CS33: Intro Computer Organization
Midterm, Form: $\boxed{\text{B}}$ 39

Name: [redacted]

ID: [redacted]

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

Problem	Score	Points Possible
1	12	18
2	8	8
3	9	12
4	20	20
5	12	15
6	4	0
7	15	17
8	21	10
	101	

Question 1. The bigger the better. (18, 3 pts each) 12

1. What is the largest number that can be represented by a 7 bit floating point number (say with the same rules as IEEE 754 floating point), with a 1 bit sign, 3 bit exponent, and 3 bit significand (bias=3)? 15
2. In C, what's the largest int plus one? 2^{31} $\boxed{-2^{31}}$
3. Consider an n-bit signed number, what's the largest one? $2^{n-1} - 1$
4. In C, what's the smallest unsigned int minus one? $2^{32} - 1$
5. Which can represent the largest number in C, the largest float or the largest signed long or largest unisgned int? signed long largest float.
6. Which integer type in C is large enough to store a pointer without loss of precision? long

1) looks like

0 110 111.

This is $(-1)^0 \times 2^{e-bias} \times (1+f)$ bias = $2^{3-1} - 1. = 4 - 1 = 3.$

$= 2^{6-3} \times (1 + \frac{1}{2} + \frac{1}{4} + \frac{1}{8}) = 8 \times (1\frac{7}{8})$

$= 8 \times (\frac{15}{8}) = 15.$

Question 2. Matchmaker (8 Pts, 1 pts each)

8

Pretend to be a compiler.
You are free to assign registers to variables however you choose. Assume x and y are of type int. Remember, the compiler(me) may have done some optimizations.

g y=x+y	(a) shl $ 5 %edi
a x=x*32 (x in $rdi)	(b) xorl %edi %edi
f x=x*5+3 (x in $edi)	(c) shr $ 31 %edi
c x=(x < 0) ? -1 : 0	(d) movl $1 %eax
d x=1 (% x now in %eax)	(e) imul %edi %edx
h x=x*3+5 (x in $eai)	(f) leaq 3(%edi,%edi,4)
b x=0 (xor on x^x cancels itself).	(g) addl %edi %edi
e x=x*y (x in %edx, y in %edi)	(h) leaq 5(%edi,%edi,2)

Question 3. Unholy Union (9 pts) 12

9

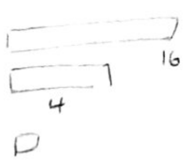
```
#include <stdio.h>
#include <string.h>

void main(char** argv, int argc) {   -1
  union U {
    char s[16];
    int i;
    char c;        struct a;
  } u;

  strcpy(u.s,"evil_prof"); //Copy string to destination from source

  printf("%x\n", u.c);
  printf("%x\n", u.i);
}
```

struct a{
char[16];
}

1. What does this program print? (6 pts)

~~0x65~~
~~0x65 76 69 6C~~
e v i l

2. To which addresses may this union be aligned? (3pts)

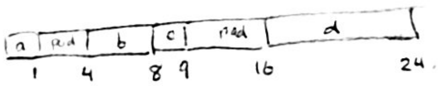


can be aligned to address that are a multiple of 16 (end with 000_2)

This aligns to 4.

Question 4. Deconstructed (20 pts, 5 Each)

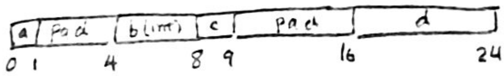
```c
#include <stdio.h>

typedef struct {
  char a;
  int b;
  char c;
  double d;
} X;

void main(char** argv, int argc) {
  X x[10];
  printf("%d\n",(int)sizeof(X));
  printf("%d\n",(int)sizeof(x));
}
```



1. What does this program print?

X is 24 bytes, so prints 24
x is 240 bytes 240.

2. Draw the memory layout of X, where your diagram indicates which byte offset each variable is located at, as well as any space allocated just for padding:



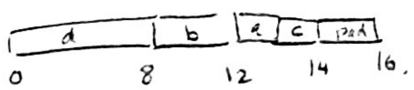
3. Write an assembly snippet that performs `x[10].c=0`. Assume that x is in register $rdi.

```
movq   $0xA, $rax          # 10 into rax.
imult  $0x18, $rax         # multiply 10 by 24.
movq   $0, 8($rdi, $rax)   # set x[10].c  (offset of 0x8 because that's where c is).
```

4. Describe how you would reduce the memory consumption of x. How small can you make x?

* Arrange fields with largest data types first:

```c
typedef struct{
  double d;
  int b;
  char a;
  char c;
}
```



The struct itself can be 14 bytes, but is aligned by 16.

Thus, can make array 160 bytes.

Question 5. I can puzzle, (15 Pts, 2 pts each)

Answer these true false puzzles. Assume the following setup:

```
int x = foo();
int y = bar();
unsigned ux = x;
unsigned uy = y;
```

- F `x > 0 && y > 0 ⟹ x + y > 0` No, can get overflow to negatives.
- F `5*ux > ux` ux = 1000 5×ux = 40 => 1010000 => 1000, not > 1000
- F `x < 100 ⟹ 10*ux > ux` x = 0.
- T `-x == ~x+1`
- -3 T `x >> 2 == x / 4`

Question 6. ... and so can you! (Up to 4 pts Extra Credit)

1. Write a C Puzzle of the form above, give the solution, and explain why you think its cool.

(~(x & y)) & ~((~x) & (~y)) == 1 true +4

⇓

!(x==y)

- (~(x&y)) & ~((~x) & (~y)) is a way to express x ^ y, since ~(x&y) creates a mask where all places x and y both have 1's goes to 0, and ~(~x & ~y)) is another way to express x | y.

- if x ^ y is true (non zero), then the two cannot be equal.

It's cool because it tests that you know to test for equivalence with !(x^y), and how ^ works. Also, there are a lot of squigglies.

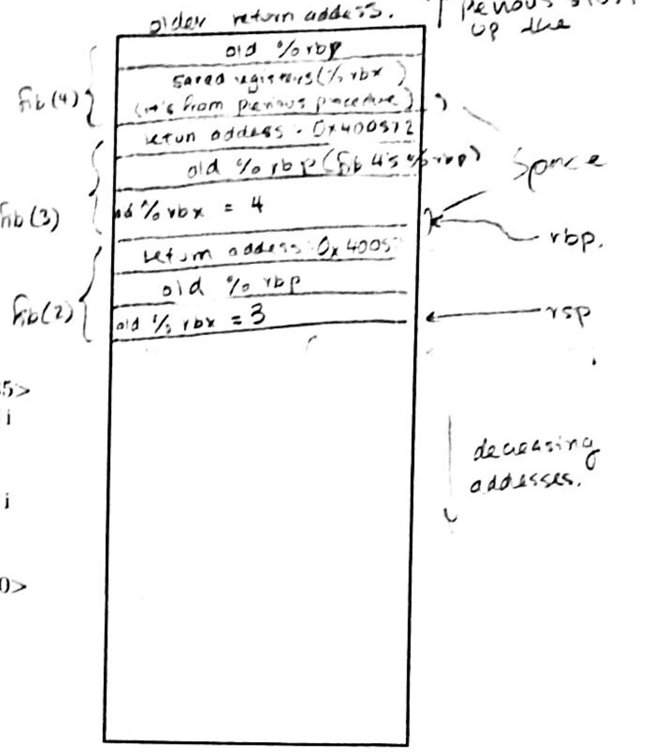
Question 7. Your fibs are stacking up (16 Pts)

Recall the fibbonacci code that we discussed in class, and its associated disassembly: (the instruction addresses are omitted for simplicity, just the offsets remain)

```
int fib(int a) {
  if(a < 2) {
    return 1;
  }
  return fib(a-1) + fib(a-2);
}
```

+11

```
fib: 0x40055d <+0>:  push   %rbp
     0x40055e <+1>:  push   %rbx
     0x40055f <+2>:  sub    $0x8,%rsp
     0x400563 <+6>:  mov    %edi, %ebx
     0x400565 <+8>:  cmp    $0x1, %edi
     0x400568 <+11>: jle    0x400580 <fib+35>
     0x40056a <+13>: lea    -0x1(%rdi), %edi
     0x40056d <+16>: callq  0x40055d <fib>
heat 0x400572 <+21>: mov    %eax, %ebp
     0x400574 <+23>: lea    -0x2(%rbx), %edi
     0x400577 <+26>: callq  0x40055d <fib>
     0x40057c <+31>: add    %ebp, %eax
     0x40057e <+33>: jmp    0x400585 <fib+40>
     0x400580 <+35>: mov    $0x1, %eax
     0x400585 <+40>: add    $0x8, %rsp
     0x400589 <+44>: pop    %rbx
     0x40058a <+45>: pop    %rbp
     0x40058b <+46>: retq
```



1. This function calls itself recursively. Imagine in gdb we put a breakpoint on line 0x40056d, then call fib(4). Furthermore we hit continue two more times in gdb, so that the stack frames of fib(4), fib(3), and fib(2) are all on the stack. Draw the contents of the stack in the box above, and be sure to indicate the stack pointer. Draw everything you know about ~~the~~ stack! If you know what the value is, write the value, otherwise indicate what it is. (10 pts)
2. On which line(s) (specify as offset from fib please!) is/are callee saved registers being saved? (1pt)
3. On which line(s) is/are callee saved registers being restored? (1pt)
4. On which line(s) is/are the input argument to fib being set? (1pt)
5. On which line(s) is/are the return value from fib being set (for the final time)? (1pt)
6. On which line(s) is/are the stack being allocated? (1pt)
7. On which line(s) is/are the stack being de-allocated? (1pt)

+4

2). 0x400Se (0x40055d as well, if you consider %rbp callee-saved).
3) 0x400589 (0x40058a as well, if you consider %rbp callee-saved).
4). 0x40056a, 0x400574
5). 0x40057c
6). 0x400572, 0x40055d, 0x40055e. 5f
7). 0x400589, 0x40058a. 85

Question 8. Oh Fuuuudge (10 pts)

You just finished your CS32 homework when all of a sudden you "**rm -f my_homework.c**". Thankfully, you didn't delete your binary file – phew. You forgot all the expressions in your source code, but you kind of remembered the overall structure. It's time to analyze the binary to fill out the remaining expressions.

```
<+0>:  mov     $0x1,  %r9d
<+6>:  jmp     <func+54>
<+8>:  movslq  %r9d,  %rax
<+11>: mov     (%rdi, %rax, 4),  %r8d
<+15>: lea     -0x1(%r9),  %eax
<+19>: jmp     <func+28>
<+21>: mov     %edx,  0x4(%rdi, %rcx, 4)
<+25>: sub     $0x1,  %eax
<+28>: test    %eax,  %eax
<+30>: js      <func+43>
<+32>: movslq  %eax,  %rcx
<+35>: mov     (%rdi, %rcx, 4),  %edx
<+38>: cmp     %r8d,  %edx
<+41>: jg      <func+21>
<+43>: cltq
<+45>: mov     %r8d,  0x4(%rdi, %rax, 4)
<+50>: add     $0x1,  %r9d
<+54>: cmp     %esi,  %r9d
<+57>: jl      <func+8>
<+59>: repz  retq
```

r9d is i   esi i-
rax holds i
r8d is key
r9d -1 (j = i-1)
%eax holds j.
eax holds j.
test eax
if negative jump 43
rcx holds j.
edx holds arr[j]
move key into arr[j + 4].
i++.
esi is n.

rdi is arr.
r9d is i
esi is n
r8d is key
eax is j.

1. Fill in the code (2 Pts each .. Extra Credit Possible)

```
void func(int arr[], int n)
{
    int i, key, j;
    for (i = _1_; i _<n_; i++)
    {
        key = arr[_i_];
        j = i-1;
        while (_j_>=0 && _arr[j]_ > _key_)
        {
            arr[_j+1_] = arr[_j_];
            j = _j-1_;
        }
        arr[_i_] = key;
    }
}
```

r8d is key

i = 1
arr[i] = key = 3
j = 0
arr[j] = 1
arr[i] = 3.
i = 2
arr[i] = key = 2.
j = 1
arr[j] = 3.
arr[2] = arr[1]

2. What well-known algorithm is this? (2 Pts Extra Credit)

Bubble sort.

+1