1. Processors based on the VLIW architecture rely on the compiler to encode multiple operations into a long instruction word so hardware can safely schedule these operations at run-time on multiple functional units without dependency analysis. Why might the compiler be better able to find instructions that do not have dependencies than the run-time hardware of a superscalar computer?

2. If ADD and SUB take one cycle and MUL takes 7 cycles to execute, then what would be the first instruction to complete using Tomasula's algorithm on the following program?

MUL R6, R4, R8
ADD R2, R6, R7
STORE R2, 8(R6)
ADD R2, R3, R4
SUB R4, R5, R2
LOAD R4, 16(R4)
ADD R1, R2, R3

b) How does register renaming help the STORE instruction save the correct R2 value to memory while still allowing later instructions to execute?
3. For a typical program on a traditional computer, more time is spent doing procedure/method calls than anything else. Why are procedure calls so time consuming?

4. What ways are parameters passed during a procedure call?

5. The Itanium assembly language can eliminate branch instructions by using predicate registers (e.g., p2, p3) as the following code:

```
if (R1 == R2)
    R3 = R3 + R1
else
    R3 = R3 - R1
end if
```

However, why does this technique only improve performance if the “then” and “else” bodys are small sections of code?