上 海 交 通 大 学 试 卷（ A 卷）

（ 2017 至 2018 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: FP (9points)

1. [1]

   [2]

   [3]

2.

3.

Problem 2: HCL (7points)

1.

2.

Problem 2: Y86 (17 points)

1. [1] [2]

   [3] [4]

   [5] [6]

   [7]

2.

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: Memory Allocation (16 points)

1.

2.

3.

4.

Problem 5: Optimization (16 points)

1.

2.

3.

Problem 6: Processor (35 points)

1.

2.

3. [1]

[2] [3] [4]

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

5.

6.

Problem 1: FP (9 points)

The following figure shows the floating-point format we designed for the exam, called **Float11**. Except for the length, it's the same as the IEEE 754 single-precision format you have learned in the class.

S	Exp(4bits)	Fract(6bits)

1. Fill the blanks with proper values. (3')
   1) **Denormalized**: $(-1)^{s} \times (0.\mathtt{Fract}) \times 2^{E}$, where **E** = ____[1]____
   2) **NaN** (any correct **binary** form): ____[2]____
   3) **Biggest Denormalized Value** (in **binary** form): ____[3]____

2. Convert the number $(-5.75)_{10}$ into the **Float11** representation (in **binary**). (3')

3. Assume we use IEEE **round-to-even** mode to do the approximation. Please calculate the subtraction: $(1\ 0000\ 101100)_2 - (0\ 0100\ 110010)_2$ and write the answer in **binary**. (The answer is represented in **Float11** as well) (3')

Problem 2: HCL (7 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).

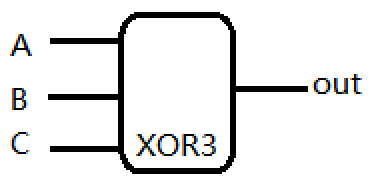
EXAMPLE: Show if the two input signals **a** and **b** are equal

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

1. The HCL expression for a signal **nand**, which is equal to **NAND** of inputs **a** and **b**, the truth table is given, and you should **only** use **NOT** (**!**) and **OR** (**||**) operators. (3')

NAND	0	1
0	1	1
1	1	0

2. The HCL expression for **a three-way xor** called XOR3. If and only if all the inputs are the same, output will be true (1). Each input and output is one-bit wise. (Hints: You can use boolean expressions or case expressions.) (4')



Problem 3: Y86 (17 points)

```
0x000:                            |  .pos 0
0x000:                            | init:
0x000: 30f40002000000000000       |  irmovq stack, %rsp
0x00a: ________[1]_________       |  irmovq stack, %rbp
0x014: 801e000000000000000        |  call main
0x01d: 00                         |  halt
0x01e:                            | main:
0x01e: 30f70003000000000000       |  irmovq list, %rdi
0x028: 30f60300000000000000       |  _____[2]_____
0x032: 803c0000000000000          |  call calculate
0x03b: 90                         |  ret
0x03c:                            | calculate:
0x03c: 6300                       |  _____[3]_____
0x03e: 30f30800000000000000       |  irmovq $8, %rbx
0x048: 2072                       |  rrmovq %rdi, %rdx
0x04a: 706d000000000000000        |  jmp test
0x053:                            | loop:
0x053: 50120000000000000000       |  _____[4]______
0x05d: 6010                       |  addq %rcx, %rax
0x05f: ________[5]_________       |  andq %rsi, %rax
0x061: 6032                       |  addq %rbx, %rdx
0x063: 50220000000000000000       |  mrmovq (%rdx), %rdx
0x06d:                            | test:
0x06d: 6222                       |  andq %rdx, %rdx
0x06f: 745300000000000000         |  jne loop
                                  |
0x200:                            |  ______[6]______
0x200:                            | stack:
                                  |
0x300:                            | .pos 0x300
0x300:                            | .align 8
0x300:                            | list:
0x300: 0800000000000000           |  .quad 0x8
0x308: 2003000000000000           |  .quad ____[7]____
0x310: 0900000000000000           |  .quad 0x9
0x318: 0000000000000000           |  .quad 0x0
0x320: 0a00000000000000           |  .quad 0xa
0x328: 1003000000000000           |  .quad 0x310
```

1. Please fill in the blanks within above Y86 binary and assembly code. (2'*7=7')
2. Please calculate the value of **%rax** after the program **HALT**. (3')

Problem 4: Memory Allocation (16 points)

Now we organize the heap as a sequence of **contiguous** allocated and free blocks, as shown below. **Allocated** blocks are shaded, and **free** blocks are blank (each block represents 1 word = 4 bytes). **Headers** and **footers** are labeled with the number of bytes and allocated bit. The allocator maintains **double-word** alignment. You are given the execution sequence of memory allocation operations (**`malloc()`** or **`free()`**) from 1 to 6.

									↓P1
32/0							32/0	16/1	

									↓P2
	16/1	24/0					24/0	16/1	

							↓P3		
	16/1	16/0			16/0	16/1			16/1

```
1.P4 = malloc(10)
2.free(P1)
3.P5 = malloc(10)
4.P6 = malloc(5)
5.free(P3)
6.P7 = malloc(10)
```

Please answer the questions below. Assume that **immediate coalescing** strategy and **splitting free blocks** are employed.

1. Assume **best-fit** algorithm is used to find free blocks. Please draw the **final** status of memory and mark with block size in headers and footers after the operation sequence is executed (4’).
2. Assume **first-fit** algorithm is used to find free blocks. Please draw the **final** status of memory and mark with block size in headers and footers after the operation sequence is executed (4’).
3. In our current algorithm, we do immediate coalescing by merging any adjacent free blocks each time a block is freed. However, such policy can hurt performance dramatically sometimes. Please give a simple example of such situation and explain why (4’). (**HINT**: consider the performance influence of frequent coalescing operations)
4. To handle such problems, we defer coalescing until some allocation requests fails and then scan the entire heap, coalescing all free blocks. Assume **first-fit** algorithm under such coalescing policy, draw the **final** status of memory and mark with block size in headers and footers after the operation sequence is executed (4’).

Problem 5: Optimization (16 points)

```
1.  typedef struct {
2.      float *data; /* points to an array */
3.      long capacity; /* the maximum length of the array */
4.      long length; /* number of elements in the array */
5.  } array_t;
6.  long get_length (array_t *arr) {return arr->length;}
7.  long get_capacity (array_t *arr) {return arr->capacity;}
8.  void copy_array(array_t *dst, array_t *src) {
9.      for (long i = 0; i < get_length(src); i++) {
10.         if (i >= get_capacity(dst))
11.             break;
12.         dst->data[i] = src->data[i];
13.     }
14.     dst->length = min(get_length(src), get_capacity(dst));
15. }
16. void sum_array(float *arr, long n, long *sum) {
17.     float ans = 0;
18.     for (long i = 0; (i+1) < n; i += 2)
19.         ans = ans + (arr[i] + arr[i + 1]);
20.     if (i < n)
21.         ans += arr[i];
22.     *sum = ans;
23. }
```

```
24. .Loop:
25.   movss (%rax, %rdx, 8), %xmm0
26.   addss 8(%rax, %rdx, 8), %xmm0
27.   addss %xmm0, %xmm1
28.   addq $2, %rdx
29.   cmpq $rdx, %rbp
30.   jg .Loop
```

1. Please rewrite the function `copy_array` with a combination of **at least 4 different optimizations** you learned in class. Comment briefly on the optimization. (2’*4=8’) NOTE: your optimizations cannot change the functionality of code above.
2. The translation of code in **line 18-19** is presented **in line 24-30**. Please abstract the operations as a data-flow graph and draw the graph. Please also mark the critical path(s) in the graph. (5’)
3. The code **in line 19** is modified as the following code in the table. After the modification, the CPE measurement increases from **X** to **2X.** Please point out why the CPE measurement increases. (3’)

```
ans = (ans + arr[i]) + arr[i + 1];
```

Problem 6: Processor (35 points)

In Lab 6, you were asked to implement a **ncopy** program, which will copy and count the number of positive integer. Here is the core loop in the initial version:

```
Loop:  mrmovq    (%rdi), %r10      // %r10 = *src
       rmmovq    %r10, (%rsi)      // *dst = %r10
       andq      %r10, %r10
       jle       Npos              // if (%r10 > 0)
       iaddq     $1, %rax          //     %rax++
Npos:                              // else, just skip
       iaddq     $8, %rdi          // src++
       iaddq     $8, %rsi          // dst++
       iaddq     $-1, %rdx         // cnt--
       jg        Loop              // if cnt > 0, continue
       halt                        // else, halt
```

1. Assume the input is "**1, 2, -3, 4, -5, -6**", the program is running on unmodified PIPE implementation. Please show **all hazards** involved and **how many bubbles** will be inserted for each. (6')

As the loop overhead and load-use hazard could be resolved by loop unrolling and code rewriting, the most overhead is from the **misprediction**. Here we will implement a new instruction: selective skip, **cskipxx**, with following encoding:

	Byte	0	
cskipxx		**E**	**Fn**

Name	Value (hex)	Meaning
ICSKIPXX	E	Code for cskipxx instruction

This instruction will "**skip" the next instruction if the condition is satisfied**, which causes the next instruction be treated as a **nop**. We use **always-not-taken** prediction, which means you will load the next instruction anyway, then cancel it if necessary. By correctly implementing this instruction, the misprediction penalty will be reduced to **1 cycle**.

2. Please rewrite the loop code by replacing **jle** with **cskipxx**, and answer the question 1 again. (Note: you do **NOT** need to write comment) (3'+3')

3. There will be a new control hazard called **cskipxx misprediction**. Please list the **detection conditions** like Figure 4.64 and **new control action** like Figure 4.66. (2'*4)

Condition	Trigger				
cskipxx misprediction	[1]				
	Pipeline register				
	F	D	E	M	W
	[2]	[3]	[4]	--	--

4. **cskipxx** misprediction and **ret** hazard could be combined in the above way. Please show how to handle it correctly based on your **cskipxx** implementation. (1'*4)

Mispredict

M	
E	**cskipxx**
D	

ret 1

M	
E	
D	**ret**

Pipeline register				
F	D	E	M	W
[1]	[2]	[3]	[4]	normal

5. Now we want to extend **cskipxx** to **cskip3xx**, which will skip **3 instructions** when condition is satisfied. Could **cskip3xx** misprediction be handled by simply inserting more bubbles when detected in **E stage**? Why? (3') Please describe your proposed solution in English. (2') And show the main HCL modifications. (4')

6. Please analyze the **pros** and **cons** of **cskip3xx** compared to **jxx**. (2')

草稿纸