Worksheet 6 Solutions

1. What are some optimizations that can be made to the following function?

```
void cs33fun(char* Midterm, char* Grade, int* Final, int n) {
      for (int i = 0; i < (strlen(Midterm)); i++) {
            strcat(Grade, Midterm);
            for (int j = 0; j < n; j++)
                  for (int k = 0; k < i; k++)
                        Final[j] += strlen(Grade);
      }
}
```

There are many ways this function can be optimized, including but not limited to:

- The innermost loop can be replaced with the statement:
    - `Final[j] += i * strlen(Grade);`
- Move `strlen(Midterm)` outside of the loop

There are several things you SHOULDN'T do:

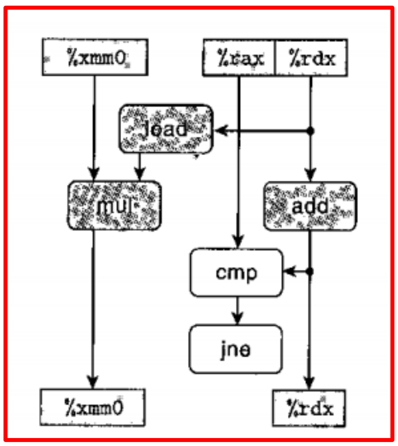
1. Based on "Procedure calls" - Move `strcat` out of the loop
    - `strcat` is required for the logic of the function
2. Based on "Procedure calls" - Move `strlen(Grade)` outside of the outermost loop (and nothing else)
    - The string `Grade` changes over each iteration of the outermost for loop
    - BUT can be moved outside of the middle loop
        - Since `strlen(Grade)` increments by `strlen(Midterm)` during each outermost iteration, can actually be moved outside the outermost loop if handled correctly

2. For the following assembly, draw a data flow graph and identify the critical path.

```
.L1:
        vmulsd      (%rdx), %xmm0, %xmm0
        addq        $8, %rdx
        cmpq        %rax, %rdx
        jne         .L1
```

Given the following table, what is the lowest bound latency to execute n iterations of this loop for integer operations? What about floating point operations?

Latency Table (CPE)	int	double
Arithmetic (except multiply)	1	3
Multiply	2	4
Load/Store	1	1



The critical path is determined by the multiply operation in the loop. For integer operations, from the table, we can see that the latency is 2 cycles for multiplication and for n iterations of the loop, it would take 2n cycles to execute. Similarly for floating point operations, the latency is 4 cycles, and it would take 4n cycles to execute n iterations of the loop.

3. 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:

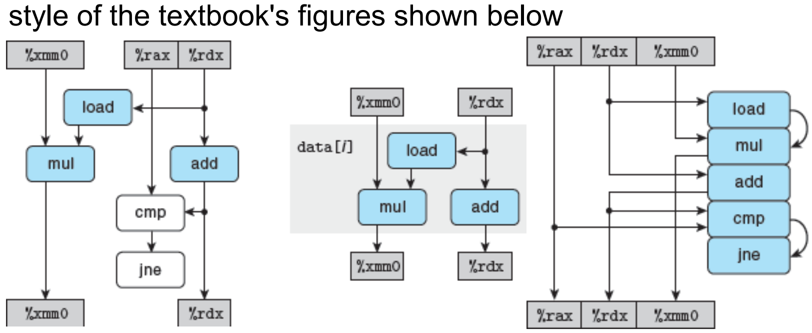
```
/* Inner product. Accumulate in temporary */
void inner4(vec_ptr u, vec_ptr v, data_t* dest) {
      long i;
      long length = vec_length(u);
      data_t *udata = get_vec_start(u);
      data_t *vdata = get_vec_start(v);
      data_t sum = (data_t) 0;
      for (i = 0; i < length; i++) {
            sum = sum + udata[i] * vdata[i];
      }
      *dest = sum;
}
```

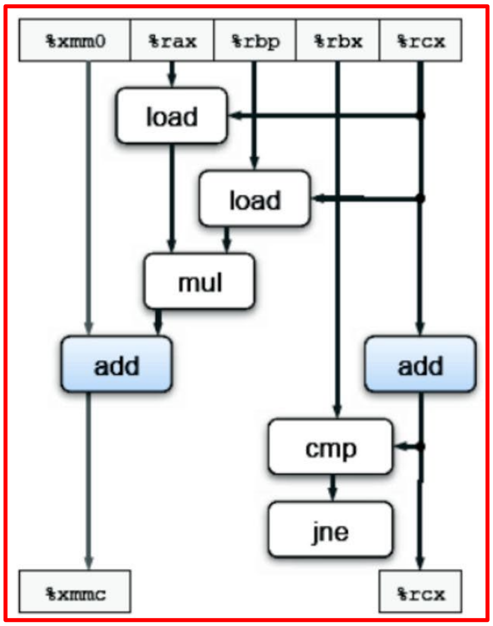
Our measurements show that this function has CPEs of 1.50 for `int` data and 3.00 for `double` data. Use the latency table from the previous question for the latencies of operations.

For data type double, the x86-64 assembly for the inner loop is as follows:

```
.L15:                                                ;loop:
  vmovsd  0(%rbp, %rcx, 8), %xmm1                    ; Get udata[i]
  vmulsd  (%rax, %rcx, 8), %xmm1, %xmm1              ; Multiply by vdata[i]
  vaddsd  %xmm1, %xmm0, %xmm0                        ; Add to sum
  addq     $1, %rcx                                  ; Increment i
  cmpq     %rbx, %rcx                                ; Compare i:limit
  jne      .L15                                      ; If !=, goto loop
```

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 the textbook's figures shown below





The multiplication operation is not on any critical path.

b. For data type double, what lower bound on the CPE is determined by the critical path?
The critical path is formed by the addition operation updating variable `sum`. This puts a lower bound on the CPE equal to the latency of floating-point addition.

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?
For integer data, the lower bound would be just 1.00. Some other resource constraint is limiting the performance.

d. Explain how the floating-point version can have CPEs of 3.00, even though the multiplication operation requires more than 3 clock cycles.
The multiplication operations have longer latencies, but these are not part of a critical path of dependencies, and so they can just be pipelined through the multiplier.