CS33: Intro Computer Organization
Midterm, Form: A 19

Name: Hao Wang
ID: 204681263

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

Problem	Score	Points Possible
1	18	18
2	8	8
3	6	12
4	19	20
5	~~9~~	15
6	3	0
7	16	17
8	22	10
	101	

18

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

1. Consider an n-bit signed number, what's the largest one? $2^{n-1}-1$
2. In C, what's the largest `int` plus one? 0x10000000
3. Which can represent the largest number in C, the largest `float` or the largest `signed long` or largest `unisgned int`? largest float
4. Which integer type in C is large enough to store a pointer without loss of precision? long (in 64 bits machine)
5. 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
6. In C, what's the smallest `unsigned int` minus one? 0xFFFFFFFF
   the largest unsigned int

$1\frac{7}{8} \times 2^{3}$

15 66
8 8

E-1.

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

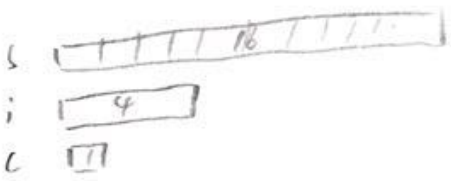
Pretend to be a compiler. 8

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.

(h)	x=(x < 0) ? -1 : 0	(a) `addl %edi %edi`
(c)	x=x*3+5	(b) `xorl %edi %edi`
(f)	x=x*32	(c) `leaq 5(%edi,%edi,2)`
(b)	x=0	(d) `imul %edi %edx`
(e)	x=1	(e) `movl $1 %eax`
(g)	x=x*5+3	(f) `shl $ 5 %edi`
(d)	x=x*y	(g) `leaq 3(%edi,%edi,4)`
(a)	y=x+y	(h) `shr $ 31 %edi`

Question 3. Unholy Union (9 pts) 12 6

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

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

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

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



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

   ~~65 76 69 6c 20 70 72 6f 66~~

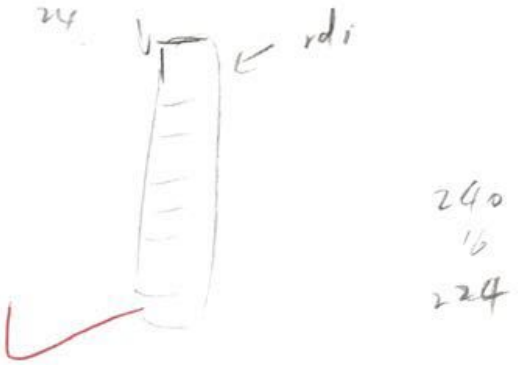
   65 76 69 6c

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

   ~~This is a union with size 16 bytes, since it is aligned to the largest element. size~~

Question 4. Deconstructed (20 pts, 5 Each)

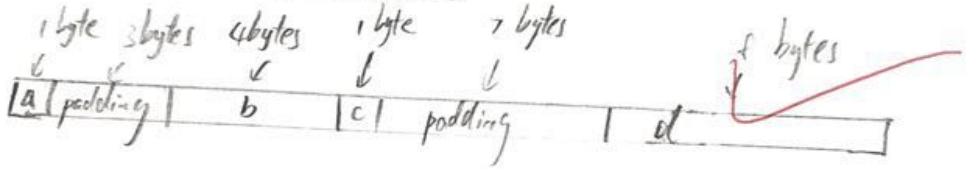
```
#include <stdio.h>

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?

   24
   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.

   movb $0, 224(%rdi)

   -1

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

   Put elements in struct in descending order of size

   i.e.

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

   in this way sizeof(x) is 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;
```

-3 false `-x == ~x+1`

✓ false `x > 0 && y > 0 ⟹ x + y > 0`

✓ false `5*ux > ux`

✓ false `x < 100 ⟹ 10*ux > ux`

-3 true `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.

double d = fool();

+3

d*d >= 0 true, cool because this is a new characteristics of double that int doesn't have.

ux == x false. cool because it involves implicit conversion, and I find it tricky.

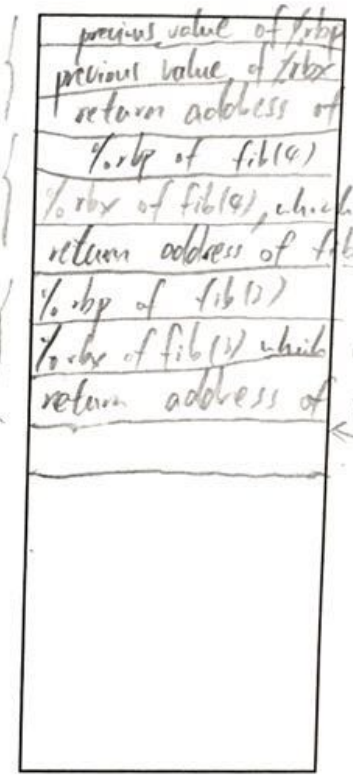
x = (int)(float)x false cool because it tests precison lost of int to float.

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);
}
```

```
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>
     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) <40> and <41>
3. On which line(s) is/are callee saved registers being restored? (1pt) <+44> and <+45>
4. On which line(s) is/are the input argument to fib being set? (1pt) <+13> and <+23>
5. On which line(s) is/are the return value from fib being set (for the final time)? (1pt) <+35>
6. On which line(s) is/are the stack being allocated? (1pt) <+2> (<+0> <+1> also uses stack memory without direct allocation)
7. On which line(s) is/are the stack being de-allocated? (1pt) <+40>

(<+44>, <+45> also deallocates memory without direct de allocation of memory)

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
```

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

```
void func(int arr[], int n)
{
    int i, key, j;
    for (i = ____; i<____; i++)
    {
        key = arr[____];
        j = i-1;

        while (____>=0 && ____ > ____)
        {
            arr[____] = arr[____];
            j = ____;
        }
        arr[____] = key;
    }
}
```

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

insertion sort (ascending)

+2.

ASCII TABLE

Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char
0	0	[NULL]	32	20	[SPACE]	64	40	@	96	60	`
1	1	[START OF HEADING]	33	21	!	65	41	A	97	61	a
2	2	[START OF TEXT]	34	22	"	66	42	B	98	62	b
3	3	[END OF TEXT]	35	23	#	67	43	C	99	63	c
4	4	[END OF TRANSMISSION]	36	24	$	68	44	D	100	64	d
5	5	[ENQUIRY]	37	25	%	69	45	E	101	65	e
6	6	[ACKNOWLEDGE]	38	26	&	70	46	F	102	66	f
7	7	[BELL]	39	27	'	71	47	G	103	67	g
8	8	[BACKSPACE]	40	28	(	72	48	H	104	68	h
9	9	[HORIZONTAL TAB]	41	29	)	73	49	I	105	69	i
10	A	[LINE FEED]	42	2A	*	74	4A	J	106	6A	j
11	B	[VERTICAL TAB]	43	2B	+	75	4B	K	107	6B	k
12	C	[FORM FEED]	44	2C	,	76	4C	L	108	6C	l
13	D	[CARRIAGE RETURN]	45	2D	-	77	4D	M	109	6D	m
14	E	[SHIFT OUT]	46	2E	.	78	4E	N	110	6E	n
15	F	[SHIFT IN]	47	2F	/	79	4F	O	111	6F	o
16	10	[DATA LINK ESCAPE]	48	30	0	80	50	P	112	70	p
17	11	[DEVICE CONTROL 1]	49	31	1	81	51	Q	113	71	q
18	12	[DEVICE CONTROL 2]	50	32	2	82	52	R	114	72	r
19	13	[DEVICE CONTROL 3]	51	33	3	83	53	S	115	73	s
20	14	[DEVICE CONTROL 4]	52	34	4	84	54	T	116	74	t
21	15	[NEGATIVE ACKNOWLEDGE]	53	35	5	85	55	U	117	75	u
22	16	[SYNCHRONOUS IDLE]	54	36	6	86	56	V	118	76	v
23	17	[ENG OF TRANS. BLOCK]	55	37	7	87	57	W	119	77	w
24	18	[CANCEL]	56	38	8	88	58	X	120	78	x
25	19	[END OF MEDIUM]	57	39	9	89	59	Y	121	79	y
26	1A	[SUBSTITUTE]	58	3A	:	90	5A	Z	122	7A	z
27	1B	[ESCAPE]	59	3B	;	91	5B	[	123	7B	{
28	1C	[FILE SEPARATOR]	60	3C	<	92	5C	\	124	7C	|
29	1D	[GROUP SEPARATOR]	61	3D	=	93	5D	]	125	7D	}
30	1E	[RECORD SEPARATOR]	62	3E	>	94	5E	^	126	7E	~
31	1F	[UNIT SEPARATOR]	63	3F	?	95	5F	_	127	7F	[DEL]

1.