复旦大学软件学院
2010～2011 学年第二学期期末考试试卷

Problem 1: (16 points)

1. Consider the following C code and assembly code for a strange but simple function on x86(32-bit machine) platform:

```
int lol(int a, int b)
{
    switch(a){
      case 210:
        b *= 13;
        ___[1]___;
      case 213:
        b = 18243;
      case 214:
        b *= b;
        ___[2]___;
      case 216:
      case 218:
        b -= a;
        ___[3]___;
      case 219:
        b += 17;
        ___[4]___;
      default:
        b = -9;
    }
    return b;
}
```

```
<lol>:
0x8048330: push  %ebp
0x8048331: mov   %esp,%ebp
0x8048333: mov   0x8(%ebp),%eax
0x8048336: mov   0xc(%ebp),%ecx
0x8048339: lea   0xffffff2e(%eax),%edx
0x804833f: cmp   $0x8,%edx
0x8048342: jbe   0x8048350
0x8048345: mov   $0xfffffff7,%eax
0x804834a: ret
0x8048350: jmp   *0x8048470(,%edx,4)
0x8048358: sub   %eax,%ecx
0x804835a: mov   %ecx,%eax
0x8048360: ret
0x8048361: mov   %ecx,%eax
0x8048363: imul  %ecx,%eax
0x8048367: ret
0x8048369: mov   $0x13d63b89,%eax
0x804836e: ret
0x8048370: lea   (%ecx,%ecx,2),%eax
0x8048373: lea   (%ecx,%eax,4),%eax
0x8048376: ret
```

1) For each case in the switch block which should have a "**break**", write it on the corresponding blank line. (2' * 4 = 8')
(**Hint**: 0xffffff2e = -210 and 0x13d63b89 = 18243 * 18243)

2) Using the available information, fill in the jump table below. (2' * 4 = 8')

```
Jump table entries:
   0x8048470:  ___[5]___
   0x8048474:  0x08048345
   0x8048478:  0x08048345
   0x804847c:  ___[6]___
   0x8048480:  ___[7]___
   0x8048484:  0x08048345
   0x8048488:  ___[8]___
   0x804848c:  0x08048345
   0x8048490:  0x08048358
   0x8048494:  0x00000000
```

Problem 2: (27points)

Suppose the following C code and assembly code are executed on a 32-bit **little endian** machine. Please read the following code carefully and answer questions.

Assembly Code:

```
main:
  3d:   8d 4c 24 04             lea    0x4(%esp),%ecx
  41:   83 e4 f0                and    $0xfffffff0,%esp
  44:   ff 71 fc                pushl  -0x4(%ecx)
  47:   55                      push   %ebp
  48:   89 e5                   mov    %esp,%ebp
  4a:   51                      push   %ecx
  4b:   83 ec 28                sub    $0x28,%esp
  4e:   c6 45 ec 12             movb   $0x12,-0x14(%ebp)
  52:   c6 45 ed 34             movb   $0x34,-0x13(%ebp)
  56:   c6 45 ee 56             movb   $0x56,-0x12(%ebp)
  5a:   c6 45 ef 78             movb   $0x78,-0x11(%ebp)
  5e:   66 c7 45 e8 ___[1]___   movw   $0x1234,-0x18(%ebp)
  64:   66 c7 45 ea ___[2]___   movw   $0x5678,-0x16(%ebp)
  6a:   c7 45 f0 ___[3]___      movl   $0x78563412,-0x10(%ebp)
  71:   c7 45 f4 ___[4]___      movl   $0x12345678,-0xc(%ebp)
  78:   8d 45 ec                lea    ___[5]___,%eax
  7b:   89 45 f8                mov    %eax,-0x8(%ebp)
  7e:   8d 45 e8                lea    -0x18(%ebp),%eax
  81:   89 44 24 04             mov    %eax,0x4(%esp)
  85:   8b 45 f8                mov    -0x8(%ebp),%eax
  88:   89 04 24                mov    %eax,(%esp)
  8b:   e8 fc ff ff ff          call   8c <main+0x4f>
  90:   89 c2                   mov    %eax,%edx
  92:   8b 45 f8                mov    -0x8(%ebp),%eax
  95:   89 10                   mov    %edx,(%eax)
  97:   83 c4 28                add    $0x28,%esp
  9a:   59                      pop    %ecx
  9b:   5d                      pop    %ebp
  9c:   8d 61 fc                lea    -0x4(%ecx),%esp
  9f:   c3                      ret
```

C Code:

```
int func(int *addr, int* save)
{
    int x = *addr;
    int y = *(addr + 1);

    y = x & y;
    *save = y;
    x = *(addr + 2);
    y = x | y;

    return y;
}

int main(void)
{
    char a[] = {0x12,0x34,0x56,0x78};
    short b[] = {0x1234,0x5678};
    int c = 0x78563412;
    int d = 0x12345678;
    int *e = (int*)a;

    *e = func(e, (int*)b);
}
```

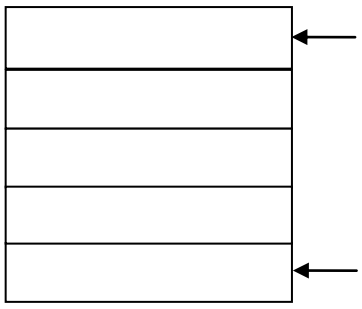
1. Please fill the blanks in the assembly code according to C Code with the correct **byte ordering**. (1' * 5 = 5')
2. According to the C code, please fill the following blanks with "<", "=" or ">". (2' * 3 = 6')

   `*((int*)a)` **___[1]___** `*((int*)b)`

   `*((int*)a)` **___[2]___** `c`

   `*((int*)a)` **___[3]___** `d`

3. Just before the call instruction in main function has been executed, the **%esp** value is **0xbfaa42c0**. Please fill the following stack diagram with correct value in hex format. (NOTE: If the value is undetermined, fill "-" instead) (2' * 4 = 8')



4. Suppose the statement "***e = func(e, (int *)b);**" in main() function has been executed, please fill the following table. (2' * 4 = 8')

a[0]	a[2]	b[0]	b[1]
__[1]__	__[2]__	__[3]__	__[4]__

Problem 3: (27 points)

1. A software school student decided to write a dynamic memory allocator for an **x86-64** machine in which each block has the following form:

Header	Id String	**Payload**	Footer

Where

- Header is a **4** byte header
- Id string is an **8** byte string
- Payload is arbitrary size, including padding
- Footer is a **4** byte footer

Assume the student wants to print the Id string with the following function

1) Where bp points to the beginning of the **Payload** and is aligned to 0 modulo 8. Write **"YES"** to each of the following 4 macros if it will correctly print the Id string, else write **"NO"** (2' * 4 = 8')

a) `#define GET_ID(bp) ((char *)(((long)bp)-8))` ___**[1]**___
b) `#define GET_ID(bp) ((char *)(((int)bp)-8))` ___**[2]**___
c) `#define GET_ID(bp) ((char *)(((long *)bp)-1))` ___**[3]**___
d) `#define GET_ID(bp) ((char *)(((int **)bp)-2))` ___**[4]**___

2) The student uses **Header** to encode the block size (including all padding). If we impose a **one-word** alignment (8 bytes) constraint, then how many bits of **Header** can be used to indicate whether the block is allocated or free? (2')

2. Assume we use the **buddy** system to manage memory. In the buddy system, a block of memory size $\mathbf{2^i}$, is called an **i-block**, and the **i-list** consists of starting addresses of **free** i-blocks. Suppose that the total memory size in our computer system is **1664** bytes, whose starting address is **0**. (18')

1) Originally, all these 1664 bytes are free, and are represented as **3** free blocks, and sizes are 1024 bytes, 512 bytes and 128 bytes. What are the originally 3 free blocks? (2' * 3 = 6')

___**[1]**___-block , ___**[2]**___-block , ___**[3]**___-block

2) Suppose we have to issue a request for allocating 100 bytes, then we should check __**[4]**__-list first, and find there are/is __**[5]**__ free block(s) in that list. (2' * 2 = 4')

3) If your answer to **[5]** is 0, then we will split the free block in **__[6]__**-list to satisfy the 100 bytes allocating request; **else**, we don't need to split any free blocks, and please write NULL to **[6]**. (2')

4) After issue the 100 bytes request, suppose the following memory requests are issued sequentially:

   1. malloc(128)
   2. malloc(28)
   3. malloc(56)
   4. malloc(110)
   5. malloc(60)
   6. malloc(48)
   7. malloc(230)

   Tell the number of free blocks in each of the free lists in the following table, after these memory requests are processed. (1' * 6 = 6')

Free list	**Number of free blocks in the list**
5-list	**[7]**
6-list	**[8]**
7-list	**[9]**
8-list	**[10]**
9-list	**[11]**
10-list	**[12]**

Problem 4: (30 points)

1. Please optimize the following function **func()**. The function length(x) returns the length of vector x, and the function get_element(x,y) returns the yth element of vector x. (15')

   The function func() is to calculate:

$$*dest \mathrel{+}= \sum_{i=1} (a[i-1] + a[i-1] + a[i-1] + a[i-1]) * (b[i-1] + b[i-1] + b[i-1] + b[i-1])$$

```
int get_element(int *x, y)
{
    if(y<1 || y> length(x))
        return -1;
    int ret= x[y-1];
    return ret;
}

int func(int* a, int* b, int *dest)
{
    int i,j,k;
    if(length(a) != length(b))
       return 0;
    for (i = 1; i <= length(a); i++) {
       int tempMul, val1_mul2 = 0, val2_mul2 = 0;
       int tempSum = get_element(a,i) + get_element(b,i);
       for (j=0;j<4;++j) {
          val1_mul2 = val1_mul2+get_element(a, i);
       }
       for(k=0;k<4;++k) {
          val2_mul2 = val2_mul2+get_element(b, i);
       }
       tempMul= val1_mul2 * val2_mul2;
       *dest = *dest + tempMul;
    }
    return 1;
}
```

2. Consider the following function. Assume that there is a single cache with an **8** bytes block size. "A" is a N*M array of **int** (the size of int is 4 bytes). **The array size M and L are so large that a single matrix row does not fit in the cache.** All loop indexes (i, j and k) are stored in registers, which not require any load or store instructions. The memory alignment and compiler-driven optimizations are _NOT_ considered.

```
#define N 10
#define M 20
#define L 20

int A[N][M];
int B[L];

int func(void)
{
   int ret=0, i, j, k;
   for (i = 0 ; i < N ; i ++) {
      for (j = 0 ; j < M ; j ++){
         for (k = 0 ; k < L ; k ++) {
            if (A[i][j] + B[k] > 0 )
               ret ++;
         }
      }
   }

   return ret;
}
```

1) What is the expected cache miss rate of the access to A and B? (6')

   A: ___[1]___ B: ___[2]___

2) Fill in the blanks of code below with the most cache friendly iteration order you think. (e.g. i, j, k, N, M, L) (3')

```
#define N 10
#define M 20
#define L 20

int A[N][M];
int B[L];

int func2(void)
{
   int ret=0, i, j, k;
   for (__[1]__= 0; __[1]__ < __[2]__; __[1]__ ++) {
      for (__[3]__ = 0 ; __[3]__ < __[4]__; __[3]__ ++){
         for (__[5]__ = 0 ; __[5]__ < __[6]__; __[5]__ ++) {
            if (A[i][j] + B[k] > 0 )
               ret ++;
         }
      }
   }

   return ret;
}
```

3) What is the expected cache miss rate of the access to A and B? (6')

   A: ___[1]___ B: ___[2]___

Solution

Problem 1: (16points)

1. [1] [2]

[3] [4]

[5] [6]

[7] [8]

Problem 2: (27points)

1 [1] [2]

[3] [4]

[5]

2 [1] [2] [3]

3 [1] [2]

[3] [4]

4 [1] [2]

[3] [4]

Problem 3: (27points)

1 1) [1] [2] [3] [4]

2)

2 [1] [2] [3]

[4] [5]

[6]

[7] [8] [9]

[10] [11] [12]

Problem 4: (30points)

1.

2.

1) [1] [2]

2) [1] [2]

[3] [4]

[5] [6]

3) [1] [2]