Introduction to Computer Systems 2011
First Midterm Examination

Name_____ Student No._____ Score_____

Problem 1: (20 points)

1. Consider the following C program

```
short sy = -4;
unsigned short usy = sy
int y = sy;
int x = -30;
unsigned int ux = x;
```

Assume we are running code on a 6-bit machine using two's complement arithmetic for signed integers. Assume also that right shifts of signed values are performed arithmetically. A "short" integer is encoded using 3 bits. Fill in the empty boxes in the table below. The following definitions are used in the table: (2' * 8)

Expression	Binary Representation
-9	[1]
ux	[2]
y	[3]
usy	[4]
(x >> 2) << 1	[5]
x + y	[6]
x & !y	[7]
UMax	[8]
TMax	[9]
-TMin	[10]

Problem 2: (24points)

Implement the following 2 functions and make their functionalities be the same with the description. **The coding rules are also same as Lab1**. (If you think there is nothing to do, just fill the blank with '--')

1. absValue (10' + 2')

```
/* [absValue]
 *  - Calculate the absolute value of x
 *
 * Example: absValue(5) = 5, absValue(-29) = 29
 * Legal ops: ~ & ^ | - << >>
 */

int absValue (int x) {
    /* Please fill your code in the answer paper */

    return ret;
}
```

2. swapBits (2' * 6)

```
/* [swapBits]
 *  - Do swap operation on input x:
 *    Swap n bits started from position i with n bits started
 *    from position j
 * Explain for i and j:
 *    i=1 means the second bit from the right;
 *    j=5 means the sixth bit from the right
 *
 * Examples: swapBits(0x2F,1,5,3) = 0xE3,
 * Legal ops: ~ & ^ | - << >>
 */

int swapBits (int x, int i, int j, int n) {
    int mask= (1 << __[1]__ ) - 1;
    int xorTemp = ((x >> __[2]__) ^ (x >> __[3]__)) & __[4]__;
    return x __[5]__ ((xorTemp << i) __[6]__ (xorTemp << j));
}
```

Problem 3: (12points)

Suppose a 32-bit little endian machine has the following memory and register status. Fill in the blanks using **4** byte size and **hex**.
Each operation take effect on the status of memory and register
(1′ * 12 = 12′)

Memory status:

Address	Value
0x100	0xffffffff
0x104	0x87654321
0x108	0x00000001
0x10c	0x00000002
0x110	0x22347688
0x114	0x12345678

Register status:

Register	Value
%eax	0x87654421
%ebx	0x00000104
%ecx	0x00000002
%edx	0x00000008

Fill in the blanks

Operation	Destination	Value
subl (%ebx),%eax	%eax	0x100
incl 4(%eax)	**[1]**	**[2]**
decl %ecx	**[3]**	**[4]**
imull $4,0x100(%edx,%ecx,4)	**[5]**	**[6]**
notl (%eax,%edx)	**[7]**	**[8]**
andl (%eax,%ecx,8),%eax	**[9]**	**[10]**
leal 9(%eax,%ecx,2),%edx	**[11]**	**[12]**

Problem 4: (26points)

Suppose the following C code and assembly code are executed on a 32-bit **little endian** machine.

C code:

```
int mystery(int i) {
   if ( i != 0 )
      return i + mystery(i-1);
   return i;
}

int main(void) {
   return mystery(10);
}
```

Assembly code:

```
      08048324 <mystery>:
Line1  8048324: 55                 push   %ebp
Line2         : 89 e5              mov    %esp,%ebp
Line3         : 83 ec 08           sub    $0x8,%esp
Line4 ___[1]__: 83 7d 08 00 00     cmpl   0x0,0x8(%ebp)
Line5         : 74 18              je     8048348 <mystery+0x24>
Line6         : 8b 45 08           mov    0x8(%ebp), %eax
Line7         : 83 e8 01           sub    $0x1, %eax
Line8         : 89 04 24           mov    %eax, __[2]__
Line9         : e8 e6 ff ff ff     call   8048324 <mystery>
Line10        : 8b 55 08           mov    0x8(%ebp),%edx
Line11        : 01 2c              add    __[3]__,%edx
Line12        : 89 55 fc           mov    %edx,0xfffffc(%ebp)
Line13        : eb 06              jmp    804834e <mystery+0x2a>
Line14 8048348: 8b 45 08           mov    0x8(%ebp), %eax
Line15        : 89 45 fc           mov    %eax, 0xfffffffc(%ebp)
Line16 804834e: 8b 45  fc          mov    __[4]__, %eax
Line17        : c9                 leave
Line18        : c3                 ret
```

```
      08048353 <main>:
Line19        : 8d 4c 24 04        lea    0x4(%esp),%ecx
Line20        : 83 e4  f0          and    $0xfffffff0,%esp
Line21        : ff 71 fc           pushl  $0xfffffffc(%ecx)
Line22        : 55                 push   %ebp
Line23        : 89 e5              mov    %esp,%ebp
```

```
Line24          : 51                    pub    %ecx
Line25          : 83 ec 04              sub    $0x4, %esp
Line26          : c7 04 24 0a 00        mov    __[5]__,(%esp)
                  00 00
Line27          : e8 b4 ff ff ff        call   8048324 <mystery>
Line28          : 83 c4 04              add    $0x4,%esp
Line29          : 59                    pop    %ecx
Line30          : 5d                    pop    %ebp
Line31          : 8d 61 fc              lea    0xfffffc(%ecx),%esp
Line32          : c3                    ret
```

1. Please fill in the blanks within assembly code (**Note**: the blanks [1] and is the start address of the instructions). (2'*5 = 10')
2. Suppose the value of **%ebp** is **0xbf8ce638** and the value of **%esp** is **0xbf8ce5cc** before the instruction **Line22** executed, then please answer the following questions: (13')
   1) What are the value of **%ebp** and **%esp** after the instruction **Line3** executed? (2'*2)
   2) What's the meaning of instruction **Line8**, **Line11** and **Line16**? (3'*3)
3. Explain where is the return value for main function? (3')

Problem 5: (18points)

Suppose the following C code and assembly code are executed on a 32-bit **little endian** machine. Read the code and answer the following question:

C code:

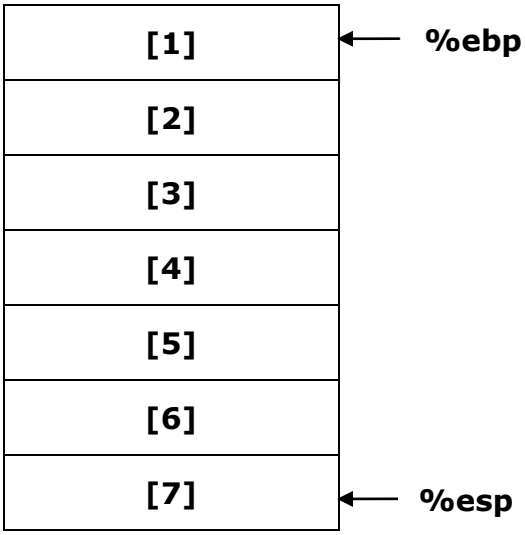
```
int byte_abs(int * a, int b) {
    char* ptr = (char*)a;
    while(b > 0){
        int temp = (int)(*ptr);
        if(temp < 0)
            ptr[0] = (char)(-temp);
        ptr += 4;
        b --;
    }
}
int main(void) {
    int data[2] = {0x12345678, 0x9ABCDEF};
    int flag = 0x457823AB;
    byte_abs(data, 3);
    return 0;
}
```

Assembly Code:

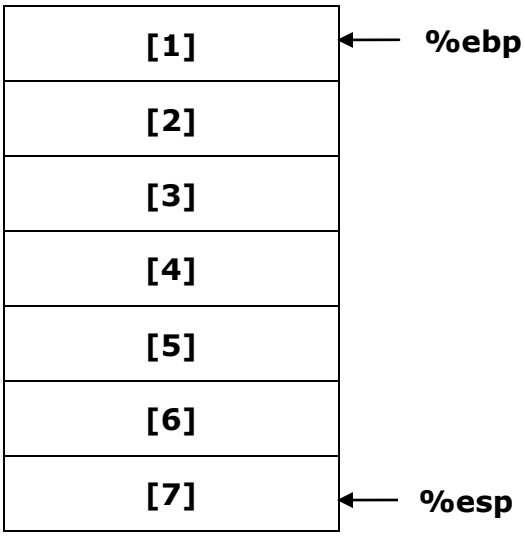
```
main:
8048390:      8d 4c 24 04            leal   4(%esp), %ecx
8048394:      83 e4 f0               andl   $fffffff0, %esp
8048397:      ff 71 fc               pushl  -0x4(%ecx)
804839a:      55                     pushl  %ebp
804839b:      89 e5                  movl   %esp, %ebp
804839d:      51                     pushl  %ecx
804839e:      83 ec 14               subl   $0x14, %esp
80483a1:      c7 45 f0 78 56 34 12   movl   $0x12345678, -0x10(%ebp)
80483a8:      c7 45 ____[1]____      movl   $0x9ABCDEF, -0xc(%ebp)
80483bf:      c7 45 ____[2]____      movl   $0x457823AB, -0x8(%ebp)
80483b6:      c7 44 24 04 03 00 00   movl   $0x3, 0x4(%esp)
80483bd:      00
80483be:      8d 45 f0               leal   -0x10(%ebp), %eax
80483c1:      89 04 24               movl   %eax, (%esp)
80483c4:      e8 8b ff ff ff         call   <byte_abs>
80483c9:      83 c4 14               addl   $0x14, %esp
80483cc:      59                     popl   %ecx
80483cd:      5d                     popl   %ebp
80483ce:      8d 61 fc               leal   -0x4(%ecx), %esp
80483d1:      c3                     ret
```

1. Please fill the blanks in the assembly code with the correct **byte ordering**.(2'*2=4')

2. Suppose the %ebp is **0xbfb88bbc** before executing the instruction at **0x804839b** ("movl %esp, %ebp"). Please draw a diagram to show the memory value between **%ebp** and **%esp** **_BEFORE_** executing the instruction at **0x80483c4** ("call byte_abs"). If the value can't be determined, fill '**--**' instead. (1'*7=7')



3. We have the same assumption as question 2. Please draw a diagram to show the memory value between **%ebp** and **%esp** **_AFTER_** executing the instruction at **0x80483cb** ("call byte_abs"). If the value can't be determined, fill '--' instead. (1'*7=7')



Solution

Problem 1: (20 points)

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

Problem 1: (24 points)

1

2

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

Problem 3: (12 points)

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

Problem 4: (26 points)

1 [1] [2] [3]

[4] [5]

2 1)

2)

3

Problem 5: (16 points)

1 [1]

[2]

2 [1]

[2]

[3]

[4]

[5]

[6]

[7]

3 [1]

[2]

[3]

[4]

[5]

[6]

[7]