上 海 交 通 大 学 试 卷（ A 卷）

（ 2011 至 2012 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: Y86 (13points)

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

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

Problem 2: Memory (12points)

1 1)

2)

3)

Problem 3: x86-64 (20points)

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

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 (27points)

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

2. [1] [2] [3] [4]
   [5] [6] [7] [8]
   [9] [10] [11] [12]
   [13]

3. 1)

   2)

   3)

Problem 5: Linking (28points)

1. [1] [2] [3]
   [4] [5]

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

3. [1]
   [2]
   [3]

4. 1)

   2)

Problem 1: Y86 (13 points)

	`# Execution begins at address 0`
`0x000:`	`.pos 0`
`0x000: 30f400010000`	`init: irmovl Stack, %esp`
`___[1]___: 30f500010000`	`irmovl Stack, %ebp`
`0x00c: 7020000000`	`jmp Main`
	`# Array of 4 elements`
`0x014:`	`.align 4`
`0x014:`	`array:`
`0x014: 000b0000`	`.long 0x00000b00`
`0x018: ___[2]___`	`.long 0x0fff6320`
`0x01c: ___[3]___`	`.long 0x12345678`
`0x020: 30f214000000`	`Main: ___[4]___`
`0x026: a02f`	`pushl %edx`
`0x028: 802e000000`	`call Foo`
`0x02d: 00`	`halt`
`0x02e: a05f`	`Foo: pushl %ebp`
`0x030: 2045`	`rrmovl %esp,%ebp`
`0x032: 501508000000`	`mrmovl 8(%ebp),%ecx`
`0x038: 30f203000000`	`irmovl $3,%edx`
`0x03e: 6222`	`andl %edx,%edx`
`0x040: ___[5]___`	`je End`
`0x045: 506100000000`	`Loop: mrmovl (%ecx),%esi`
`0x04b: 2060`	`rrmovl %esi,%eax`
`0x04d: 30f304000000`	`irmovl $4,%ebx`
`0x053: 6031`	`addl %ebx,%ecx`
`0x055: 30f3ffffffff`	`irmovl $-1,%ebx`
`0x05b: 6032`	`addl %ebx,%edx`
`0x05d: 7445000000`	`jne Loop`
`0x062: b05f`	`End: ___[6]___`
`0x064: 90`	`ret`
`0x100:`	`.pos 0x100`
`0x100:`	`Stack:`

1) Fill in the blanks according to the Y86 assembly and object code. (1'*6 = 6')

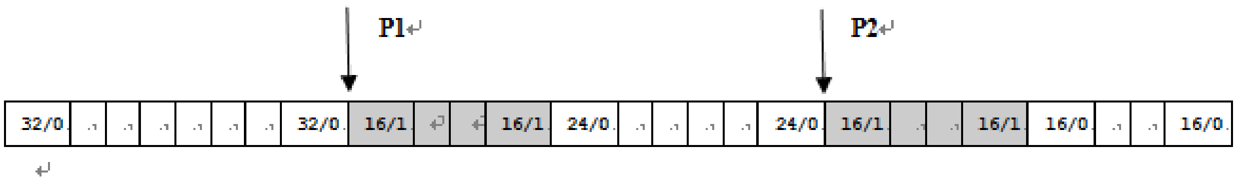
2) Fill in the value of the registers after the above program has **halted**. (1' * 7 = 7')

`%eax:` ___[1]___ `%ebx:` ___[2]___ `%ecx:` ___[3]___ `%edx:` ___[4]___

`%esi:` ___[5]___ `%edi:` -- `%ebp:` ___[6]___ `%esp:` ___[7]___

Problem 2: Memory (12 points)

1. The figure simulates the initial status of memory at a certain time. Allocated blocks are shaded, and free blocks are blank (each block represents **1 word = 4bytes**). Headers and footers are labeled with the number of bytes and allocated bit. The allocator maintains **double-word** alignment. Given the execution sequence of memory allocation operations (**`malloc`** or **`free`**) from 1 to 7. Please answer the following questions.




```
1:   P3 = malloc(16);
2:   P4 = malloc(8);
3:   P5 = malloc(3);
4:   free(P3);
5:   free(P4);
6:   free(P2);
7:   P6 = malloc(24);
```

1) Assume **first-fit** algorithm is used to find free blocks. Please draw the status of memory and mark with variables after the **3rd** operation is executed. (4’)

2) Assume **best-fit** algorithm is used to find free blocks. Please draw the status of memory and mark with variables after the **3rd** operation is executed. (4’)

3) To analyze the memory utilization and time utilization of **first-fit** and **best-fit** in above execution sequence. **Which** is better and **why**? (4’)

Problem 3: x86-64 (20 points)

Suppose the following C and assembly code is defined on a **64-bit little endian** machine (x86-64/Linux).

```
void foo (long int *pt1, long int *pt2, char *pt3, char *pt4,
          int in5, int in6, long int *pt7, long int *pt8)
{
    long int temp = *pt7 + *pt8;
}
int main(void)
{
    long int ia=1, ib=0xabcd;
    char ic=2, id=0x12;
    int ie=6, ig=7;
    long int ih=8, il=9;
    foo(&ia,&ib,&ic,&id,ie,ig,&ih,&il);
    return 0;
}
```

```
0000000000000034 <main>:
 34:  55                        push   %rbp
 35:  48 89 e5                  mov    %rsp,%rbp
 38:  53                        push   %rbx
 39:  48 83 ec 40               sub    $0x40,%rsp
 3d:  48 c7 45 e8 01 00 00 00   movq   $0x1,-0x18(%rbp)
 45:  48 c7 45 e0 cd ab 00 00   movq   $0xabcd,-0x20(%rbp)
 4d:  c6 45 df 02               movb   $0x2,-0x21(%rbp)
 51:  c6 45 de 12               movb   $0x12,-0x22(%rbp)
 55:  c7 45 f0 06 00 00 00      movl   $0x6,-0x10(%rbp)
 5c:  c7 45 f4 07 00 00 00      movl   $0x7,-0xc(%rbp)
 63:  48 c7 45 d0 08 00 00 00   movq   $0x8,-0x30(%rbp)
 6b:  48 c7 45 c8 09 00 00 00   movq   $0x9,-0x38(%rbp)
 73:  44 8b 45 f4               mov    -0xc(%rbp),%r8d
 77:  8b 7d f0                  mov    -0x10(%rbp),%edi
 7a:  48 8d 4d de               lea    -0x22(%rbp),__[1]__
 7e:  48 8d 55 df               lea    -0x21(%rbp),__[2]__
 82:  48 8d 5d e0               lea    -0x20(%rbp),%rbx
 86:  48 8d 45 e8               lea    -0x18(%rbp),%rax
 8a:  48 8d 75 c8               lea    __[3]__,%rsi
 8e:  48 89 74 24 08            mov    %rsi,0x8(%rsp)
 93:  48 8d 75 d0               lea    -0x30(%rbp),%rsi
 97:  48 89 34 24               mov    %rsi,__[4]__
 9b:  45 89 c1                  mov    %r8d,%r9d
 9e:  41 89 f8                  mov    %edi,__[5]__
```

```
  a1:  48 89 de                    mov    %rbx,%rsi
  a4:  48 89 c7                    mov    %rax,__[6]__
  a7:  e8 00 00 00 00              callq  foo <main+0x78>
   ...                             ...

0000000000000000 <foo>:
   0:  55                       push   %rbp
   1:  48 89 e5                 mov    %rsp,%rbp
   4:  48 89 7d e8              mov    %rdi,-0x18(%rbp)
   8:  48 89 75 e0              mov    __[7]__,-0x20(%rbp)
   c:  48 89 55 d8              mov    %rdx,-0x28(%rbp)
  10:  48 89 4d d0              mov    __[8]__,-0x30(%rbp)
  14:  44 89 45 cc              mov    %r8d,-0x34(%rbp)
  18:  44 89 4d c8              mov    __[9]__,-0x38(%rbp)
  1c:  48 8b 45 10              mov    0x10(%rbp),%rax
  20:  48 8b 10                 mov    (%rax),%rdx
  23:  48 8b 45 18              mov    __[10]__,%rax
  27:  48 8b 00                 mov    (%rax),%rax
  2a:  48 8d 04 02              lea    (%rdx,%rax,1),%rax
  2e:  48 89 45 f8              mov    %rax,-0x8(%rbp)
  ...                           ...
```

1) Fill in the blanks according to the assembly code and c code. (2’*10 = 20’)_

Problem 4: Cache (27 points)

Consider a **32-bit** machine with a **4-way** set associative cache. There are **16 sets**. Each block is **32 bytes**. Answer the following questions:

1. please fill the following blanks (4’)

   Cache size:___[1]___bytes

Field	Length(bit)
Total	32
Tag	[2]
Set	[3]
Offset	[4]

2. Assume the cache is initially **empty**. There are several memory accesses in order. Fill the second table and compute the miss rate. (1′ * 12 + 2′ = 14′)

Order	Address	Set	Miss or not (Yes/No)
1	**0xe166**	[1]	[2]
2	**0xe17e**	[3]	[4]
3	**0xe15a**	[5]	[6]
4	**0x9d60**	[7]	[8]
5	**0x9d5f**	[9]	[10]
6	**0xe142**	[11]	[12]

Miss rate: [13]

3. Assume the following code is executed on this machine. You should also assume the following:
   - ✧ sizeof(int) = 4
   - ✧ the starting address of array a is 0x100
   - ✧ variable i, j, t are in registers

```
#define M 4
#define N 8

int a[M][N];
int i, j;

for (i = 0; i < M - 1; i++) {
    int t = a[i + 1][0];
    for (j = 0; j < N; j++)
        a[i][j] = a[i][j] * t;
}
```

1) Total number of memory accesses: [1] (2′)

2) Miss rate: [2] (2′)

3) Suppose you have the chance to **double** the cache size. **Which** following scheme is best choice for above code and **why**? (5′)

   Scheme A: double the number of sets. (i.e., 32 sets)
   Scheme B: double the block size (i.e., 64 bytes)
   Scheme C: double the number of ways (i.e., 8-way)

Problem 5: Linking (28 points)

The following program consists of three source files: node.h, display.c and main.c. The corresponding relocatable object files are also listed.

display.c

```
#include <stdio.h>
#include "node.h"

extern node_t head;
extern node_t second;

void display(void)
{
  printf("%d %d",
     head.value,
     second.value);
}
```

display.o

```
.text:
00000000 <display>:
   0: 55                     push   %ebp
   1: 89 e5                  mov    %esp,%ebp
   3: 83 ec 18               sub    $0x18,%esp
   6: a1 00 00 00 00         mov    0x0,%eax
   b: 8b 15 00 00 00 00      mov    0x0,%edx
  11: 89 44 24 08            mov    %eax,0x8(%esp)
  15: 89 54 24 04            mov    %edx,0x4(%esp)
  19: c7 04 24 00 00 00 00   movl   $0x0,(%esp)
  20: e8 fc ff ff ff         call   21<display+0x21>
  25: c9                     leave
  26: c3                     ret
```

node.h and main.c

```
node.h:

typedef struct node {
  int value;
  struct node* next;
} node_t;

--------------------

#include "node.h"

extern void display();
node_t head, second;

int main(void)
{
  head.next=&second;
  head.value=1;
  head.next->value=2;
  second.next=0;

  display();

  return 0;
}
```

main.o

```
00000000 <main>:
   0: 8d 4c 24 04            lea    0x4(%esp),%ecx
   4: 83 e4 f0               and    $0xfffffff0,%esp
   7: ff 71 fc               pushl  0xfffffffc(%ecx)
   a: 55                     push   %ebp
   b: 89 e5                  mov    %esp,%ebp
   d: 51                     push   %ecx
   e: 83 ec 04               sub    $0x4,%esp
  11: c7 05 04 00 00 00 00   movl   $0x0,0x4
  18: 00 00 00
  1b: c7 05 00 00 00 00 01   movl   $0x1,0x0
  22: 00 00 00
  25: a1 04 00 00 00         mov    0x4,%eax
  2a: c7 00 02 00 00 00      movl   $0x2,(%eax)
  30: c7 05 04 00 00 00 00   movl   $0x0,0x4
  37: 00 00 00
  3a: e8 fc ff ff ff         call   3b <main+0x3b>
  3f: b8 00 00 00 00         mov    $0x0,%eax
  44: 83 c4 04               add    $0x4,%esp
  47: 59                     pop    %ecx
  48: 5d                     pop    %ebp
  49: 8d 61 fc               lea 0xfffffffc(%ecx),%esp
  4c: c3                     ret
```

Partial **.symbol table** after relocation

Num	Value	Size	Type	Bind	Vis	Ndx	Name
52:	08049670	8	OBJECT	GLOBAL	DEFAULT	24	second
61:	080483a4	39	FUNC	GLOBAL	DEFAULT	12	display
68:	08049678	8	OBJECT	GLOBAL	DEFAULT	24	head
71:	080483cc	77	FUNC	GLOBAL	DEFAULT	12	main

1. The global variable "**head**" is referenced **three times** and "**second**" is referenced **twice** in **main.c** file. Please fill in the following relocation entries for symbol "**head**" and "**second**" in **main.o** (1′ * 5 = 5′)

OFFSET	TYPE	SYMBOL
[1]	R_386_32	head
[2]	R_386_32	head
[3]	R_386_32	head
[4]	R_386_32	second
[5]	R_386_32	second

2. Please fill in the relocation entries in **display.o** (2′*6 = 12′)

OFFSET	TYPE	SYMBOL
[1]	[2]	head
[3]	[4]	second
0x0000001c	R_386_32	.rodata
[5]	[6]	printf

3. After relocation and the program is built, what changes will happen to the underlined 3 instructions according to **.symbol table** given above? (2′*3 = 6′)

b: 8b 15 00 00 00 00 (display.o mov 0x0,%edx)

After relocation: ___[1]___

3a: e8 fc ff ff ff (main.o: call 3b <main+0x3b>)

After relocation: ___[2]___

11: c7 05 04 00 00 00 00 00 00 00 (main.o: movl $0x0,0x4)

After relocation: ___[3]___

4. After relocation and the program is built, part of its procedure linkage table (PLT) is shown as follow:

```
080482b8 <printf@plt>:
 80482b8:  ff 25 60 96 04 08      jmp    *0x8049660
 80482be:  68 10 00 00 00         push   $0x10
 80482c3:  e9 c0 ff ff ff         jmp    8048288 <_init+0x18>
```

   1) What is the **value** of word starting from address `0x08049660`? (2′)

   2) Briefly describe the procedure of lazy binding when “**`printf`**” is called. (3′)