

Introduction to Computer Systems

2018 Fall Middle Examination

Name_____ Student No._____ Score_____

Problem 1:

[1]	[2]	[3]
[4]	[5]	[6]
[7]		

Problem 2:

[1]	[2]
[3]	[4]
[5]	[6]
[7]	[8]
[9]	[10]
[11]	[12]

Problem 3:

1. [1]	[2]
[3]	[4]
[5]	[6]
[7]	[8]
2.	
3. [1]	[2]
[3]	[4]

Problem 4:

1. [1]	[2]
[3]	[4]

2.

3.

Problem 5:

1 [1]

[2]

[3]

[4]

[5]

2. [1]

[2]

[3]

[4]

[5]

[6]

[7]

[8]

[9]

[10]

3.

Problem 6:

1 [1]

[2]

[3]

[4]

[5]

[6]

[7]

[8]

2

3 a.

b.

Problem 1: (14 points)

1. Consider the following C program

```
int a = 0x21;
short b = a - 30;
unsigned int c = a | 0x11;
int d = ~(int)b + !c;
unsigned short e = d - 15;
```

Assume the program will run on an **8-bit machine** and use two's complement arithmetic for signed integers. A 'short' integer is encoded in **4 bits**, while a normal 'int' is encoded in **8 bits**. Please fill in the blanks below. ($2^7=128$)

Expression	Binary Representation
a	0010 0001
b	[1]
c	[2]
d >> 2	[3]
e	[4]
(e & 0xf) ^ 0x3	[5]
(a - 54) + (0x11 >> 1)	[6]
(a >> 1) & (b << 3)	[7]

Problem 2: (12points)

Suppose a **64-bit little endian** machine has the following memory and register status. ($1^2=1$)

Memory status

Address	Low	High
0x2010	0x00 0x00 0x00 0x00 0x00 0x00 0x18 0x20	
0x2018	0x30 0x20 0x10 0x00 0x00 0x00 0x00 0x00	
0x2020	0xff 0xff 0xff 0xff 0x00 0x00 0x00 0x00	
0x2028	0x00 0xab 0xcd 0xef 0x00 0x00 0x00 0x00	
0x2030	0xff 0xff 0xff 0xff 0xab 0xcd 0xef 0xff	
0x2038	0xff 0xff 0xff 0xff 0x11 0x22 0x33 0x44	

Register status

Register	Hex Value
%rax	0x00000000 00002018
%rbx	0x00000000 00000002
%rcx	0xffffffff ffffffff
%rdx	0xffffffff abababab
%rsi	0x00000000 00000001
%rsp	0x00000000 00002038

The following instructions are executed **sequentially**. (HINT: there should be no interference between instructions.)

	Operation
1	addq \$0x2018,%rbx
2	movq \$2018,0x2018(,%rcx,8)
3	imulq \$16,0x2018
4	sarq \$0x1,8(%rax)
5	leaq 0x2020(,0x1,8),%rdx
6	pushq %rsi

After executing six instructions above, please fill in the blanks below. For 'Hex Value', write in **8-byte hex value**. For example, if the value on the address from 0x2010 to 0x2017 are 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x18 0x20, the hex value should be 0x2018000000000000

Address	Hex Value
0x2010 ~ 0x2017	[1]
0x2018 ~ 0x201f	[2]
0x2020 ~ 0x2027	[3]
0x2028 ~ 0x202f	[4]
0x2030 ~ 0x2037	[5]
0x2038 ~ 0x203f	[6]

Register	Hex Value
%rax	[7]
%rbx	[8]
%rcx	[9]
%rdx	[10]
%rsi	[11]
%rsp	[12]

Problem 3: (16points)

Please answer the following questions according to the definition of heterogeneous data structures. (**NOTE:** the size of data types in x86-64 is shown in the Figure 3.1 in ICS book.)

```
struct s {
    char raw[0];
    char name;
    union u {
        char (*arr)[8][2];
        short num;
        char c;
    } ele [3];
    char alias[1];
    int (*callback) (int, int, double *);
} s1;
```

1. Fill in the following blocks. (please represent address with **Hex**) (8')

Representation	x86-64
sizeof(s1.raw)	[1]
sizeof(s1.ele[0])	[2]
sizeof(s1)	[3]
&s1	0x555555755040
&(s1.ele[1].c)	[4]
&(s1.callback)	[5]
&(s1.raw[32])	[6]
&(s1.alias[0])	[7]
&(s1.ele[-1])	[8]

2. How many bytes are **WASTED** in **struct s** under x86-64? Explain your solution. (4')
3. Data alignment is good for memory system performance. However, those added padding bloats the size of the data structures and may cause unfavorable results when memory footprint is critical. GCC provides facility to avoid structure padding. The GCC document on "`__attribute__((packed))`" says "This attribute, attached to struct or union type definition, specifies that each member of the structure or union is placed to minimize the memory required." **NOTE: It just avoids padding and DOES NOT rearrange the fields.**

```
struct s {
    /* remains the same */
} __attribute__((packed)) s1;
```

Fill in the blocks based on the modified definition of `struct s`. (4')

Representation	x86-64
<code>sizeof(s1.ele[0])</code>	[1]
<code>sizeof(s1)</code>	[2]
<code>&s1</code>	0x555555755040
<code>&(s1.ele[1].c)</code>	[3]
<code>&(s1.callback)</code>	[4]

Problem 4: (16 points)

The following figure shows the floating-point format we designed for the exam. Except for the field length, it's the same as the IEEE 754 single-precision format you have learned in the class.

s (1bit)	exp (11bits)	frac. (4bits)
----------	--------------	---------------

1. Fill the blanks with proper values. (2'*4=8')

- Normalized: $(-1)^{sign} \times (1.fraction) \times 2^{exp-bias}$, where bias= [1] ;
- `-Infinite` ($-\infty$) (in binary form): [2] ;
- Largest Positive Denormalized Value (in binary form): [3] ;
- Smallest Positive Normalized Value (in binary form): [4] ;

2. Convert the number $(-25.6)_{10}$ into the float representation we designed above. (4')

3. Assume we use the float representation we designed above and IEEE round-to-even mode to do the approximation. Please calculate the addition: $(1\ 10000001101\ 0000)_2 + (1\ 10000001100\ 1111)_2$ and write the answer in binary. (4')

Problem 5: (22points)

One of TA wrote a simple C program and some assembly code is provided. Suppose both of them are executed on a **64-bit little-endian** machine. Please read the code and answer the following questions.

<pre>long transform(long a, long b){ if (a > <u>[1]</u>) a = a + 2; else a = a * 2; long result = (a + b) >> 1; switch(result){ case 1: result = a * result + b; break; case <u>[2]</u> : result++; case <u>[3]</u> : result = a + b; case 4: result = <u>[4]</u> break; case 2: result = b >> 2; break; default: result = result * 3 - 2; } return result; } int main(){ long A[3][5]; for (long i=0; i < 3; i++) for (long j=0; j < 5; j++) A[i][j] = i+j; int *B = (int *)A[1]; long res = transform(A[0][2], A[2][0]); return res; }</pre>	<pre>main: transform: pushq %rbp movq %rsp,%rbp movq %rdi,-8(%rbp) movq %rsi,-16(%rbp) cmpq \$5,-8(%rbp) jle <u>[5]</u> movq -8(%rbp),%rax addq \$2,%rax movq %rax,-8(%rbp) jmp .L3 .L2: movq -8(%rbp),%rax shlq \$1,%rax movq %rax,-8(%rbp) .L3: movq -8(%rbp),%rax movq <u>[6]</u>,%rcx addq %rcx,%rax sarq \$1,%rax movq %rax,-24(%rbp) movq -24(%rbp),%rax decq %rax movq %rax,%rcx subq \$4,%rcx movq %rax,-32(%rbp) movq %rcx,-40(%rbp) ja .L12 leaq .L0,%rax movq -32(%rbp),%rcx movq (%rax,%rcx,8),%rdx jmpq <u>[7]</u></pre>
---	--

<pre> .L4: movq -8(%rbp), %rax imulq -24(%rbp), %rax addq -16(%rbp), %rax movq %rax, -24(%rbp) jmp <u> [8] </u> .L5: movq -24(%rbp), %rax addq \$1, %rax movq %rax, -24(%rbp) .L6: movq -8(%rbp), %rax addq -16(%rbp), %rax movq %rax, -24(%rbp) .L7: cmpq \$4, -8(%rbp) jge .L9 movq -24(%rbp), %rax subq -8(%rbp), %rax movq %rax, -48(%rbp) jmp .L10 .L9: movq -8(%rbp), %rax subq -24(%rbp), %rax movq %rax, -48(%rbp) </pre>	<pre> .L10: movq -48(%rbp), %rax movq %rax, -24(%rbp) jmp .L13 .L11: movq -16(%rbp), %rax <u> [9] </u> \$2, %rax movq %rax, -24(%rbp) jmp .L13 .L12: movq -24(%rbp), %rax imulq \$3, %rax subq \$2, %rax movq %rax, -24(%rbp) .L13: <u> [10] </u> popq %rbp retq .section .rodata .align 8 .L0: .quad .L4 .quad .L11 .quad .L5 .quad .L7 .quad .L6 </pre>
--	---

- Suppose the address of **A** in the main function is **0x802018**, what's the value of following expressions (1'*5=5')

Operation	value
&(A[i][j])	[1]
A[2]	[2]
B[0]	[3]
B[1]	[4]
B[2]	[5]

- Please fill in the blanks within C and assembly code. (1.5' * 10)
NOTE: no more than one instruction/statement per blank. If you think nothing is required to write, please write NONE.
- What is the value of **variable res** in the **main function**? (2')

Problem 6: (20points)

One of TAs of ICS wrote a **buggy program**. The following C code and assembly code are executed on a **64-bit little endian** machine.

<pre> long f(char* arr, unsigned long i) { if (i >= 20) return 0; *(long *)&arr[i] = i % 2; *(long *)&arr[i + 1] = i + 2; *(long *)&arr[i + 2] = i + 5; *(long *)&arr[i + 3] = i % 20; return i + f(arr, i + 4); } </pre>	<pre> long buggy(unsigned long s){ char arr[0x40]; return f(arr + s, 0); } int main(){ unsigned long x = 0; printf("%ld", buggy(x)); return 0; } </pre>
--	--

```

400506 <f>:
400506:    55                push    %rbp
400507:    48 89 e5          mov     %rsp,%rbp
40050a:    53                push    %rbx
40050b:    48 83 ec 18       sub     $0x18,%rsp
40050f:    48 89 7d e8       mov     %rdi,-0x18(%rbp)
400513:    48 89 f3          mov     %rsi,%rbx
400516:    48 83 fb 27       cmp     $0x13,%rbx
40051a:    76 07            jbe     400523 <f+0x1d>
40051c:    b8 00 00 00 00    mov     $0x0,%eax
400521:    eb 57            jmp     40057a <f+0x74>
[1] :    .. .. .
// NOTE: assign value to arr[i] ~ arr[i+3]
.. .. .
400564:    48 8d 53 04       lea     0x4(%rbx),%rdx
400568:    48 8b 45 e8       mov     -0x18(%rbp),%rax
40056c:    .. .. .          [2]
40056f:    .. .. .          [3]
400572:    e8 8f ff ff ff    callq  400506 <f>
.. .. .
// NOTE: prepare return value
4005xx:    .. .. .          [4]
// NOTE: restore stack and registers
4005xx:    .. .. .          [5]
4005xx:    .. .. .          [6]
4005xx:    .. .. .          [7]
400580:    c3                retq

```

400581 <buggy>:

```
400581:    55                push    %rbp
400582:    48 89 e5          mov     %rsp,%rbp
400585:    .. .. ..         _____ [8]
400589:    48 89 fa          mov     %rdi,%rdx
40058c:    48 8d 45 c0        lea     -0x40(%rbp),%rax
400590:    48 01 d0          add     %rdx,%rax
400593:    be 00 00 00 00    mov     $0x0,%esi
400598:    48 89 c7          mov     %rax,%rdi
40059b:    e8 66 ff ff ff    callq  400506 <f>
4005a0:    c9                leaveq  %rdi
4005a1:    c3                retq
```

4005a2 <main>:

```
4005a2:    55                push    %rbp
4005a3:    48 89 e5          mov     %rsp,%rbp
// prepare arguments
.. .. ..
4005b9:    e8 c3 ff ff ff    callq  400581 <buggy>
4005be:    48 89 c6          mov     %rax,%rsi
4005c1:    bf 64 06 40 00    mov     $0x400664,%edi
4005c6:    b8 00 00 00 00    mov     $0x0,%eax
4005cb:    e8 10 fe ff ff    callq  4003e0 <printf@plt>
4005d0:    b8 00 00 00 00    mov     $0x0,%eax
4005d5:    c9                leaveq  %rdi
4005d6:    c3                retq
```

1. Fill in the blank in above **assembly code**. (8'). **NOTE: DO NOT fill more than one instructions or statements in each blank.**
2. What is the **output** of this program? (3')
3. Assume we can modify the '0' in **long x = 0** in **main()** function to give different initial value to variable **x**. **NOTE: DO NOT** consider **arr + s** overflows the max value of **unsigned long**.
 - a. What's the range of value of **x** that do not affect the normal execution and states of this program? (4')
 - b. Now a student wants to trigger buffer overflow to return to a strange address from **buggy()**. Figure out all the possible addresses between **0x100000** and **0x7fff00000000** that he can return to (except **main()**), and show their corresponding value of **x**. (5')