上 海 交 通 大 学 试 卷（ B 卷）

（ 2016 至 2017 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: HCL (7points)

1.

2.

Problem 2: Y86 (10 points)

1. [1] [2]

[3] [4]

[5] [6]

[7] [8]

2.

Problem 3: Processor (18 points)

1. [1]

[2]

[3]

[4]

[5]

[6]

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)									

2. [1]

   [2] [3] [4]

3.

Problem 4: Cache (32 points)

1. [1] [2] [3]

2.

3. [1] [2] [3]

   [4] [5] [6]

   [7] [8] [9]

   [10] [11] [12]

   [13]

4 1)

  2)

  3)

Problem 5: Memory Allocation (16 points)

1.

2.

3.

Problem 6: Optimization (17 points)

1.

2.

3

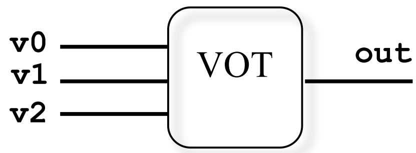
Problem 1: HCL (7 points)

Please write down the HCL expressions for the following signals (HINT: you can refer to the **Section 4.2.2** in the CSAPP book). (3'+4'=7')

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 **and**, which equals to the **AND** of inputs **a** and **b**, and you should **only** use **OR** (||) and **NOT** (!) operators.
2. The HCL expression for a three-way voter **VOT**. It means, if and only if there are equal or greater than 2 inputs are true (1), output will be true (1). Each input and output is one-bit wise. (Hints: You can use boolean expressions or case expressions.)



Problem 2: Y86 (10 points)

```
0x000:                      |         .pos 0
0x000: 30f460010000         | init:   _____[1]______
0x006: 30f560010000         |         irmovl Stack, %ebp
0x00c: 8012000000           |         call   Main
0x011: 00                   |         halt
                            |
0x012: a05f                 | Main:   pushl  %ebp
0x014: 2045                 |         rrmovl %esp,  %ebp
0x016: ____[2]_____         |         irmovl Array, %ecx
0x01c: 30f204000000         |         irmovl $4,    %edx
0x022: 30f000000000         |         irmovl $0,    %eax
0x028: 506100000000         | Loop:   mrmovl (%ecx),%esi
0x02e: 30f700000000         |         irmovl $0,    %edi
0x034: 6167                 |         subl   %esi,  %edi
0x036: ____[3]_____         |         cmovg  %edi,  %esi
0x038: 6060                 |         addl   %esi,  %eax
0x03a: 30f304000000         |         irmovl $4,    %ebx
0x040: 6031                 |         addl   %ebx,  %ecx
0x042: 30f3ffffffff         |         irmovl $-1,   %ebx
_[4]_: 6032                 |         addl   %ebx,  %edx
0x04a: 7428000000           |         _____[5]______
0x04f: 2054                 | End:    rrmovl %ebp, %esp
0x051: b05f                 |         popl %ebp
0x053: 90                   |         ret
```

```
                              |
  _[6]_:                      |        .align 4
  _[6]_: feffffff             | Array: .long   0xfffffffe
  0x058: 0a000000             |        .long   0x0000000a
  0x05c: ____[7]_____         |        .long   0x0000face
  0x060: fcffffff             |        .long   0xfffffffc
                              |
  0x160:                      |         _____[8]______
  0x160:                      | Stack:
```

1. Please fill in the blanks within above Y86 binary and assembly code. (1'*8=8')
2. Please calculate the value of **%eax** after the program **HALT**. (2')

Problem 3: Processor (18 points)

Suppose we are using hardware structure of **PIPE-2016** which is modified from **PIPE** (Figure 4.52 in CSAPP book). Now, we want to add a new instruction: simply selective jump, **ssjxx**, to the original Y86 instruction set, using the following encoding:

	Byte 0		1		2	3	4	5
ssjxx	E	Fn	rA	F	Dest			

The **Fn** field is the same as that of **jxx**. For example, **0xE5** stands for **ssjge**. If the condition described by **Fn** is satisfied, we will jump to the address stored in **rA**, we call this case as **TAKEN**. Otherwise, we will jump to **Dest**, we call this case as **NOT_TAKEN**.

1. Please fill in the **generic** function of each stage for **ssjxx** on PIPE-2016 like **Figure 4.21**. (1'*6=6')

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

2. Suppose **ssjxx** reuse the same forwarding circuit for **ret** and **jxx** and **NOT_TAKEN** prediction strategies. There will be some hazards due to this new instruction. Please list **new detection conditions** like Figure 4.64 and **new control action** like Figure 4.66. (3'+2'*3)

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

3. If we change prediction strategies of **ssjxx**, which uses **TAKEN** prediction strategies rather than **NOT_TAKEN**. Which design do you think is better? Why? (Hints: think about branch prediction in both designs.) (3’)

Problem 4: Cache (32 points)

Jack has a **16-bit** machine with a **2-way** set associative cache. There are **8 sets**. Each block is **4 bytes**. The following table shows the content of the data cache at time T. **`ByteX`** is the byte value stored at offset **`X`**.

Set	Tag	Valid	Byte0	Byte1	Byte2	Byte3	Tag	Valid	Byte0	Byte1	Byte2	Byte3
0	0x5df	1	0x66	0x22	0xfe	0x43	0x5d2	1	0xca	0xdb	0xed	0x00
1	0x7cf	1	0xab	0xcd	0xef	0xff	0x34e	1	0xdf	0x11	0x22	0x33
2	0x233	0	0x23	0x32	0x23	0x33	0x34e	1	0xfd	0x44	0x55	0x66
3	0x435	1	0xde	0xad	0xbe	0xef	0x34e	1	0xdf	0x11	0x22	0x33
4	--	0	--	--	--	--	--	0	--	--	--	--
5	0x701	1	0xff	0xff	0xcc	0xcc	0x435	1	0xad	0x18	0x24	0x19
6	0x881	1	0xde	0xed	0xbe	0xef	0x701	0	0x23	0x32	0xff	0xdd
7	--	0	--	--	--	--	--	0	--	--	--	--

1. How would a **16-bit** physical memory address be split into tag/set-index /block-offset fields in this machine? (2’*3=6’)

   tag __[1]__ bits, set-index __[2]__ bits, and block-offset __[3]__ bits

2. What is the size of this cache in bytes? (2’)

3. Assume the cache line replacement policy is **LRU**. A short program will read memory in the following sequences starting from time T. Each access will read **one byte**. Please fill the following blanks and compute the miss rate. If there is a cache miss, enter '--' for ‘Byte Returned’. (1’*12 + 2’*1 = 14’)

Order	Address	Set	Hit/Miss	Byte Returned
1	0xbbe0	0	Hit	0x66
2	0xeea3	[1]	[2]	[3]
3	0xe039	[4]	[5]	[6]
4	0x69c6	[7]	[8]	[9]
5	0xbb41	[10]	[11]	[12]

   Miss rate: ___[13]___

Jack buys a **NEW** machine with a **32-bit** physical memory address and tests the following program. The machine has a **4-way** set associative cache. There are **4 sets**. Each block is **16 bytes**. The cache line replacement policy is also **LRU**. The size of `int` value is **4 bytes**. NOTE that `i`, `j` and `result` are stored in **registers**. The cache is **empty** before each execution. Please only consider **data cache** access.

```
int x[4][16];
int y[4];
int z[4][16];

int fun(void) {
   int i, j, result = 0;
   for (j = 0; j < 16; j++)
      for (i = 0; i < 4; i++)
         result += x[i][j] * z[i][j] + y[i];
   return result;
}
```

Assume the address of `x[0][0]` is `0x000`. The address of `y[0]` is `0x100`. The address of `z[0][0]` is `0x110`. Answer the following questions:

1. What is total number of memory accesses? (2')
2. Please calculate the miss rate. (2')
3. Jack executes `fun(void)` on another two different data caches:
   1) **C1** is a 8-way set associative cache with 4 sets and 16 bytes cache line size;
   2) **C2** is a 4-way set associative cache with 8 sets and 16 bytes cache line size;
   Can **C1** reduce the miss rate? (1') If yes, please calculate the miss rate. If no, please explain the reason. (2')
   Can **C2** reduce the miss rate? (1') If yes, please calculate the miss rate. If no, please explain the reason. (2')

Problem 5: 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								
24/1					24/1	24/0			

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

	↓P3								
16/1			16/1	24/0					24/0

```
1.P4 = malloc(5)
2.P5 = malloc(11)
3.free(P3)
4.P6 = malloc(8)
5.free(P1)
6.P7 = malloc(2)
```

Please answer the questions below. Assume that **immediate coalescing** strategy and **splitting free blocks** are employed. (*NOTE*: **DON'T** need consider P1, P2 and P3 when calculating **internal** fragments)

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′). Please also calculate the total bytes of the **internal fragments** (2′).

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′). Please also calculate the total bytes of the **internal fragments** (2′).

3. According to the final status of memory, we find that best-fit algorithm makes better utilization of memory than first-fit algorithm. However, first-fit algorithm is better in efficiency. Please explain why first-fit enjoys **better performance** based on above execution sequences. (You must use above execution sequences as example.) (4′)

Problem 6: Optimization (17 points)

In US presidential election, each state votes separately. **state_result** records how many people vote for Trump and Clinton in a state. According to a state_result array **ra**, function **stat** computes the total votes among all states, and the winner and the gap between two candidates for each state.

```
typedef struct {
  int clinton;
  int trump;
  int winner;   // 0 for Clinton, 1 for Trump
  int gap;
} state_result;

typedef struct {
  int length;
  state_result *results;
} rst;

int get_length(rst *ra) { return ra->length; }
int get_t(rst *ra, int i) {return (ra->results)[i].trump;}
int get_c(rst *ra, int i) {return (ra->results)[i].clinton;}

void stat(rst *ra, int *sum) {
  for (int i = 0; i < get_length(ra); i++)
    *sum = *sum + get_t(ra, i) + get_c(ra, i);

  state_result *states = ra->results;
  for (int i = 0; i < get_length(ra); i++)
    if (states[i].clinton > states[i].trump) {
      states[i].winner = 0;
      states[i].gap = states[i].clinton - states[i].trump;
    } else {
      // assume Trump wins if he gets equal or more
      states[i].winner = 1;
      states[i].gap = states[i].trump - states[i].clinton;
    }
}

int total_clinton(state_result *r, int len) {
  if (len <= 0) return 0;
  return r->clinton + total_clinton(r + 1, len - 1);
}
```

Note: your optimizations cannot change the functionality of code above.

1. Please rewrite the loop **in line 18-19** with a combination of **at least 5 different optimizations** you learned in class. Comment briefly on the optimization. (2'*5=10')
2. Please rewrite the loop body **in line 23-30** to reduce the branch prediction miss rate, with the fact that Clinton won in about half of all US states. You cannot change the code other than line 23-30. (4')
3. For an array of length L, the recursion function **total_clinton** will recur L times. Please rewrite the function body **in line 34-35** to reduce the depth to about L/2, with an optimization similar to loop unrolling. But you cannot use loop in your solution. NOTE that you can show how to invoke your optimized function if it helps simplify your solution. (3')