Problem 1:

A[1]: 30 f4 00 02 00 00 0000 0000

[2]: 40 ff 3f ff 3f ff ff ff

[3]: 30 f7 18 00 00 00 0000 0000

[4]: 48 mov q $1, %r9

[5]: 50 A7 0000 0000 0000 0000

[6]: 2b BA

2. 结果: [1] 0x038 [3]: 0x0 [4] 1 [5] 0 [6] 0

[1]: 0x 0000 abcd abcd abcd abcd

Problem 2: Median

1. word Median:

[a>=b&&b>=c : b;

c>=b && b>=a : b;

a>=c && c>=b : c;

b>=c && c>=a : c;

!: a];

2. word Select:

[a>=b && b>=c && c>0 : a;

a>=b && b>=c && c<=0 : c;

a>=c && a>=b && c>0 : a;

a>=c && c>=b && c<=0 : b;

b>=a && a>=c && c>0 : b;

b>=a && a>=c && c<=0 : c;

b>=c && c>=a && c>0 : b;

b>=c && c>=a && c<=0 : a;

c>=a && a>=b && c>0 : c;

c>=a && a>=b && c<=0 : b;

c>=b && b>=a && c>0 : c;

c>=b && b>=a && c<=0 : a;]

及3、bool OPD=
[a>0 && b<=0 && c<=0 : true;
a<=0 && b>0 && c<=0 : true;
a<=0 && b<=0 && c>0 : true;
a>0 && b>0 && c>0 : true;
1 = false;]

Problem 四
1、a)：ret 指令只有到 D stage 才可以获取真正的下一条指令的 PC，并且能错误的执行一条指令。

b)：与 ~~Mispredicted Branch~~ processing ret 类似，都是在 F 时不知道正确的 PC 而错误的执行指令，并会利用 bubble 来消除错误指令。

2、a)：[1] = E_icode　　[2]：E_valA

b)：WHY：在 D stage 插入一个 bubble 来消除被错误执行的下一条指令

- WHAT：[1]：E_icode　　[2] = I[I]XX

四、3、[1] = E_icode == I[I]XX && e_Cnd

[2]：Bubble　　[3]：Bubble

Problem 4:

1.
```
long sum_vec_list(vec_list *list, vec_rec *vec) {
    long len1 = get_vec_rec_len(vec);
    long len2 = get_vec_list_len(list); // 减少函数调用
    if (len1 < len2) return;
    long *vecdata = vec->data;
    vec_rec *listdata = list->data; // 减少对重复地址访问
    ~~long i=0;~~ long j=0;
    for (long i=0; i< len2; ++i) {
        vecdata[i]=0;
        ~~long for(j=0~~; long len3 = get_vec_rec_len(&list->data[i]);
                // 减少程序调用
        for (j=0; j<len3-1; j+=2) {    // loop unrolling
            ~~vecdata[i] += list.data[i].data[j]~~;
            long val = listdata[i].data[j] + listdata[i].data[j+1];
            vecdata[i] += val;
        }
        if (j< len3). vecdata[i] += listdata[i].data[j];
    }
    return 0;
}
```

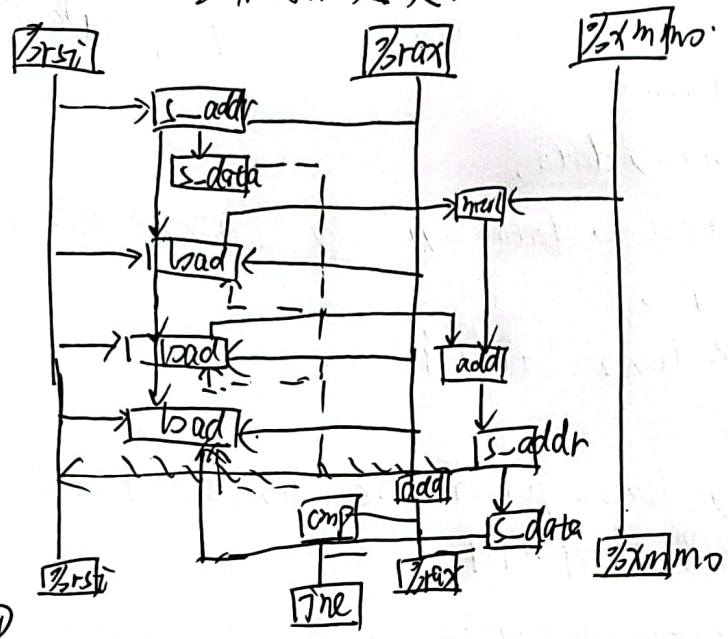
2. speedup ratio $= \dfrac{1}{0.4 + \frac{0.6}{5}} = \dfrac{1}{0.52} = 1.92$

3. 不一定能实现和原程序相同的功能。

在原程序中~~可能会出~~由于每次都要读出result指针中的值，故可能由于 ~~mem~~ memory alising 而重复取出某个 data[i] 的值，即有：data[i] += data[i] 情况可能发生，而优化后的程序不会出现这个情况，因为 tmp 与 data[i] 不会 memory aliasing. 即优化后程序一定是从0开始加的

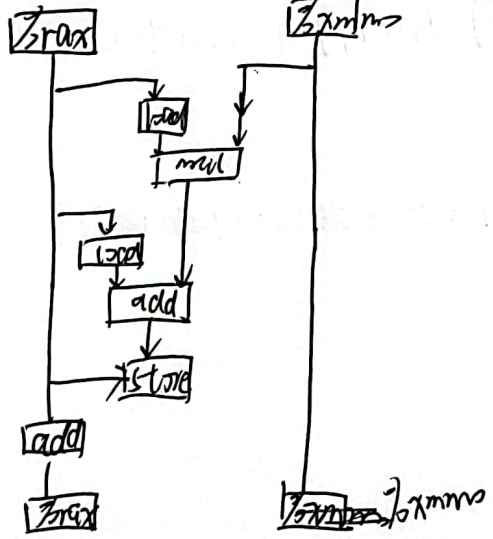
Problem 5:

1、%xmm1 的作用：作为临时中间变量参与运算，保证了 k 在每次 loop 时 ~~Problem 5~~ 的值保持不变。

2、 

3、estimated CPE = 11.

4、简化后的 data-flow graph 为：critical path 为：



critical path 为 mul - add - store，故 CPE = 5 + 3 + 3 = 11

4、① 不能改善 CPE，② 同样这段汇编是使用 loop unrolling (2x1) 进行优化，但是单独使用 loop-unrolling 并没有改变数数据之间的 dependency，故 critical path 不变，CPE 不变。

⑤
```
void k_prefix_sum(vec_t *arr, double k) {
    long len = arr->len;
    double *data = arr->data;
    for (i=1; i<len; i++) {
        double num = data[i];
        double num1 = data[i-1];
        data[i] += k*num1;
        data[i+1] += k*(num + k*num1.);
    }
}
```

利用 loop unrolling 以及中间变量 num, num1 消除 data[i+1] 与 data[i] 之间 dependency.