Module 5 (Lectures 21-24) Pipelined processors

1. Consider the 5 stage processor pipeline that we discussed. As we observed, this pipeline can potentially speed-up program execution by a factor of 5, when compared to an equivalent non-pipelined processor. What would the speed-up be considering a program for which every 5th instruction suffers a 1 cycle `bubble' due to a hazard?
2. Consider the MIPS 1 code fragment on the 5 stage processor pipeline that we discussed. Many of the instructions in this fragment are dependent on each other. Mark the data dependencies, labeling each by its type (RAW, WAR, etc) and identifying those that would NOT be handled by the pipeline implementation techniques (e.g., result forwarding) that we discussed.
```
LW     R7, -8(R7)
ADD    R3,  R5,  R7
SUB    R5,  R3,  R7
OR     R7,  R3,  R7
```
3. The manual of a particular computer provides the following warnings for programmers regarding the processor pipeline: There are two load delay slots. There is one branch delay slot. There must be at least 2 instructions between a floating point computation instruction and a floating point store instruction that uses the value computed by the computation operation for correct operation to occur. You are given the inner loop of a program below. The instructions whose mnemonics start with the letter `F' are floating point instructions that use floating point registers F0..F31.
```
Loop:  FLOAD     F0,      0(R1)
       FLOAD     F2,      0(R2)
       FADD      F4,      F0,      F2
       FSTORE    0(R1),   F4
       ADDI      R1,      R1,      8
       ADDI      R2,      R2,      8
       BLE       R1,      R3,      Loop
```
   How many cycles does one iteration of the loop take in its present form, once it has been corrected with insertion of NOPs (no-operation instructions) to take into account the warnings mentioned above? Do static instruction scheduling to improve the loop as much as you can. How many cycles does each iteration now take?
4. The manual of a particular computer provides the following warnings for programmers regarding the processor pipeline: There is one load delay slot. There are two branch delay slots. There must be at least 2 instructions between a floating point computation instruction and a floating point store instruction that uses the value computed by the computation operation for correct operation to occur. You are given the inner loop of a program below. The instructions whose mnemonics start with the letter `F' are floating point instructions that use floating point registers F0..F31.
```
saxpy: ADD   R5,      R1,      R3
       FLOAD          F2,      0(R5)
```

```
          FMULT      F2,     F0,     F2
          ADD   R6,  R1,     R4
          FLOAD      F4,     0(R6)
          FADD  F2,  F2,     F4
          FSTORE     0(R5),  F2
          ADDI  R1,  R1,     4
          BLT   R1,  R2,     saxpy
```

How many cycles does one iteration of the loop take in its present form once it has been corrected (by insertion of NOPs) to take into account the warnings mentioned above? Do static instruction scheduling to improve the loop as much as you can. How many cycles does each iteration now take?

5. Some pipelined processors handle control hazards using dynamic branch prediction; they incorporate hardware that predicts whether a given branch will be taken or not, based on the recent behaviour of branches in the program. Dynamic branch predictors with very high prediction accuracy (prediction accuracy: the percentage of branch executions that it predicted correctly). A simple example of this idea is the One Bit Branch Predictor. In this dynamic branch prediction scheme, the hardware maintains a Branch Prediction Table (BPT), which contains a 1 bit entry for each branch in the program. Initialized to 0, the entry is set to 1 if the branch is executed and taken, or reset to 0 if the branch is executed and not taken. In order to predict the next behaviour of the given branch, the hardware looks up the BPT entry for that branch. If the table entry is 0, it predicts that the branch will not be taken; if the table entry is 1, it predicts that the branch will be taken. What would be the prediction accuracy of a One Bit Branch Predictor for the following C program fragment?

```
int i, j=0;
for (i=0; i<100; i++)
        if (i % 2) j++;
        else j+=2;
```

You should assume that the compiled version of this program contains 2 branch instructions, one to implement the *for* loop (i.e., transfer of control from the end of the loop body to the loop header) and the other to implement the *if-then-else* (i.e., transfer of control to the else part after checking the condition).