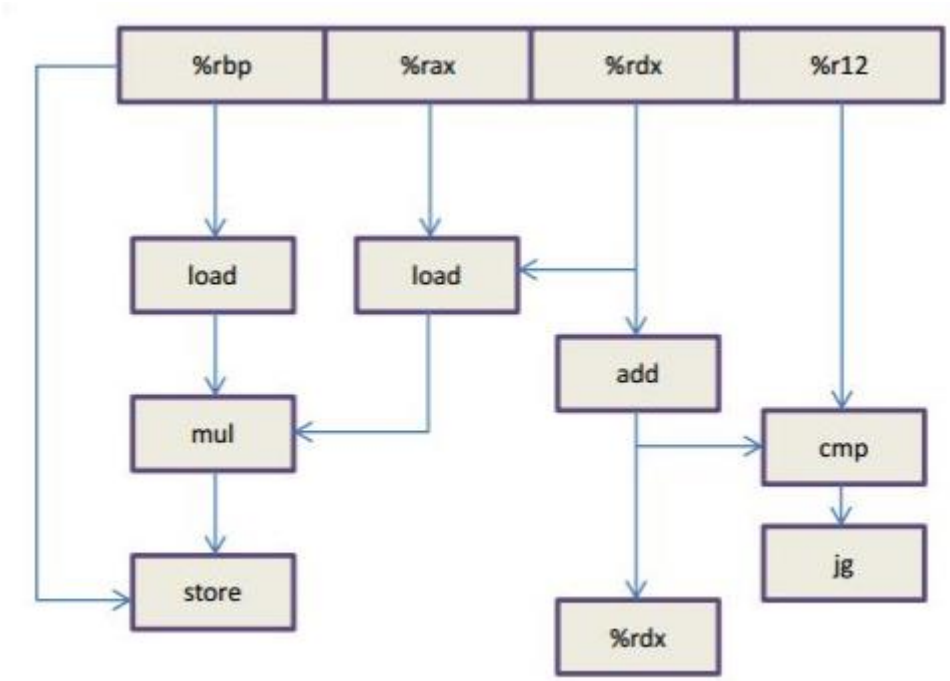
Homework 11

Problem 1

The assembly code generated for the compiled loop of combine3 is shown below:

```
combine3: data_t = float, OP = *
i in %rdx, data in %rax, dest in %rbp
1 .L498:                          loop:
2 movss (%rbp), %xmm0             Read product from dest
3 mulss (%rax, %rdx, 4), %xmm0    Multiply product by data[i]
4 movss %xmm0, (%rbp)             Store product at dest
5 addq $1, %rdx                   Increment i
6 cmpq%rdx, %r12                  Compare i: limit
7 jg .L498                        If >, goto loop
```

Illustrate the code above with data-flow graph like figure 5.14(a) or (b) in textbook. You can use "store" to identify operation in line 4.



Problem 2

How to test the latency of load a value from memory? Try to write a simple program to test it. Assume you can test your program's CPE.

Write a program which uses the value that was loaded in last iteration.

```
typedef struct ELE {
    struct ELE *next;
    long data;
} list_ele, *list_ptr;


long test(list_ptr ls) {
    long len = 0;
    while (ls) {
        len++;
        ls = ls->next;
    }
    return len;
}
```

Problem 3

The following code seems not very good.

```
void sum_array(float *arr, long n, long *sum) {
    float ans = 0;
    for (long i = 0; (i+1) < n; i += 2)
        ans = ans + (arr[i] + arr[i + 1]);
    if (i < n)
        ans += arr[i];
    *sum = ans;
}
```

(1) Please rewrite it.

```
void sum_array(float *arr, long n, long *sum) {
    float acc1 = 0, acc2 = 0, acc3 = 0, ans = 0;
    long i = 0;
    for (; (i+2) < n; i += 3) {
        acc1 += arr[i];
```

```
        acc2 += arr[i+1];
        acc3 += arr[i+2];
    }
    ans = acc1 + acc2 + acc3;
    for (; i < n; i++)
        ans += arr[i];
    *sum = ans;
}
```

(2) The code in line 4 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.