上海交通大学试卷（A卷）

（ 2012 至 2013 学年 第 2 学期 ）

班级号______________________ 学号_____________ 姓名 ___________

课程名称 计算机系统基础（1）____________________成绩 ____________

Problem 1: Floating Point (14points)

1. [1] [2]

[3] [4]

2.

3.

4.

Problem 2: X86-64 (14points)

1 [1] [2] [3]

[4] [5] [6]

[7]

Problem 3: Memory Allocation (14points)

1 1)

2)

2

3

我承诺，我将严格遵守考试纪律。

承诺人：__________

题号	**1**	**2**	**3**	**4**	**5**				
得分									
批阅人(流水阅卷教师签名处)									

Problem 4: Cache (16points)

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

2. [1] [2] [3]

   [4] [5] [6]

   [7] [8] [9]

   [10] [11] [12]

Problem 5: Linking (26points)

1. [1] [2]

   [3] [4]

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

   [5] [6] [7] [8]

   [9] [10] [11] [12]

3. [1] [2] [3]

4. [1] [2]

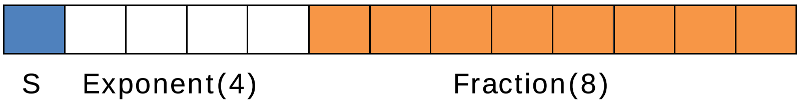
Problem 6: Optimization (16points)

1.

2.

Problem 1: Floating Point (14 points)

Number Conversion: IEEE 754 single precision float standard with a little change is illustrated below.



1. Filling the blanks with proper values. (4')
   1) **Normalized**: $(-1)^{\mathbf{sign}}$ * (1.fraction) * $2^{\mathbf{exponent\text{-}bias}}$, where **bias**= ___[1]___;
   2) **Infinity** (s = 0 and In **binary** form): ___[2]___;
   3) **Largest Normalized Value** (s = 0 and in **binary** form):___[3]___;
   4) **Smallest Denormalized Value** (s = 0 and in **binary** form):___[4]___;

2. Convert the number **(-9.9375)**$_{\mathbf{10}}$ into IEEE 754 FP single precision representation (in binary). (3')

3. What is the equivalent value to **(1 0000 10010100)**$_{\mathbf{2}}$ as a decimal number? (3')

4. Calculate both the sum of **(0 0100 00111100)**$_{\mathbf{2}}$ and **(0 1001 10101001)**$_{\mathbf{2}}$, and then round the results with Round-to-Even rounding modes. (NOTE: Please give your steps detailed and the result should be in **binary** form.)? (4')

Problem 2: x86-64 (14 points)

Suppose the following C and assembly code is defined on a **64-bit little endian** machine (x86-64/Linux). Please fill in the blanks according to the assembly code and c code. (2'*7 = 14')

```
int foo (int arg1, int arg2,
              int arg3, int arg4)
{
   int a[] = {1,2,3,4};
   int sum = a[arg1] + a[arg4];
   return sum;
}
```

```
int main(int argc, char *argv[])
{
    int i = 1;
    int j = foo(0, 1, 2, 3);
    printf("i=%d, j=%d\n", i, j);
    return 0;
}
```

```
<foo>:
  pushq %rbp
  movq %rsp, %rbp
  movl %edi, -36(%rbp)
  movl %esi, -40(%rbp)
  movl %edx, -44(%rbp)
  movl ___[5]___, -48(%rbp)
  movl $1, -32(%rbp)
  movl $2, -28(%rbp)
  movl $3, -24(%rbp)
  movl $4, -20(%rbp)
  movl ___[6]___, %eax
  cltq
  movl -32(%rbp,%rax,4), %edx
  movl -48(%rbp), %eax
  cltq
  movl ___[7]___, %eax
  addl %edx, %eax
  movl %eax, -4(%rbp)
  movl -4(%rbp), %eax
  popq %rbp
  ret
```

```
<main>:
  pushq %rbp
  movq %rsp, %rbp
  subq $32, %rsp
  movl %edi, -20(%rbp)
  movq %rsi, -32(%rbp)
  movl $1, ___[1]___
  movl $3, ___[2]___
  movl $2, %edx
  movl $1, %esi
  movl $0, %edi
  call foo
  movl ___[3]___, ___[4]___
  movl $.LC0, %eax
  movl -4(%rbp), %edx
  movl -8(%rbp), %ecx
  movl %ecx, %esi
  movq %rax, %rdi
  movl $0, %eax
  call printf
  movl $0, %eax
  leave
  ret
```

Problem 3: Memory Allocation (14 points)

The figure simulates the **initial** status of memory at a certain time. Allocated blocks are **shaded**, and free blocks are **blank** (each block represents **4 bytes**). The allocator maintains **double-word** alignment. Given the execution sequence of memory allocation operations (**malloc()** or **free()**) from 1 to 4. Please answer the following questions.

						↓P1								↓P2							
24/0					24/0	16/1	▨	▨	16/1	16/0			16/0	8/1	8/1	24/0					24/0

```
1:   P3 = malloc(9);
2:   P4 = malloc(3);
3:   free(P2);
4:   P5 = malloc(15);
```

1. Assume **first-fit** algorithm is used to find free blocks and **coalesce immediately**. Please draw the status of memory and mark with variables after the **2**nd and **4**th operation is executed. (8’)
2. Compute the total number of bytes of the **internal** fragments. (3’)
3. Compute the total number of bytes of the **external** fragments. (3’)

Problem 4: Cache (16 points)

Consider a **12–bit** machine with a **2-way** set associative cache, memory access are to **1-byte** words, the contents of the cache are as follows, with Hex notation.

Set	Tag	Valid	Byte0	Byte1	Byte2	Byte3	Tag	Valid	Byte0	Byte1	Byte2	Byte3
0	0x09	1	0x86	0x30	0x3F	0x10	0x00	0	--	--	--	--
1	0x45	1	0xAB	0xCD	0xEF	0x00	0x38	0	0x00	0xBC	0x0B	0x37
2	0xEB	0	--	--	--	--	0x0B	0	--	--	--	--
3	0x06	0	--	--	--	--	0x32	1	0x12	0x08	0x7B	0xAD

1. please fill the following blanks (4’)

   Cache size:___[1]___bytes

Field	Length(bit)
Tag	[2]
Set	[3]
Offset	[4]

2. With **above** cache contents, cache replacement policy is **LRU**, we have several **sequentially** executed memory accesses, please fill in the following blanks. (12’) (NOTE: if unknown fill in ‘**--**’)

Order	Address	Set	Hit or not (Yes/No)	Byte Returned
1	0x455	[1]	[2]	[3]
2	0xEF4	[4]	[5]	[6]
3	0xEF5	[7]	[8]	[9]
4	0xAB7	[10]	[11]	[12]

Problem 5: Linking (26 points)

The following program consists of two modules: main and utility. Their corresponding source codes and relocatable object files are shown below.

main.c

```
#define TOTAL 30
struct grade {
   unsigned int id;
   short score;
};
struct grade list[TOTAL];
void get_id(unsigned int *id);
void get_score(short *score);
```

```
int main(void) {
   int i;
   for (i = 0; i != TOTAL; i++) {
      get_id(&list[i].id);
      get_score(&list[i].score);
   }
   return 0;
}
```

main.o

```
00000000 <main>:
  0:   55                        push   %ebp
  1:   89 e5                     mov    %esp,%ebp
  3:   83 e4 f0                  and    $0xfffffff0,%esp
  6:   83 ec 20                  sub    $0x20,%esp
  9:   c7 44 24 1c 00 00 00 00   movl   $0x0,0x1c(%esp)
  11:  eb 30                     jmp    43 <main+0x43>
  13:  8b 44 24 1c               mov    0x1c(%esp),%eax
  17:  c1 e0 _[1]_               shl    ___[2]___,%eax
  1a:  05 00 00 00 00            add    $0x0,%eax
  1f:  89 04 24                  mov    %eax,(%esp)
  22:  e8 fc ff ff ff            call   23 <main+0x23>
  27:  8b 44 24 1c               mov    0x1c(%esp),%eax
  2b:  c1 e0 _[1]_               shl    ___[2]___,%eax
  2e:  05 00 00 00 00            add    $0x0,%eax
  33:  83 c0 _[3]_               add    ___[4]___,%eax
  36:  89 04 24                  mov    %eax,(%esp)
  39:  e8 fc ff ff ff            call   3a <main+0x3a>
  3e:  83 44 24 1c 01            addl   $0x1,0x1c(%esp)
  43:  83 7c 24 1c 1e            cmpl   $0x1e,0x1c(%esp)
  48:  75 c9                     jne    13 <main+0x13>
  4a:  b8 00 00 00 00            mov    $0x0,%eax
  4f:  c9                        leave
  50:  c3                        ret
```

utility.c

```
void get_id(unsigned int *id) { scanf("%u", id); }
void get_score(short *score) { scanf("%hd", score); }
```

utility.o

```
00000000 <get_id>:
  0:   55                        push   %ebp
  1:   89 e5                     mov    %esp,%ebp
  3:   83 ec 18                  sub    $0x18,%esp
  6:   8b 45 08                  mov    0x8(%ebp),%eax
  9:   89 44 24 04               mov    %eax,0x4(%esp)
  d:   c7 04 24 00 00 00 00      movl   $0x0,(%esp)
  14:  e8 fc ff ff ff            call   15 <get_id+0x15>
  19:  c9                        leave
  1a:  c3                        ret

0000001b <get_score>:
  1b:  55                        push   %ebp
  1c:  89 e5                     mov    %esp,%ebp
  1e:  83 ec 18                  sub    $0x18,%esp
  21:  8b 45 08                  mov    0x8(%ebp),%eax
  24:   89 44 24 04               mov    %eax,0x4(%esp)
  28:   c7 04 24 03 00 00 00      movl   $0x3,(%esp)
  2f:   e8 fc ff ff ff            call   30 <get_score+0x15>
  34:   c9                        leave
  35:   c3                        ret
```

Partial **.symbol table** after relocation

Name	Type	Value
main	FUNC	08048464
get_id	FUNC	0804842c
get_score	FUNC	08048447
list	OBJECT	0804a040
_GLOBAL_OFFSET_TABLE_	OBJECT	0804a000

Partial **.PLT (Procedure Linkage Table)** after linking:

```
08048330 <__isoc99_scanf@plt>:
8048330:  ff 25 14 a0 04 08     jmp   *0x804a014
8048336:  68 10 00 00 00        push  $0x10
804833b:  e9 c0 ff ff ff        jmp   8048300 <_init+0x2c>
```

1. Fill in the blanks in **main.o** (1' * 4 = 4')
2. Fill in the relocation entries of **main.o** and **utility.o** respectively. Relocation entries of **main.o**:(12')

Section	Offset	Name	Type
.text	0x1b	list	[1]
.text	[2]	list	[3]
.text	0x23	get_id	[4]
.text	[5]	get_score	R_386_PC32

Relocation entries of **utility.o**:

Section	Offset	Name	Type
.text	[6]	scanf	[7]
.text	0x30	scanf	[8]
.text	0x10	[9]	[10]
.text	[11]	[12]	R_386_32

3. Write down the underlined three instructions **after linking** according to all information provided: (2' * 3 = 6')

```
1a:  05 00 00 00 00        (main.o: add  $0x0,%eax)
After relocation: ___[1]___

39:  e8 fc ff ff ff        (main.o: call  3a <main+0x3a>)
After relocation: ___[2]___

14:  e8 fc ff ff ff        (utility.o: call  15 <get_id+0x15>)
After relocation: ___[3]___
```

4. Please answer the following questions (2’*2 = 4’)

   1) What is the value of 32-bit word at **0x804a014** just before **get_id()** is **first** called? (2’)

   2) What is the index of **scanf()** in **_GLOBAL_OFFSET_TABLE_**?
   (NOTE: Index starts from 0). (2’)

Problem 6: Optimization (16 points)

Suppose we have the following codes that run with little efficiency.

```
typedef struct { // This is a n*2 matrix.
    int n;
    int *base;  // All elements are within [0,100).
} mat2_t;
int row_count(mat2_t *p) { return p->n; }
int elem_at(mat2_t *p, int i, int j) { return p->base[i * 2 + j]; }
void find_max_min(mat2_t *p, int *max, int *min)
{
    *max = -1;
    for (int i = 0; i < row_count(p); i++)
        for (int j = 0; j < 2; j++)
            if (elem_at(p, i, j) > *max)
                *max = elem_at(p, i, j);
    *min = *max;
    for (int i = 0; i < row_count(p); i++)
        for (int j = 0; j < 2; j++)
            if (elem_at(p, i, j) < *min)
                *min = elem_at(p, i, j);
}
```

1. Optimize the code using the machine-independent optimization techniques learned from the ICS course. (Hint: You need to use at least **SIX** techniques) (12’)
2. Further optimize the code by eliminating the nested-for-loop.(4’)