上 海 交 通 大 学 试 卷（ A 卷）

（ 2015 至 2016 学年 第 1 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: Linking (14points)

1. [1] [2] [3] [4]
   [5] [6] [8] [8]
   [9] [10] [11] [12]
   [13] [14] [15] [16]
   [17] [18] [19] [20]

2.

Problem 2: PIC (8points)

1. [1] [2]

2.

3.

Problem 3: Relocation (20points)

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

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)									

3. [13]

[14]

[15]

[16]

[17]

[18]

[19]

[20]

Problem 4: HCL (8points)

1.

2.

3.

Problem 5: Y86 (15points)

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

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

Problem 6: Processor (35points)

1.

Field	retxx
Fetch	
Decode	
Execute	
Memory	
Write Back	
PC update	

2.

3.

4. [1] [2] [3]

[4] [5] [6]

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

Problem 1: Linking (14 points)

The following program consists of two modules: **main** and **tmp**. Their corresponding source code files are shown below.

/* main.c */

```
#include <stdio.h>

void tmp(int *x, int *y, int n);
short b=2;
int d;
int x[3] = {1,2,3};
int y[3] = {0,0,0};
int main(void) {
  int a=7;
  tmp(x, y, b);
  printf("%x %x %x %x", a, b, y[2], d);
  return 0;
}
```

/* tmp.c */

```
static short b = 3;
int d =8;

void
tmp(int *x, int *y, int n) {
  int i;
  for (i=0; i<n; i++)
    y[i] = x[i] + 4;
}
```

1. For symbols that are defined and referenced in **main.o**, please indicate whether they will have a symbol table entry in the **.symtab** section in module **main.o**. If **Yes**, please complete the symbol table; If **No**, fill with '--'. (0.5'x20=10')

Symbol	.symtab entry	TYPE	Bind	Value	Size	Ndx
d	[1]	[2]	GLOBAL	[3]	[4]	[5]
x	Y	OBJECT	[6]	00000004	[7]	[8]
y	[9]	OBJECT	GLOBAL	00000000	[10]	[11]
main	[12]	[13]	GLOBAL	[14]	92	[15]
tmp	[16]	[17]	GLOBAL	[18]	[19]	[20]

2. Please write down the output of **main.c**. (4')

Problem 2: PIC (8 points)

TA wrote a program using **shared library**. In the program there are some calls to function **`printf`** and **`scanf`** which are from **`libc`**.

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

```
08048330 <printf@plt>:
 8048330:   ff 25 0c a0 04 08       jmp    *0x804a00c
 8048336:   68 00 00 00 00          push   $0x0
 804833b:   e9 e0 ff ff ff          jmp    8048320
 ...
08048360 <__isoc99_scanf@plt>:
 8048360:   ff 25 18 a0 04 08       jmp    __[1]__
 8048366:   68 18 00 00 00          push   $0x18
 804836b:   e9 b0 ff ff ff          jmp    8048320
```

Partial **.GOT** (_GLOBAL_OFFSET_TABLE_) after linking is below:

Address	Entry	Content
0x804a000	GOT[0]	0x08049f14
0x804a004	GOT[1]	0xf7ffd938
0x804a008	GOT[2]	0xf7ff04f0
0x804a00c	GOT[3]	[2]
0x804a010	GOT[4]	0x08048346
0x804a014	GOT[5]	0xf7e1f990
0x804a018	GOT[6]	0x08048366

1. Please fill the blanks in **`.PLT`** and **`.GOT`** (2'*2=4')
2. What is the address of **`.dynamic`** section? (2')
3. What will be stored in the address **`0x804a00c`** **after** first calling **`printf`**? (2')

.

Problem 3: Relocation (20 points)

The following program consists of two source files: **foo.c** and **bar.c**. The relocatable object files are also listed.

```
#include<stdio.h>

extern int a[2];
extern void fun2();

int b[3] = {0,1,2};
int* d = &a[1];

int main(void)
{
  a[1]=4;
  fun2();
}
```

```
/* foo.o */
.text
00000000 <main>:
 0: 55                      push %ebp
 1: 89 e5                   mov  %esp,%ebp
 3: 83 e4 f0                and  $0xfffffff0,%esp
 6: c7 05 04 00 00 00 04    movl $0x4,   [1]
 d: 00 00 00
10: e8   [2]                call 11 <main+0x11>
15: c9                      leave
16: c3                      ret

.data:
00000000:
 0: 00 00 00 00
 4: 01 00 00 00
 8: 02 00 00 00

0000000c:
 c: 04 00 00 00
```

```
/* bar.c */
#include<stdio.h>

extern int b[3];
extern int *d;

int a[2];
int* c = &b[2];

void fun1(void) {
  *d=0;
}

void fun2(void) {
  fun1();
}
```

```
/* bar.o */
.text
00000000 <fun1>:
 0: 55                      push %ebp
 1: 89 e5                   mov  %esp,%ebp
 3: a1 00 00 00 00          mov  0x0,%eax
 8: c7 00 00 00 00 00       movl $0x0,   [3]
 e: 5d                      pop  %ebp
 f: c3                      ret

00000010 <fun2>:
10: 55                      push %ebp
11: 89 e5                   mov  %esp,%ebp
13: e8 fc ff ff ff          call 14 <fun2+0x4>
18: 5d                      pop  %ebp
19: c3                      ret

.data:
00000000:
 0:    [4]
```

1. Fill in the blanks in above code of **foo.o** and **bar.o** respectively. (1'*4=4')

2. Fill in the relocation entries of **foo.o** and **bar.o** respectively. (1'*8=8')

Relocation entries of **foo.o**

Section	Offset	Type	Symbol Name
.text	[5]	R_386_32	a
.text	00000011	[6]	func2
.data	[7]	R_386_32	[8]

Relocation entries of **bar.o**

Section	Offset	Type	Symbol Name
.text	[9]	[10]	d
.text	00000014	R_386_PC32	[11]
.data	[12]	R_386_ 32	b

3. After relocation and the program is built, what changes will happen to the underlined instructions/data according to a part of the symbol table and partial comparison of relocation tables given below? (2'*4=8')

Name	Section	Type	Value
a	.bss	OBJECT	[13]
b	.data	OBJECT	[14]
c	.data	OBJECT	0804a02c
d	.data	OBJECT	0804a028
fun1	.text	FUNC	08048404
fun2	.text	FUNC	[15]
main	.text	FUNC	080483ed

A comparison of relocation table of **foo.o**

Section	Before relocation	After Relocation
.text	6:c7 05 04 00 00 00 04 movl $0x4,[1] 00 00 00	[16]
.text	10: e8 [2] call 11 <main+0x11>	[17]
.data	0000000c: 04 00 00 00	0804a028: 38 a0 04 08

A comparison of relocation table of **bar.o**

Section	Before relocation	After relocation
.text	3:a1 00 00 00 00 mov 0x0,%eax	[18]
.text	13:e8 fc ff ff ff call 14 <fun2+0x4>	[19]
.data	00000000: [4]	[20]

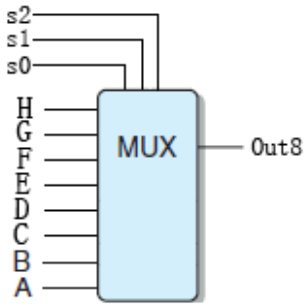
Problem 4: HCL (8 points)

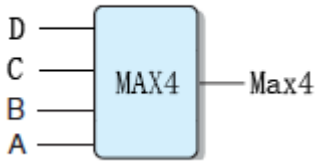
Please write down the HCL expressions for the following signals (HINT: you can refer to the **Section 4.2.2** in the CSAPP book). (2’+3’+3’ = 8’)

EXAMPLE: Show if the two input signals **a** and **b** are equal

```
bool eq = (a&&b) || (!a && !b);
```

1. The HCL expression for a signal **xor**, equal to the Exclusive-Or of inputs **A** and **B**
2. The HCL expression for an eight-way multiplexor **MUX**





3. The HCL expression for a signal **Max4**, which chooses the biggest one among the four inputs (**A**, **B**, **C**, and **D**)

Problem 5: Y86 (15 points)

```
0x000:                  |        .pos 0
  0x000: 30f400100000   |  init:  irmovl 0x1000, %esp
  0x006: 30f500100000   |         irmovl 0x1000, %ebp
  0x00c: ____[1]____    |         call Main
  0x011: 00             |         halt
                        |
  _[2]_:                |         .align 4
  _[3]_: 33010000       |  Array: .long 0x133
  0x018: fc0d0000       |         .long 0xdfc
  0x01c: 2f0f0000       |         .long 0xf2f
  0x020: 33020000       |         .long 0x_[4]_
                        |
  _[5]_: a05f           |  Main:  pushl %ebp
  0x026: 2045           |         rrmovl %esp, %ebp
  0x028: a03f           |         ____[6]____
  0x02a: ____[7]____    |         irmovl Array, %edx
  0x030: 500200000000   |         mrmovl (%edx), %eax
  0x036: 503204000000   |         mrmovl 4(%edx), %ebx
  0x03c: 501208000000   |         mrmovl 8(%edx), %ecx
  0x042: 50220c000000   |         mrmovl 0xc(%edx), %edx
  _[8]_: ____[9]____    |         addl %eax, %ebx
  0x04a: 6112           |         subl %ecx, %edx
  0x04c: 6131           |         ____[10]____
  0x04e: b03f           |         popl %ebx
  0x050: 7055000000     |         jmp End
  0x055: 2054           |  End:   rrmovl %ebp, %esp
  0x057: b05f           |         popl %ebp
  0x059: 90             |         ret
```

1. Please fill in the blanks within above Y86 binary and assembly code. (1'*10=10')
2. Please calculate the value of below registers after the program HALT. (1'*5=5')

   R[%edx] = [1] R[%esp] = [2]

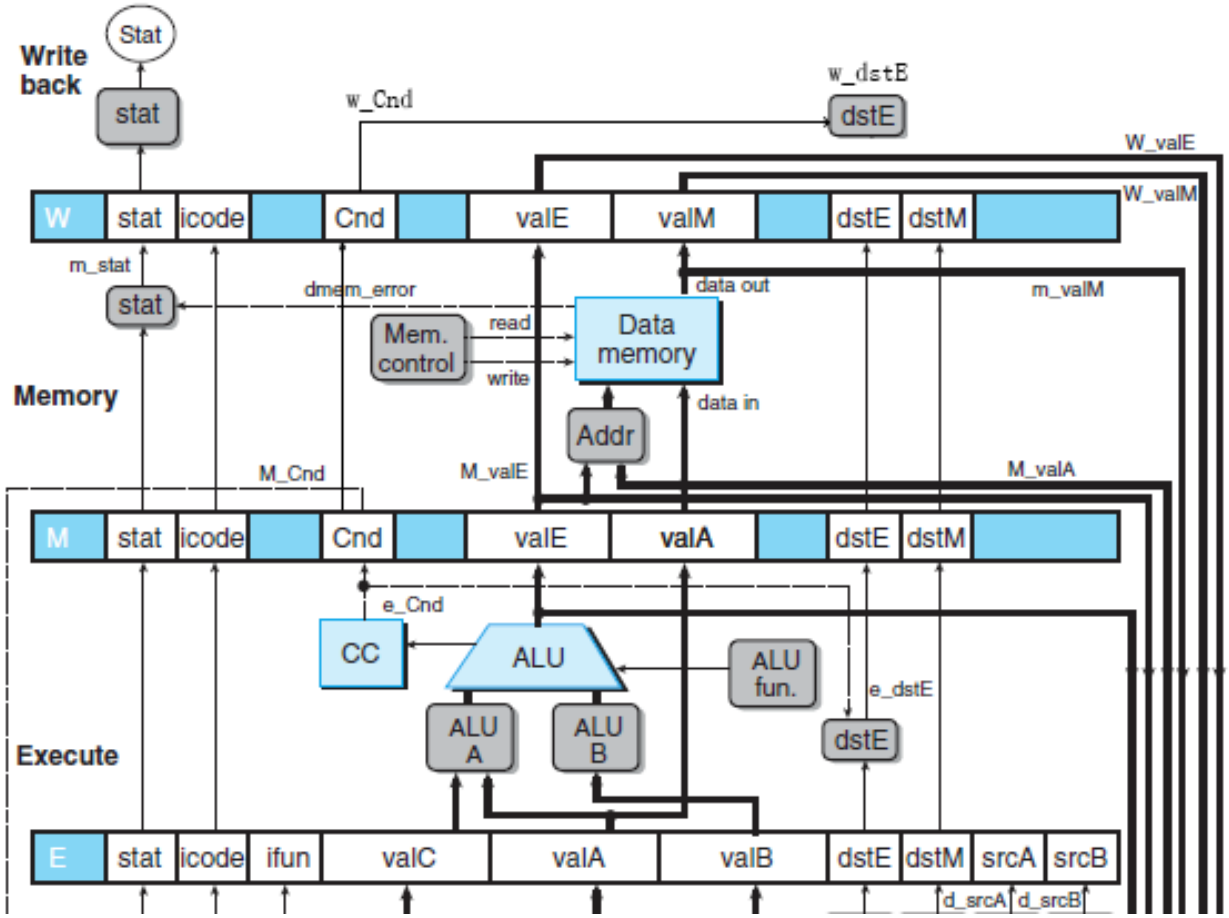
   CC: ZF = [3] SF = [4] OF = [5]

Problem 6: Processor (35 points)

Suppose we are using hardware structure of **PIPE-2015** which is modified from **PIPE** (see Figure 4.52 in CSAPP book). The **Cnd** signal is kept and can be used to write stage in **PIPE-2015**. Now, we want to add a new instruction: condition return, **retxx**, to the original Y86 instruction set, using the following encoding: (**NOTE** the original **ret** instruction will still be used)

	Byte	0	1	2	6
retxx		**E**	**Fn**		

The **Fn** field is the same as that of **jxx**. For example, **0xE1** stands for **retle**



1. Please fill in the **generic** function of each stage for **retxx** on PIPE-2015 like **Figure 4.21**. (Your design should be available in above circuit figure (2'*6=12') (NOTE: fill all functions in each stage, and use '–' for empty stage)

Field	retxx
Fetch	[1]
Decode	[2]
Execute	[3]
Memory	[4]
Write Back	[5]
PC update	[6]

2. Suppose **retxx** reuse the same forwarding circuit for **ret** and **jxx** and **NOT TAKEN** branch prediction strategies. There will be some hazards due to this new instruction. You need to list **new detection conditions** like Figure 4.64 and **new control action** like Figure 4.66. (2'+5')

Condition	Trigger

Condition	Pipeline register				
	F	D	E	M	W

3. Compared with the original instruction set and hardware structure **PIPE** (Figure 4.52), the new instruction (**`retxx`**) may cause some new combinations of hazards. Please draw the **pipeline states** figure (Figure 4.67) and list pipeline **control action** (see the table of Problem 4.35 and 4.36) for **ALL** new combinations. NOTE: we use **NOT_TAKEN** branch prediction strategies. (**HINT**: For **`retxx`**, **NOT TAKEN** strategy means assuming that ret would never happen.) (6')

4. Please estimated each of penalties and the frequency of instruction type, the frequency the condition arises are as following. Please fill the table and calculate the **CPI** (cycles per instruction). (6')

Cause	Name	Instruction frequency	Condition frequency	Bubbles	Product
Load/Use	LP	0.20	0.30	1	[2]
Mispredict Jump	MP	0.30	0.60	2	[3]
Return	RP	0.01	1.00	3	[4]
Mispredict Return	CRP	0.01	0.50	[1]	[5]

CPI = [6]

5. Please calculate the number of **cycles** and **waste cycles** for the following codes in **original** and **new** architecture. The initial value of all registers are **zero** and we always use **NOT_TAKEN** branch prediction strategy for all conditional jumps and returns. (**Hint**: you need calculate the number of cycles until the last stage of the last instruction) (4')

Original

```
     irmovl  Stack,%esp
     irmovl  $3,%eax
main:
     call    loop
     halt
loop:
     irmovl  $-1,%ebx
     addl    %ebx,%eax
     jne     loop
     ret
Stack:
```

Cyclyes: [1]

Wasted cycles: [2]

New

```
     irmovl  Stack,%esp
     irmovl  $3,%eax
main:
     call    loop
     halt
loop:
     irmovl  $-1,%ebx
     addl    %ebx,%eax
     rete
     jmp     loop
Stack:
```

Cyclyes: [3]

Wasted cycles: [4]