Question 1:

Suppose we wish to write a procedure that computes the inner product of two vectors u and v. An abstract version of the function has a CPE of 14–18 with x86- 64 for different types of integer and floating-point data. By doing the same sort of transformations we did to transform the abstract program combine1 into the more efficient combine4, we get the following code:

```
1   /* Inner product.  Accumulate in temporary */
2   void inner4(vec_ptr u, vec_ptr v, data_t *dest)
3   {
4       long i;
5       long length = vec_length(u);
6       data_t *udata = get_vec_start(u);
7       data_t *vdata = get_vec_start(v);
8       data_t sum = (data_t) 0;
9
10      for (i = 0; i < length; i++) {
11          sum = sum + udata[i] * vdata[i];
12      }
13      *dest = sum;
14  }
```

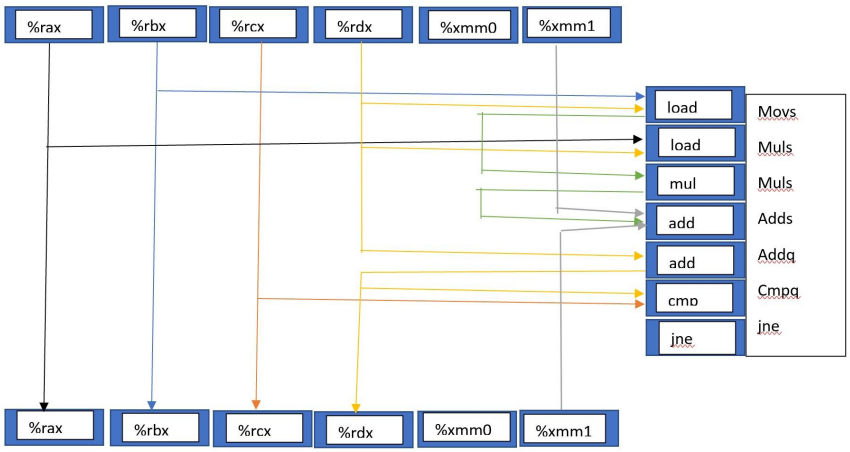
Our measurements show that this function has CPEs of 1.50 for integer data and 3.00 for floating-point data. For data type double, the x86-64 assembly code for the inner loop is as follows:

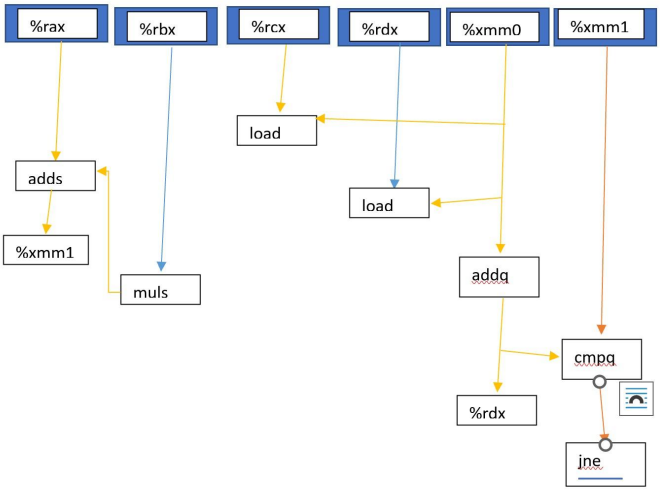
```
    Inner loop of inner4.  data_t = double, OP = *
    udata in %rbp, vdata in %rax, sum in %xmm0
    i in %rcx, limit in %rbx
1   .L15:                                   loop:
2     vmovsd  0(%rbp,%rcx,8), %xmm1           Get udata[i]
3     vmulsd  (%rax,%rcx,8), %xmm1, %xmm1     Multiply by vdata[i]
4     vaddsd  %xmm1, %xmm0, %xmm0             Add to sum
5     addq    $1, %rcx                        Increment i
6     cmpq    %rbx, %rcx                      Compare i:limit
7     jne     .L15                            If !=, goto loop
```

Assume that the functional units have the characteristics listed in Figure 5.12.

A. Diagram how this instruction sequence would be decoded into operations and show how the data dependencies between them would create a critical path of operations, in the style of textbook Figures 5.13 and 5.14.

%rax = vdata
%rbx = udata
%rcx = length
%rdx = i
%xmm0 = value of the loaded data
%xmm1 = sum





B. For data type double, what lower bound on the CPE is determined by the critical path?

4/3 + 3/3 = 7/3 = 2.33

C. Assuming similar instruction sequences for the integer code as well, what lower bound on the CPE is determined by the critical path for integer data?

3/3 + ⅓ = 4/3 = 1.33

D. Explain how the floating-point versions can have CPEs of 3.00, even though the multiplication operation requires 5 clock cycles.

By the latency indicate, the multiplication operation requires 4 or 5 cycles.
Sum = sum + udata[i] * vdata[i]
The value of i executes from 0 to 3
So, sum = ((sum + udata[0] * vdata[0]) + (udata[1] * vdata[1]) + (udata[2] * vdata[2]) + (udata[3] * vdata[3]))
The number of times the value is executed is up to 4 cycles and the integer i executed up to 5 times, leading to the point where the floating point can have CPE of 3.00.

Question 2:
Write a version of the inner product procedure described in Question 1 that uses 6 × 6 loop unrolling. Our measurements for this function with x86-64 give a CPE of 1.06 for integer data and 1.01 for floating-point data.

What factor limits the performance to a CPE of 1.00?

The CPI, the average number of clock cycles to execute each instruction, limits the performance.

```
Void inner_6x6(vec_ptr u, vec_ptr v, data_t *dest){
        Long i;
        Long length = vec_length(u);
        Long limit = length;
        Data_t *udata = get_vec_start(u);
        Data_t *vdata = get_vec_start(v);
        Data_t acc0 = (data_t) 0;
        Data_t acc1 = (data_t) 0;
        Data_t acc2 = (data_t) 0;
        Data_t acc3 = (data_t) 0;
        Data_t acc4 = (data_t) 0;
        Data_t acc5 = (data_t) 0;
        for(i = 0; i< limit ; i+=6){
                Acc0 += udata[i] * vdata[i]
                Acc1 += udata[i+1] * vdata[i+1];
                Acc2 += udata[i+2] * vdata[i+2];
                Acc3 += udata[i+3] * vdata[i+3];
                Acc4 += udata[i+4] * vdata[i+4];
                Acc5 += udata[i+5] * vdata[i+5];
        }
        for(; i<= length; i++){
                Acc0 += udata[i] * vdata[i] ;
        }
        *dest = acc0 + acc1 + acc2 + acc3 + acc4 + acc5;
}
```

Question 3:
Write a version of the inner product procedure described in Question 1 that uses 6 × 1a loop unrolling to enable greater parallelism. Our measurements for this function give a CPE of 1.10 for integer data and 1.05 for floating-point data.

```
Void inner_6x6a(vec_ptr u, vec_ptr v, data_t *dest){
        Long i;
        Long length = vec_length(u);
        Long limit = length;
        Data_t *udata = get_vec_start(u);
        Data_t *vdata = get_vec_start(v);
        Data_t sum = (data_t) 0;
        for(i = 0; i< limit ; i+=6){
                sum += ((udata[i] * vdata[i]) + (udata[i+1] * vdata[i+1]) + (udata[i+2] * vdata[i+2]) +
                        (udata[i+3] * vdata[i+3]) + (udata[i+4] * vdata[i+4]) + (udata[i+5] * vdata[i+5]));
        }
        for(; i<= length; i++){
                sum += udata[i] * vdata[i] ;
        }
        *dest = sum;
}
```