CS33: Intro Computer Organization	Name:	
	UID:	

IMPORTANT INSTRUCTIONS: You must write your name on the back page of the exam (And above). 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.

Note that there is an ASCII Table at the end of this exam. You will need it at some point. Do NOT detach the ASCII table.

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

Good luck!

## 1) GDB Lies (10, 1pt each)

Suppose we debug the following program (assignments omitted), and break at the printf:

```
void main(int argc, char* argv) {
  int i=...
  unsigned u=...
  float f=...
  double d=...
  printf("...",...)
}
```

List any outputs from gdb that *must* have been tampered with. (ie. if it might not have been tampered with, then don't list it) For example, the output of the first command has been tampered with, because the return value is not expected:

```
(gdb) print sizeof(double)
$0 = 37
```

```
(gdb) print sizeof(short)
$1 = 2
(gdb) print sizeof(0)
$2 = 4
(gdb) print (unsigned) -1 > 1
$3 = 0
(gdb) print 0 - 1
$4 = -1
(gdb) print 1U - 2
$5 = 4294967295
(gdb) p (int)(float)i == i
$6 = 1
(gdb) p (int)(double)i == i
$7 = 0
(gdb) p (unsigned)(int) u == u
$8 = 0
(gdb) p/f 0xC2040000
$9 = 33
(gdb) p/x (int)3.14159
$10 = 0x40490fd0
```

# 2) One of a kind! (10, 1 pt each)

A. For each instruction below, write *one* alternate instruction which performs the same operation. The alternate should not use the same instruction type (known as an opcode). It is acceptable if the flags do not match. (example solutions, other reasonable solutions are good as well)

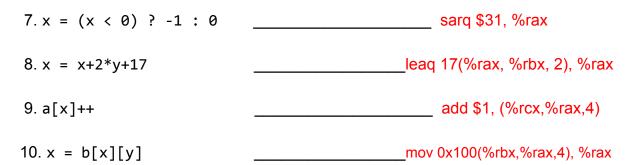
1.	leaq (%rbx, %rbp), %rbp	 addq %rbx, %rbp
2.	leaq (, %rdi, 2), %rdi	 mulq \$2, %rdi
3.	mov %rax, %rax	 nop
4.	add \$0, %eax	 nop, clear upper bits
5.	xor %rbx, %rbx	 mov \$0, rbx

B. Rewrite the following with *one* instruction:

6. cltq

(assume x is in %rax, y is in %rbx, array a (declared as int a[256]) is in address in %rcx, and that array b (declared as char b[100][4]) is at address 0x100.

\_\_\_\_\_ movslq %eax, %rax



## 3) Bitwise Number Classification (10 pts, 2 pts each)

int func1(int x) {

}

return (x>>31) & 0x1;

Match the following datalab implementations to their descriptions.

```
int func2(int x) {
  return (!!x) & (!(x+x));
}
int func3(int x) {
  return !x;
}
int func4(int x) {
    int nx = \sim x;
    int nxnz = !!nx;
    int nxovf = !(nx+nx);
    return nxnz & nxovf;
}
int func5(int x) {
  int minus_x = \sim x+1;
  return ((minus_x|x) \rightarrow 31) & 1;
}
   1. isTmin: Returns 1 if x == Tmin, 0 otherwise
   2. isTmax: Returns 1 if x == Tmax, 0 otherwise
   3. isNegative: Returns 1 if x < 0, 0 otherwise
   4. isNonZero: Returns 1 if x!=0, 0 otherwise
   5. isZero: Returns 1 if x == 0, and 0 otherwise
```

**Q4) Array of hope (10 pts)**. Consider the following code on the left, and answer the questions on the right:

```
typedef struct {
                                               1. What is printed? (2 pts each, 6 pts total)
  char g;
                                                a. Size of struct: 48
  short n[10];
  int o;
                                                          g: 0, padding 1; n: 2-21, padding 2; o:24-27,
                                                          padding 4; w: 32-39; r: 40-43, padding 4.
  double w;
                                                b. Size of union array: 24*10 = 240
  float r;
                                                          Determined by largest element 'short n[10]':
} struct_elem;
                                                          2*10 with padding of 4
typedef union {
                                                c. Floating point: 33.0
                                                          0100 0010 0000 0100 0000 0000 0000 0000
  char g;
                                                          Sign: 0; Exp: 1000 0100; Frac: 0000 1000 and
  short n[10];
                                                          rest
  int o;
  double w;
  float r;
                                               2. Notice that get_val is missing a definition. The
} union_elem;
                                                  following is the disassembly from gdb:
                                               Dump of assembler code for function get_val:
struct elem struct_array[10];
                                                                       movslq %esi,%rsi
                                                  0x00000066a <+0>:
union_elem union_array[10];
                                                  0x00000066d <+3>:
                                                                       movslq %edi,%rdi
                                                  0x000000670 <+6>:
                                                                       lea
                                                                              (%rdi,%rdi,2),%rax
   _ get_val(int x, int y) {
                                                  0x000000674 <+10>:
                                                                       lea
                                                                              (%rsi,%rax,8),%rdx
                                                  0x000000678 <+14>:
                                                                              0x2009c1(%rip),%rax
}
                                                  0x00000067f <+21>:
                                                                       movzwl 0x2(%rax,%rdx,2),%eax
                                                  0x000000684 <+26>:
                                                                       retq
int main(int argc, char** argv) {
                                               What is the definition of get val? (3pts)
  int a[16][16];
                                               (Hint: 0x2009c1(%rip) is the address of either
  printf("%ld\n", sizeof(struct_elem));
                                                  struct_array or union_array)
  printf("%ld\n", sizeof(union_array));
  union array[0].o=0x42040000;
                                                get val(int x, int y) {
  printf("%f\n", union_array[0].r);
}
                                                    return struct_array[x].n[y]
                                               }
                                               3. Which of the following orders minimizes the size of the
                                                  struct? (1pts): C
                                                          grown 48
                                                   a.
                                                   b.
                                                          nworg 48
                                                          wrong 40
```

### 5) Mutually assured instruction. (10 pts)

a) First, deduce the following functions. (4pts)

```
000000000000064a <func1>:
                                                   int func1(unsigned int n) {
64a: 48 83 ec 18
                                $0x18,%rsp
                         sub
                                                        if (n == 0)
64e: 89 7c 24 0c
                               %edi,0xc(%rsp)
                         mov
                                                            return 1:
652: 83 7c 24 0c 00
                               $0x0,0xc(%rsp)
                         cmpl
                                                        else
657: 75 07
                         jne
                               660 <func1+0x16>
659: b8 01 00 00 00
                               $0x1,%eax
                                                            return func2(n-1);
                         mov
65e: eb 0e
                               66e <func1+0x24>
                         jmp
                                                   }
660: 8b 44 24 0c
                               0xc(%rsp),%eax
                         mov
664: 83 e8 01
                         sub
                               $0x1,%eax
                                                   int func2(unsigned int n) {
667: 89 c7
                               %eax,%edi
                         mov
                                                        if ( n == 0 )
669: e8 05 00 00 00
                         callq 673 <func2>
66e: 48 83 c4 18
                         add
                               $0x18,%rsp
                                                             return 0;
672: c3
                         retq
                                                        else
                                                             return func1(n-1);
0000000000000673 <func2>:
673: 48 83 ec 18
                               $0x18,%rsp
                         sub
677: 89 7c 24 0c
                               %edi,0xc(%rsp)
                         mov
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
                               697 <func2+0x24>
                         jmp
689: 8b 44 24 0c
                               0xc(%rsp),%eax
                         mov
                               $0x1,%eax
68d: 83 e8 01
                         sub
690: 89 c7
                               %eax,%edi
                         mov
692: e8 b3 ff ff ff
                         callq 64a <func1>
697: 48 83 c4 18
                         add
                               $0x18,%rsp
69b: c3
                         reta
```

b) Suppose we call func1(4), what is the return value? (1pt)

1

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. **(5pts)** 

# Assume each line of the table represents 4 bytes.

caller return address <7-4>			
caller return address <3-0>			
Unused			
Unused			
0x 00 00 00 02			
Unused			
Unused			
Unused			
0x 00 00 00 00			
0x 00 00 06 6e			
Unused			
Unused			
0x 00 00 00 01			
Unused			
Unused			
Unused			
0x 00 00 00 00			
0x 00 00 06 97			
Unused			
Unused			
0x 00 00 00 00			
Unused			
Unused			
Unused			

**Q6)** Hex marks the spot (10 pts): As you finally navigate your way to the last problem of the exam, still reeling from the maze of increasingly difficult challenges you have solved to get here, you look down to see what you feared the most: your bomb is still active! Either find your way and deactivate the bomb, or risk exploding the entire class. **Solve the final bomblab phase:** 

# phase\_8:

```
0x6d8 <+0>: sub
                   $0x8,%rsp
0x6dc <+4>: mov
                   $0x1,%esi
0x6e1 <+9>: mov
                  $0x1,%ecx
                   $0x0,%eax
0x6e6 <+14>: mov
0x6eb <+19>: jmp
                   0x715 <phase 8+61>
0x6ed <+21>: sub
                   $0x1,%esi
                  %ecx,%eax
0x6f0 <+24>: mov
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
                   $0xff,%dl
0x703 <+43>: cmp
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
                         0xff
                               0x00
                                      0x00
                                            0x00
                                                  0xff
                                                         0x00
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
                                            0xff
0x201040 <map+32>: 0xff
                         0x00
                               0x00
                                      0xff
                                                  0x00
                                                         0xff
                                                               0xff
0x201048 <map+40>: 0xff
                         0x00
                               0xff
                                      0xff
                                            0x00
                                                  0x00
                                                         0xff
                                                               0xff
                               0x00
0x201050 <map+48>: 0xff
                         0x00
                                      0x00
                                            0x00
                                                  0xff
                                                         0x00
                                                               0xff
0x201058 <map+56>: 0xff
                         0xff
                               0xfe
                                      0xff
                                            0xff
                                                  0xff
                                                         0x21
                                                               0xff
(gdb) x/16bx unimportant_array
                               0x00
                                      0x00
                                                  0x00
                                                         0x00
                                                               0x00
0x201060 <unimportant_array>:
                                            0x00
                                                                     0x00
                                                                            0x00
0x201078 <unimportant_array+8>:
                               0x00
                                      0x00
                                            0x00
                                                  0x00
                                                         0x00
                                                               0x00
                                                                     0x00
                                                                            0x00
```

1. What string will defuse the bomb? (7 pts)

#### Answer 1: sdsdd

2. What string will activate the secret phase? (3 pts)

Answer 2: sdssassdssddddw

# 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	1	65	41	Α	97	61	a
2	2	[START OF TEXT]	34	22		66	42	В	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	1	71	47	G	103	67	g
8	8	[BACKSPACE]	40	28	(	72	48	Н	104	68	h
9	9	[HORIZONTAL TAB]	41	29	)	73	49	1	105	69	i
10	Α	[LINE FEED]	42	2A	*	74	4A	J	106	6A	j
11	В	[VERTICAL TAB]	43	2B	+	75	4B	K	107	6B	k
12	C	[FORM FEED]	44	2C	,	76	4C	L	108	6C	1
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	0	111	6F	0
16	10	[DATA LINK ESCAPE]	48	30	0	80	50	P	112	70	р
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	Υ	121	79	у
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	1	125	7D	}
30	1E	[RECORD SEPARATOR]	62	3E	>	94	5E	^	126	7E	~
31	1F	[UNIT SEPARATOR]	63	3F	?	95	5F	_	127	7F	[DEL]
									I		

# **Back of Exam**

Name:	
UID:	

	Score	Points Possible
1		10
2		10
3		10
4		10
5		10
6		10
Total		50 +10ec