+44>:  cmp    %eax,%ebx
0x40051b <+46>:  cmovg  %eax,%ebx
0x40051e <+49>:  mov    %ebx,%eax
0x400520 <+51>:  add    $0x8,%rsp
0x400524 <+55>:  pop    %rbx
0x400525 <+56>:  pop    %rbp
0x400526 <+57>:  retq
```

(note, cmov is conditional move)

everything

Caller Return Address
rbp old
rbx old
unused
0x400506
rbp old
rbx old
unused
0x400506

(assume 8 bytes wide!!)

$\boxed{-2.5}$

(a) Draw what is on the stack, provided in the space above, when smallest(2) is entered (ie. when it is called, and just before the instruction at 0x4004ed is executed) . Assume the root of the pictured tree is the input.
Note: If you don't know what a register value is, just mark it as "old rbp" etc. If you know what a register value is, write the corresponding value. **(6pts)**

(b) What is the size of the node struct? **(2pts)**

24 bytes

$$\frac{\text{int}}{4} \; \frac{\quad}{4} : \frac{*}{8} \; \frac{*}{8} = 24$$

(c) Is there any padding in the struct "node" due to alignment rules? **(1pts)**

Yes

(d) Can you rearrange the elements of "node" to reduce its size? **(1pts)**

No, because if you do, then there will still need to be padding so that the size is divisible by 8.

(e) Which of the following are possible starting addresses for a node: **(2pts)**

(i) `0x7ffe4d3be87c`
(ii) `0x7ffe4d3be874`
(iii) `0x000444444440`
(iv) `0xfffff1234568`

-1

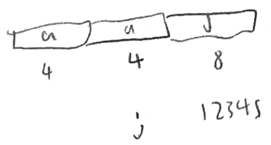
Question 9 (Bonus): Your points overfloweth! (5pts)

Consider the following code (a variation on a hopefully-familiar example).

```
#include <stdio.h>
#include <stdlib.h>

typedef struct {
  int a[2];
  double d;
} struct_t;

double fun(int i, int j) {
  volatile struct_t s;
  s.d = 12345.0;
  s.a[i] = j;
  return s.d;
}
```



a[3] = 33

What input arguments to function "fun" would return the value 33.0?

i = 3, j = 33

2

```
int function(int x) {
    int m1 = 0x11 | (0x11 << 8);
    int mask = m1 | (m1 << 16);
    int s = x & mask;
    s += x>>1 & mask;
    s += x>>2 & mask;
    s += x>>3 & mask;
    s = s + (s >> 16);
    mask = 0xF | (0xF << 8);
    s = (s & mask) + ((s >> 4) & mask);
    return (s + (s>>8)) & 0x3F;
}
```

0001 0001 0001 0001 0000 0000

0001 0001 0000 0000 0001 0001 0000 0000 $\rightarrow$ int

0 4 0 4 0 4 0 4

1000

0100

What does this function do?

- It creates a mask of: 0001 0001 000... 0000
- Then, it compares it to s.
- It computes several factors of s w/ mask, and adds it to s.

Back of Exam

Name: Rayeeb Chang

UID: 605131578

	Score	Points Possible
1	6	8
2	4	10
3	3	8
4	3	6
5	5	6
6	6	6
7	2	4
8	8.5	12
9 (ec)	2	5
10 (ec)	1	5
Total	40.5	60 +10ec