1. To keep a pipeline full, parallelism among instructions must be exploited by finding sequences of unrelated instructions that can be overlapped in the pipeline. Table 1 shows the latencies of Floating-Point (FP) operations, in which the last column is the number of intervening clock cycles needed to avoid a stall. We assume the standard five-stage integer pipeline, so that integer loads have a delay of 1 clock cycle, integer ALU operations have 0. Given a straightforward MIPS code, not scheduled for the pipeline, look like this:

```
Loop:  L.D    F0, 0(R1)        ; F0 = array element
       ADD.D  F4, F0, F2       ; add scalar in F2
       S.D    F4, 0(R1)        ; store result
       DADDUI R1, R1, #-8      ; decrement pointer
                               ; 8 bytes (per DW)
       BNE    R1, R2, Loop     ; branch R1!=R2
```

   (a) Without any scheduling, how many cycles per iteration we need to execute this loop? Explain your answer. (10 pts)
   (b) By scheduling this loop, how many cycles per iteration we need to execute this loop? Explain your answer. (10 pts)
   (c) Can we eliminate all stall cycles to execute this loop by the loop unrolling and loop scheduling techniques? How to do it? (10 pts)

Table 1. Latencies of FP operations

Instruction producing result	Instruction using result	Latency in clock cycles
FP ALU op	Another FP ALU op	3
FP ALU op	Store double	2
Load double	FP ALU op	1
Load double	Store double	0

2. We can use Amdahl's Law to measure the performance of a computer.
   (a) Explain the Amdahl's Law. (10 pts)
   (b) For a DSP processor, implementations of floating-point multiplication (Mult.D) instructions vary significantly in performance. Suppose Mult.D instructions are responsible for 30% of the execution time in a benchmark. One proposal is to enhance the Mult.D hardware and speedup this operation by a factor of 10.

The other alternative is to enhance the compiler which selects a sequence of add and shift instructions to perform the same operation. This approach improves the average CPI by 0.4 times, but the instructions count gets worse by 2 times. Compare these two design alternatives. (10 pts)

3. Explain the following questions.
   (a) For most pipelines supporting dynamic scheduling, why should we split the instruction decode stage (ID) into two stages: Issue, and Read operands? (10 pts)
   (b) Show how Tomasulo's algorithm performs dynamic pipeline scheduling. (10 pts)
   (c) Explain why the hardware speculation can exploit more instruction-level parallelism? How to use the Reorder-Buffer (ROB) to perform hardware speculation? (10 pts)

4. The importance of multiprocessors is growing today.
   (a) The problems—insufficient parallelism and long-latency remote communication—are the two biggest performance challenges in using multi-processors. How to attack these problems? (10 pts)
   (b) Suppose we have an application running on a 16-processor multiprocessor, which has 300 ns time to handle reference to a remote memory. For this application, assume that all the references, except those involving communication, hit in the local memory hierarchy. Processors are stalled on a remote request, and the processor clock rate is 2GHz. If the base CPI (assuming that all references hit in the cache) is 0.6, how much faster is the multiprocessor if there is no communication versus if 0.2% of the instructions involve a remote communication reference? (10 pts)