CSE 351 Spring 2017 – Midterm Exam (8 May 2017)

Please read through the entire examination first!

- You have 50 minutes for this exam. Don't spend too much time on any one problem!
- The last page is a reference sheet. Feel free to detach it from the rest of the exam.
- The exam is CLOSED book and CLOSED notes (no summary sheets, no calculators, no mobile phones).

There are 5 problems for a total of 50 points. The point value of each problem is indicated in the table below. Write your answer neatly in the spaces provided.

Please do not ask or provide anything to anyone else in the class during the exam. Make sure to ask clarification questions early so that both you and the others may benefit as much as possible from the answers.

Good Luck!

Your Name:_____Sample Solution_____

UWNet ID:_______woof2017_________

Name of person to your left | Name of person to your right

__________________________ | ____________________________

Problem	Topic	Max Score
1	Integers & Floats	7
2	Hardware to Software	7
3	Structs & Arrays	7
4	Pointers & Memory	14
5	Stack Discipline	15
TOTAL		**50**

1. Integers and Floats (7 points)

a. In the card game Schnapsen, 5 cards are used (Ace, Ten, King, Queen, and Jack) from 4 suits, so 20 cards in total. What are the minimum number of bits needed to represent a single card in a Schnapsen deck?

5

We need 2 bits to represent 4 suits, and 3 bits to represent 5 ranks. So 5 bits in total.

b. How many negative numbers can we represent if given 7 bits and using two's complement?

$\mathbf{2^6}$

Using 7 bits, the MSB has to be 1 for negative numbers. So there are 2^6 negative numbers in total.

Consider the following pseudocode (we've written out the bits instead of listing hex digits):

```
int a = 0b0100 0000 0000 0000 0000 0011 1100 0000
int b = (int)(float)a
int m = 0b0100 0000 0000 0000 0000 0011 0000 0000
int n = (int)(float)m
```

c. Circle one: True or **False**:

`a == b`

The right-most 1 will be truncated (cannot fit in Mantissa)

d. Circle one: **True** or False:

`m == n`

No precision will be lost

e. How many IEEE single precision floating point numbers are in the range [4, 6) (That is, how many floating point numbers are there where 4 <= x < 6?)

$\mathbf{2^{22}}$

4 in binary is $1.0 \cdot 2^2$.

6 in binary is $1.1 \cdot 2^2$.

So in Mantissa the right-most 22 bits can be either 0 or 1. Therefore, there are 2^{22} bits in range $[4, 6)$

2. Hardware to Software (7 points)

a) If the word size of a machine is m bits, which of the following is typically true:

- TRUE / **FALSE**: m bits is the size of an integer

- **TRUE** / FALSE: m bits is the width of a register

b) If the size of a pointer on a machine is 32 bits, the size of the address space is how many bytes?

2^{32}

c) ISA stands for: _**Instruction Set Architecture**____

d) **TRUE** / FALSE: The number of registers available is part of the ISA.

e) Part of the object file that keeps track of symbols/labels needed by this source file is the:

Relocation Table

f) The tool used to combine one or more .o files into an executable is called the:

___**Linker or `ld`**______.

(Hint: the answer is not "gcc", we want the general or specific name of tool that does this particular step.)

3. Structs and Arrays (7 points)
You are given the following C program run on a 64-bit x86-64 (little endian) processor:

```
struct diddle {
   int x;
   struct diddle *y;
   int z;
   char c[3];
 };

 int main(void) {
   struct diddle d;
   d.x = 0xdeadbeef;
   d.y = &d;
   d.z = d.x >> 16;
   d.c[0] = 0x12;
   d.c[1] = 0x34;
   d.c[2] = 0x56;
   return 0;
 }
```

a. Below is a view of the stack. Suppose we have just reached the return statement and assume `d` is placed at address **`0x7fffffffac0`**. Please fill in the bytes on the stack in hex (you may omit the 0x prefix).

Address	**+0**	**+1**	**+2**	**+3**	**+4**	**+5**	**+6**	**+7**
0x7fffffffac0	ef	be	ad	de				
0x7fffffffac8	c0	fa	ff	ff	ff	07	00	00
0x7fffffffad0	ad	de	ff	ff	12	34	56	
0x7fffffffad8								

c. What is the total size of this struct in bytes?

24 Bytes

d. Is there a reordering of the fields in diddle that would reduce its total size? If so, what is it?

No. The struct has to end at an address of multiple of 8.

4. **Pointers, Memory & Registers (14 points)**

Assuming a 64-bit x86-64 machine (little endian), you are given the following variables and initial state of memory (values in hex) shown below:

Address	+0	+1	+2	+3	+4	+5	+6	+7
0x00	AB	EE	1E	AC	D5	8E	10	E7
0x08	F7	84	32	2D	A5	F2	3A	CA
0x10	83	14	53	B9	70	03	F4	31
0x18	01	20	FE	34	46	E4	FC	52
0x20	4C	A8	B5	C3	D0	ED	53	17

```
int* ip = 0x00;
short* sp = 0x20;
long* yp = 0x10;
```

a) Fill in the type and value for each of the following C expressions. If a value cannot be determined from the given information answer UNKNOWN.

Expression (in C)	Type	Value (in hex)
`yp + 2`	**long***	**0x20**
`*(sp - 1)`	**short**	**0x52FC**
`ip[5]`	**int**	**0x31F40370**
`&ip`	**int****	**UNKNOWN**

b) Assuming that all registers start with the value 0, except **`%rax`** which is set to 0x4, fill in the values (in hex) stored in each register after the following x86 instructions are executed. *Remember to give enough hex digits to fill up the width of the register name listed.*

```
movl 2(%rax), %ebx
leal (%rax,%rax,2), %ecx
movsbl 4(%rax), %edi
subw (,%rax,2), %si
```

Register	Value (in hex)
`%rax`	`0x0000 0000 0000 0004`
`%ebx`	`0x84f7 e710`
`%ecx`	`0x0000 000c`
`%rdi`	`0x0000 0000 ffff fff7`
`%si`	`0x7B09`

5. Stack Discipline (15 points)

Examine the following recursive function:

```
long sunny(long a, long *b) {
  long temp;
  if (a < 1) {
    return *b - 8;
  } else {
    temp = a - 1;
    return temp + sunny(temp - 2, &temp);
  }
}
```

Here is the x86_64 assembly for the same function:

```
0000000000400536 <sunny>:
  400536:       test   %rdi,%rdi
  400539:       jg     400543 <sunny+0xd>
  40053b:       mov    (%rsi),%rax
  40053e:       sub    $0x8,%rax
  - - - - - - - - - - - - - - - - - - - - - - -   Breakpoint
  400542:       retq
  400543:       push   %rbx
  400544:       sub    $0x10,%rsp
  400548:       lea    -0x1(%rdi),%rbx
  40054c:       mov    %rbx,0x8(%rsp)
  400551:       sub    $0x3,%rdi
  400555:       lea    0x8(%rsp),%rsi
  40055a:       callq  400536 <sunny>
  40055f:       add    %rbx,%rax
  400562:       add    $0x10,%rsp
  400566:       pop    %rbx
  400567:       retq
```

We call **`sunny`** from **`main()`**, with registers %**`rsi`** = **`0x7ff`**…**`ffad8`** and %**`rdi`** = 6. The value stored at address **`0x7ff`**…**`ffad8`** is the long value 32 (0x20). We set a breakpoint at "**`return *b - 8`**" (i.e. we are just about to return from **`sunny()`** without making another recursive call). We have executed the **`sub`** instruction at **`40053e`** but have not yet executed the **`retq`**.

Fill in the register values on the next page and **draw what the stack will look like <u>when the program hits that breakpoint</u>**. Give both a description of the item stored at that location and the value stored at that location. If a location on the stack is not used, write "unused" in the Description for that address and put "-----" for its Value. You may list the Values in hex or decimal. Unless preceded by **`0x`** we will assume decimal. It is fine to use **`f`**…**`f`** for sequences of **`f`**'s as shown above for %**`rsi`**. Add more rows to the table as needed. Also, fill in the box on the next page to include the value this call to **`sunny`** will *finally* return to **`main`**.

Register	Original Value	Value **at Breakpoint**
`rsp`	`0x7ff…ffad0`	`0x7ff…ffa90`
`rdi`	`6`	`0`
`rsi`	`0x7ff…ffad8`	`0x7ff…ffaa0`
`rbx`	`4`	`2`
`rax`	`5`	`-6`

DON'T FORGET → What value is **finally** returned to **`main`** by this call? **1**

Memory address on stack	Name/description of item	Value
0x7fffffffffffad8	Local var in **`main`**	0x20
0x7fffffffffffad0	Return address back to **`main`**	0x400827
0x7fffffffffffac8	Saved %rbx	*4*
0x7fffffffffffac0	*temp*	*5*
0x7fffffffffffab8	*Unused*	---------------
0x7fffffffffffab0	Return address to sunny	**0x40055f**
0x7fffffffffffaa8	Saved %rbx	5
0x7fffffffffffaa0	*temp*	2
0x7fffffffffffa98	*Unused*	---------------
0x7fffffffffffa90	Return address to sunny	**0x40055f**
0x7fffffffffffa88		
0x7fffffffffffa80		
0x7fffffffffffa78		
0x7fffffffffffa70		
0x7fffffffffffa68		
0x7fffffffffffa60		