CS33: Intro Computer Organization

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
```

10

List GDB Outputs Tampered With: $0, $3, $7, $8, $10, $9

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.

1. `leaq (%rbx, %rbp), %rbp` add %ebx, %ebp

2. `leaq (, %rdi, 2), %rdi` imul 2, %edi

3. `mov %rax, %rax` add $0, %eax

~~4. `add $0, %eax`~~ mov %rax, %rax

5. `xor %rbx, %rbx` mov 0, %rbx

6. `cltq` movslq %eax, %rax

B. Rewrite the following with *one* instruction:
(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.

7. `x = (x < 0) ? -1 : 0` sar 31, %eax

8. `x = x+2*y+17` leaq 17(%rax, %rbx,2), %rax

9. `a[x]++` add 1, (%rcx, %rax, 4)

~~10. `x = b[x][y]`~~ leaq 0x100(%rbx, %rax, 4), %rax

8

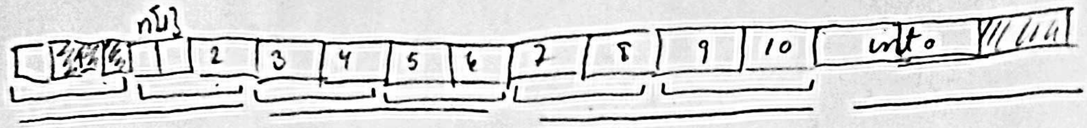
3) Bitwise Number Classification (10 pts, 2 pts each)

Match the following datalab implementations to their descriptions.

```
int func1(int x) {          ✓
    return (x>>31) & 0x1;
}

int func2(int x) {
  return (!!x) & (!(x+x));
}

int func3(int x) { ✓
  return !x;
}

int func4(int x) {
    int nx = ~x;       1000
    int nxnz = !!nx;   1
    int nxovf = !(nx+nx);  1
    return nxnz & nxovf;
}

int func5(int x) {       ✓
  int minus_x = ~x+1;
  return ((minus_x|x) >> 31) & 1;
}
```

1. isTmin: Returns 1 if x == Tmin, 0 otherwise func 2
2. isTmax: Returns 1 if x == Tmax, 0 otherwise func 4
3. isNegative: Returns 1 if x < 0, 0 otherwise func 1
4. isNonZero: Returns 1 if x!=0, 0 otherwise func 5
5. isZero: Returns 1 if x == 0, and 0 otherwise func 3

\10



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

$16 + 4 \times 8 = 16 + 32$

```
typedef struct {
  char g ;      1
  short n[10]; 2
  int o; 4
  double w; 8
  float r; 4
} struct_elem;

typedef union {
  char g;
  short n[10];
  int o;
  double w;
  float r;
} union_elem;

struct_elem struct_array[10];
union_elem union_array[10];

____ get_val(int x, int y) {
  ...
}

int main(int argc, char** argv) {
  int a[16][16];
  printf("%ld\n", sizeof(struct_elem));
  printf("%ld\n", sizeof(union_array));
  union_array[0].o=0x42040000;
  printf("%f\n", union_array[0].r);
}
```

0x42040000

01000010

0100 0010 0000 0100 0...

S E M

2^{-5}

$2^2 + 2^7$

E=5

$100001 = 1 + 2^5$

1. What is printed? **(2 pts each, 6 pts total)**

48 ✓ +2

200 24×10 +1

33 ✓ +2.

2. Notice that `get_val` is missing a definition. The following is the disassembly from gdb:

```
Dump of assembler code for function get_val:
   0x00000066a <+0>:     movslq %esi,%rsi
   0x00000066d <+3>:     movslq %edi,%rdi              rax=3 rdi
   0x000000670 <+6>:     lea    (%rdi,%rdi,2),%rax
   0x000000674 <+10>:    lea    (%rsi,%rax,8),%rdx=x+24y
   0x000000678 <+14>:    lea    0x2009c1(%rip),%rax
   0x00000067f <+21>:    movzwl 0x2(%rax,%rdx,2),%eax
   0x000000684 <+26>:    retq              eax=2+rax+2(x+24y)
```

What is the definition of get_val? **(3pts)**
(Hint: `0x2009c1(%rip)` is the address of either `struct_array` or `union_array`)

```
int get_val(int x, int y) {
        +1



}
```

3. Which of the following orders minimizes the size of the struct? **(1pts)**
   a. g r o w n
   b. n w o r g
   [✓ c.] w r o n g

+1

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

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

```
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 ( n == 0 ) ✓
        return 1 ;
    else ✓
    return func2(n-1) ;
}

int func2(unsigned int n)
{
    if ( n == 0 )
        return 0 ; ✓
    else
    return func1(n-1) ;
}
```

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

1 ✓

[problem continued on the next page]

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.

16+8=24

	caller return address <7-4>	
	caller return address <3-0>	
		6
	%edi = 2	
	0x 000	
	0x 66e ✓	
		6
	%. edi = 1	
	0x 000 ✓	
	0x 697	
		6
	%.edi = 0	
stack pointer →		

4/5

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
0x6e6 <+14>:  mov     $0x0,%eax
0x6eb <+19>:  jmp     0x715 <phase_8+61>
0x6ed <+21>:  sub     $0x1,%esi              esi=0
0x6f0 <+24>:  mov     %ecx,%eax              eax=1
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> //explode
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              r9d=1
                                                           0
0x719 <+65>:  cltq
0x71b <+67>:  movzbl  (%rdi,%rax,1),%eax          eax=rdi ~~+rax~~ =rdi
0x71f <+71>:  cmp     $0x77,%al
0x721 <+73>:  je      0x6ed <phase_8+21>
0x723 <+75>:  cmp     $0x61,%al
0x725 <+77>:  je      0x734 <phase_8+92>          77     3
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>
```

2

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
0x201068 <unimportant_array>:    0x00  0x00  0x00  0x00  0x00  0x00  0x00  0x00
0x201070 <unimportant_array+8>:  0x00  0x00  0x00  0x00  0x00  0x00  0x00  0x00
```

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

2 0x77 0x3 → w

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

w!