Problem 1: Pipeline Performance

The following program will calculate %rdx = |*%rax| + |*%rbx| + |*%rcx|

```
f:
      irmovq     $0, %rdx
      mrmovq     (%rax), %rax
      andq       %rax, %rax
      jle        sub_rax
      addq       %rax, %rdx
      jmp        rax_done
sub_rax:
      subq       %rax, %rdx
rax_done:

      mrmovq     (%rbx), %rbx
      andq       %rbx, %rbx
      jle        sub_rbx
      addq       %rbx, %rdx
      jmp        rbx_done
sub_rbx:
      subq       %rbx, %rdx
```

```
rbx_done:

      mrmovq     (%rcx), %rcx
      andq       %rcx, %rcx
      jle        sub_rcx
      addq       %rcx, %rdx
      jmp        rcx_done
sub_rcx:
      subq       %rcx, %rdx
rcx_done:
```

We run this program on the following three pipeline implementation:

1. No prediction
2. Always taken prediction
3. Always not-taken prediction

Assume *%rax=1, *%rbx=2, *%rcx=-1, please show the CPI of the three implementations.

Problem 2: Optimization

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

```
1.  typedef struct {
2.    int trump;
3.    int clinton;
4.    int winner;   // 0 for Trump, 1 for Clinton
5.    int gap;
6.  } state_result;
```

```
7.
8.  typedef struct {
9.    int length;
10.   state_result *data;
11. } rst;
12.
13. int get_length(rst *ra) { return ra->length; }
14. int get_t(rst *ra, int i) {return (ra->data)[i].trump;}
15. int get_c(rst *ra, int i) {return (ra->data)[i].clinton;}
16.
17. void stat(rst *ra, int *total) {
18.   for (int i = 0; i < get_length(ra); i++)
19.     *total = *total + get_t(ra, i) + get_c(ra, i);
20.
21.   state_result *states = ra->data;
22.   for (int i = 0; i < get_length(ra); i++)
23.     if (states[i].trump > states[i].clinton) {
24.       states[i].winner = 0;
25.       states[i].gap = states[i].trump - states[i].clinton;
26.     } else {
27.       // assume Clinton wins if she gets equal or more
28.       states[i].winner = 1;
29.       states[i].gap = states[i].clinton - states[i].trump;
30.     }
31. }
32.
33. int total_trump(state_result *r, int len) {
34.   if (len <= 0) return 0;
35.   return r->trump + total_trump(r + 1, len - 1);
36. }
```

1. Please rewrite the loop in line 18-19 with what you have learned in the class. Comment briefly on the optimization.

2. For an array of length L, the recursion function total_trump 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.

Problem 3: Optimization

```
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 what you have learned in the class. Comment briefly on the optimization. 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.

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.

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