CDA 5106 - Advanced Computer Architecture

Homework 2 (50 pts)

Reference: Chapter 2 textbook

For all questions, use the following code and latencies:

```
Loop: LD     F2,0(Rx)
I0:   DIVD   F8,F2,F0
I1:   MULTD  F2,F6,F2
I2:   LD     F4,0(Ry)
I3:   ADDD   F4,F0,F4
I4:   ADDD   F10,F8,F2
I5:   ADDI   Rx,Rx,#8
I6:   ADDI   Ry,Ry,#8
I7:   SD     F4,0(Ry)
I8:   SUB    R20,R4,Rx
I9:   BNZ    R20,Loop
```

Latencies beyond single cycle	
Memory LD	+4
Memory SD	+1
Integer ADD, SUB	+0
Branches	+1
ADDD	+1
MULTD	+5
DIVD	+12

(Hint: for latency explanation, see textbook, Case Study 1 of Chapter 2, question 2.2)

Q1 (15 pts): What would be the baseline performance (in cycles, per loop iteration) of the code sequence above, if no new instruction's execution could be initiated until the previous instruction's execution had completed?

Ignore front-end fetch and decode. Assume for now that execution does not stall for lack of the next instruction, but only one instruction/cycle can be issued. Assume the branch is taken, and that there is a one cycle branch delay slot. Show your calculations.

Q2 (15 pts): How many cycles would the loop body in the code sequence above require if the pipeline detected true data dependencies and only stalled on those, rather than blindly stalling everything just because one functional unit is busy? What is the speedup of this approach over Q1? Show the code with stalls inserted and specify what instruction each stall relates to.

Hint: Remember, the point of the extra latency cycles is to allow an instruction to complete whatever actions it needs, in order to produce its correct output. Until that output is ready, no dependent instructions can be executed. So the first LD must stall the next dependent instruction for 4 clock cycles. The MULTD produces a result for its successor, and therefore must stall 5 clocks, and so on. However, independent operations may be issued and executed on the consecutive clock cycle.

Q3 (20 pts): Consider the code produced in Q2. Imagine you are performing work of a scheduling compiler that rearranges the instructions to reduce time wasted on stalls. Schedule the code you have produced in Q2, preserving data dependencies. Assume no loop unrolling and one instruction issued per cycle. Show all your work, including stall cycles and an instruction each stall relates to. How many cycles will the scheduled code spend in each loop? What is the speedup over the original version of the code (from Q1)?