CS230 System Programming
Midterm Exam
Wednesday, 23 Oct 2013

Name:

Student Number:

Problems	Points
1 (16pts)	
2 (12pts)	
3 (10pts)	
4 (10pts)	
5 (10pts)	
6 (14pts)	
7 (6pts)	
8 (9pts)	
9 (5pts)	
10 (8pts)	
Total	

IMPORTANT: Explain your answer briefly. Do not just write a short answer or fill the assembly code.

1. [integer data representation]
   M[A] : the memory content of address A

   A. [2pts] x86 is a little endian machine. Suppose %eax = 0x12345678 and %edx = 0x1000.

   After an instruction "movl %eax, 0(%edx)" is executed, what are the following memory contents ? (one byte for each line)

   M[0x1000] = _____
   M[0x1001] = _____
   M[0x1002] = _____
   M[0x1003] = _____

   B. [2pts] Please, answer the values of z and w in hexadecimal number at the end of the following C code fragment.

```
…
int x = 0x80000000;
unsigned y = 0x80000000;
int z,
unsigned w,

z = x >> 1;
w = y >> 1;
..
```

   C. [2pts] When the number of bits of an integer register is *w*, write the maximum and minimum values represented with two's complement type.

   D. [2pts] Write the relation between the left and right constants in C. (mark with >, <, or == symbol)

```
-1    _____   0U
2147483647U _____   -2147483647-1
2147483647  ____ (int) 2147483647U
```

E. [2pts] Please, mark True or False of the following statement for unsigned and two's complement formats. If your answer is False, then write a counter example for the values of U and V.

U > 0 ➔ U + V > V

F. The following code fragment has a potential vulnerability.

```
void* copy_elements(void *ele_src[], int ele_cnt, size_t ele_size) {
    /*
     * Allocate buffer for ele_cnt objects, each of ele_size bytes
     * and copy from locations designated by ele_src
     */
    void *result = malloc(ele_cnt * ele_size);
    if (result == NULL)
             /* malloc failed */
             return NULL;
    void *next = result;
    int i;
    for (i = 0; i < ele_cnt; i++) {
        /* Copy object i to destination */
        memcpy(next, ele_src[i], ele_size);
             /* Move pointer to next memory region */
             next += ele_size;
    }
    return result;
}
```

E-1 [2pts] Please, write possible values for ele_cnt and ele_size to crash the application/system.

E-1 [4pts] Please, write extra C codes to add before the malloc call to prevent the crash.

2. [floating point data representation]

IEEE floating point format: [s][exp][frac]

A. [2pts] The IEEE single precision floating point format has 8 bits for exp, and 23 bits for fraction. The bias is 127. Write a binary content for the 32 bit floating number for -15213 (11101101101101)

B. [2pts] What is the decimal value of the following floating point number?

s = 0
exp = 000…0
frac = 100000….000

C. [2pts] What is the result of the following floating point operation in the IEEE float format?
C-1 1.0 / 0.0 :
C-2 sqrt (-1) :

D. [2pts] Write the rounded binary numbers for the following values. They should be rounded to nearest 1/2 (2 bits fright of binary point, and must use "round-to-even" rule.

10.00011 => _____
10.00110 => _____
10.11100 => _____
10.10100 => _____

E. [4pts] Write whether the following condition is always True or False. Explain your argument for the answers.

float f;
double d;
…

Is f == (float)((double) f) ?
Is d == (double)((float)d) ?

3. [Condition code]
In x86, there are four condition codes: CF (carry), ZF (zero), SF (sign), OF (overflow)

A. [4pts] “setl” (set less than) instruction checks whether (SF^OF) is true. Explain why just using “SF” is not enough for the “less than” condition.

B. “setb” (set below) instruction checks whether “CF” is true.

B-1 [2pts] What is the semantic difference between setb and setl?

B-1 [4pts] Explain why checking CF is enough for “below” condition.

4. [Control flow]

A. [3pts] Fill the assembly codes for the following if statement with conditional move instruction (use "cmovg", which stands for conditional move greater than) . x is in %edi, and y is in %esi.

```
int absdiff(int x, int y) {
    int result;
    if (x > y) {
        result = x-y;
    } else {
        result = y-x;
    }
    return result;
}
```

```
absdiff:
   movl        %edi, %edx
   subl        %esi, %edx          # tval = x-y
   movl        %esi, %eax
   subl        %edi, %eax          # result = y-x
   _____________________           # Compare x:y
   _____________________           # If >, result = tval
   ret
```

B. [4pts] Explain why using conditional move for the following cases may be risky or incorrect.

B-1 val = p ? *p : 0;

B-2 val = x > 0 ? x*=7 : x+=3;

C. [3pts] Convert the following for-statement in C to a goto version.

```
#define WSIZE 8*sizeof(int)
int pcount_for(unsigned x) {
  int i;
  int result = 0;
  for (i = 0; i < WSIZE; i++) {
    unsigned mask = 1 << i;
    result += (x & mask) != 0;
  }
  return result;
}
```

```
int pcount_for_gt(unsigned x) {
  int i;
  int result = 0;
  _________;

 loop:
  {
    unsigned mask = 1 << i;
    result += (x & mask) != 0;
  }
  ___________;

  __________

 done:
  return result;
}
```

5. [switch statement]
   A. [6pts] Compare and contrast two possible implementations of a switch statement in C. One possible implementation is to convert the switch statement to nested "if" statements, and the other is to use an indirect jump. What are the pros and cons of the two approaches?

   B. [4pts] Complete the following assembly code for the switch statement.

```
long switch_eg(long x, long y, long z)
{
    long w = 1;
    switch(x) {
      . . .
    }
    return w;
}
```

Jump table

```
.section    .rodata
  .align 4
.L7:
  .long     .L2 # x = 0
  .long     .L3 # x = 1
  .long     .L4 # x = 2
  .long     .L5 # x = 3
  .long     .L2 # x = 4
  .long     .L6 # x = 5
  .long     .L6 # x = 6
```

```
switch_eg:
    pushl       %ebp                    # Setup
    movl        %esp, %ebp              # Setup
    movl        8(%ebp), %eax           # eax = x
    ____________________
    _____       .L2
    ____________________
```

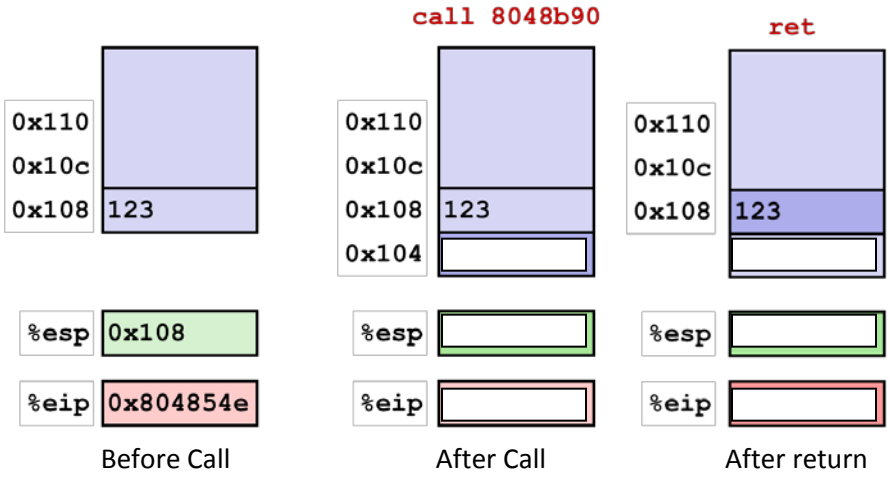
6. [Procedure call]

A. [3pts] Fill the stack and register values for the following call and return instruction.

```
804854e:  e8 3d 06 00 00   call   8048b90 <main>
8048553:  50               pushl  %eax
```

…

```
8048591:  c3               ret
```



B. [3pts] Fill the following assembly code for the swap function. Please, explain what each line you filled is doing.

i

```
void swap(int *xp, int *yp)
{
  int t0 = *xp;
  int t1 = *yp;
  *xp = t1;
  *yp = t0;
}
```

```
swap:
        _____________________
        _____________________
        _____________________
movl    8(%ebp), %edx
movl    12(%ebp), %ecx
movl    (%edx), %ebx
movl    (%ecx), %eax
movl    %eax, (%edx)
movl    %ebx, (%ecx)
        _____________________
        _____________________
ret
```

C. [5pts] Explain why the compiler and architecture define and follow the convention for caller/callee save registers .

D. [3pts] Fill the following assembly code for the recurcive pcount_r function.

```
/* Recursive popcount */
int pcount_r(unsigned x) {
  if (x == 0)
    return 0;
  else return
    (x & 1) + pcount_r(x >> 1);
}
```

```
pcount_r:
        _____________________
        _____________________
        _____________________
        subl      $4, %esp
        movl      8(%ebp), %ebx
        movl      $0, %eax
        testl     %ebx, %ebx
        je     .L3
        movl      %ebx, %eax
        shrl      %eax
        _____________________
        call      pcount_r
        movl      %ebx, %edx
        andl      $1, %edx
        leal      (%edx,____), ____
.L3:
        _____________________
        _____________________
        _____________________
        ret
```

7. [Procedure call 64 bit]

A. [3pts] Fill the following assembly code for the swap_l function.

```
void swap_l(long *xp, long *yp)
{
  long t0 = *xp;
  long t1 = *yp;
  *xp = t1;
  *yp = t0;
}
```

```
swap:
        movq      _____, %rdx
        movq      _____ , %rax
        movq      %rax, (%rdi)
        movq      %rdx, (%rsi)
        ret
```

[3pts] What is the main difference between 32 bit call convention and 64 bit call convention in x86?

8. [Array]

A. [3pts] Suppose “int array [1024]” declared in a C code and &a[0] = 0x1000. Answer the following questions.

Type of “array” ? :
Value of “array” ? :
Value of “array + 2” ? :
Suppose “int *ap = array”, value of “&(ap[4])”?

B. [3pts] Fill the assembly code for the following nest array example.

```
int get_pgh_digit
  (int index, int dig)
{
  return pgh[index][dig];
}
```

```
  movl      8(%ebp), %eax                    # index
  leal      _________ , %eax                 # 5*index
  addl      12(%ebp), %eax                   # 5*index+dig
  movl      _________, %eax                  # offset 4*(5*index+dig)
```

C. [3pts] Fill the assembly code for the following multi-level array example.

```
int get_univ_digit
  (int index, int dig)
{
  return univ[index][dig];
}
```

```
    movl      8(%ebp), %eax                    # index
    movl      __________ , %edx            # p = univ[index]
    movl      12(%ebp), %eax                   # dig
    movl      __________ , %eax            # p[dig]
```

9. [5pts] What is the difference between “structure” and “union” in C. Take an example that “union” can be useful.

10. [Caches]

A. [4pts] For a cache with 64B capacity, answer the number of sets for the following configurations

Block size = 16, associativity = 8 set = ?
Block size = 64, associativity = 1 set = ?

B. [4pts] How many cache misses occur when the block size is 64B for the following C code. Assume the array a is not cached before the execution, but the other variables are in the cache. Please, explain your reasoning for the answer.

```
int sum_array(double a[32*1024])
{
    int i;
    double sum = 0;
    for (i = 0; i < 32*1024; i++)
            sum += a[i];
    return sum;
}
```