Exercise 6

Problem 1

```
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. }
```

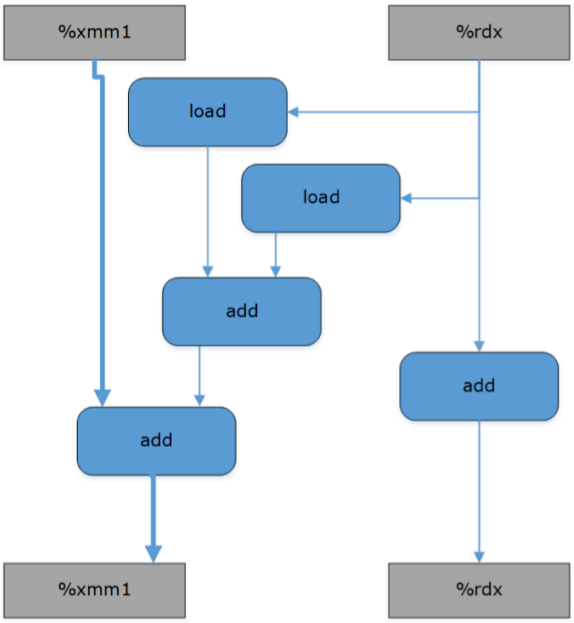
```
24. .Loop:
25.   movss (%rax, %rdx, 8), %xmm0
26.   addss 8(%rax, %rdx, 8), %xmm0
27.   addss %xmm0, %xmm1
28.   addq $2, %rdx
29.   cmpq $rdx, %rbp
30.   jg .Loop
```

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.

```
1.  void copy_array(array_t *dst, array_t *src) {
2.    // Eliminating loop inefficiency
3.    long len = get_length(src);
4.    // reduce function calls
5.    long cap = get_capacity(dst);
6.    // remove unneeded memory references
7.    float *src_data = src->data;
8.    float *dst_data = dst->data;
9.    long niters = (len < cap) ? len : cap;
10.   // loop unrolling and enhancing parallelism
11.   for (long i = 0; i + 1 < niters; i += 2) {
12.     dst_data[i] = src_data[i];
13.     dst_data[i + 1] = src_data[i + 1];
14.   }
```

```
15.   if (i < niters)
16.     dst_data[i] = src_data[i];
17.   dst->length = niters;
18. }
```

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];
```

We have two **load** and two **add** operations. **After the modification,** both **add** operations form a dependency chain between loop registers. While **before the modification**, only one of the **add** operations forms a data-dependency chain between loop registers. The first addition within each iteration can be performed without waiting for the accumulated value from the previous iteration. Thus, we reduce the minimum possible CPE by a factor of around 2.

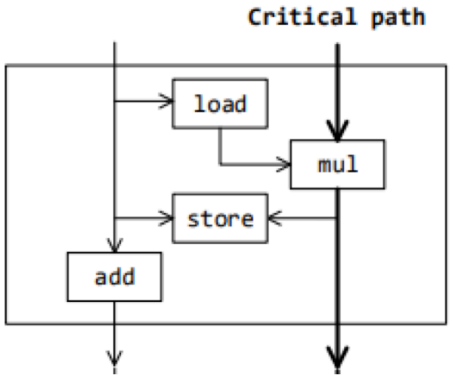
Problem 2

```
1   .L2:
2   mulsd a(,%rax,8), %xmm0  // a is the address of an array.
3   movsd %xmm0, a(,%rax,8)
4   addq  $1, %rax
5   cmpq  %rdx, %rax
6   jl .L2
```

Assume that there is only ONE double-precision multiplication unit in the processor. All other CPU resources are UNLIMITED. The latency and issue time of the units are given in the below table.

operation	Integer		Double-precision	
	latency	Issue	latency	issue
Addition	1	1	2	1
Multiplication	3	1	5	1
Load/Store	3	1	3	1

1. Draw the data flow graph and mark the critical path.



2. Please calculate the CPE on current CPU. If we have UNLIMITED number of multiplication units, how much is CPE?

5
5
There are dependencies between xmm0 so multiple units cannot accelerate it.

3. Now we swap the instruction at line 3 and line 4, please give out the CPE with original LIMITED number of multiplication units and explain your answer.

11. Now the critical path is load-mul-store.