CS704 – Advanced Computer Architecture-II

Due Date: 9th June, 2012

Assignment 2

Instructions to Solve Assignments

The purpose of assignments is to give you hands on practice. It is expected that students will solve the assignments themselves. Following rules will apply during the evaluation of assignment.

- Cheating from any source will result in zero marks in the assignment.
- Any student found cheating in any two of the assignments submitted will be awarded "F" grade in the course.
- No assignment after due date will be accepted.

Question 1: Total Points (10+5+5=20)

Following code lines are written in a high level language:
a = c + d;
b = c + e;

The corresponding instructions for MIPS are:

LW R1, 0(R0)
LW R2, 4(R0)
ADD R3, R1, R2
SW R3, 12(R0)
LW R4, 8(R0)
ADD R5, R1, R4
SW R5, 16(R0)

These instructions are to be executed on a pipelined processor with forwarding.

(a) Identify hazards by showing the execution of these instructions per cycle bases.
(b) Reorder these instructions to avoid any pipeline stalls.
(c) How many cycles are saved after executing the reordered instructions?

Question 2: Total Points (10+10+10=30)

Scheduling branch delay slots (see the three ways in Figure 2.1) can improve performance. Assume a single branch delay slot and an instruction pipeline that determines branch outcome in the second stage.

(a) For a delayed branch instruction, what is the penalty for each branch delay slot scheduling scheme if the branch is taken and if it is not taken, and what condition, if any, must be satisfied to ensure correct execution?
(b) A cancel-if-not-taken branch instruction (also called branch likely and implemented in MIPS) does not execute the instruction in the delay slot if the branch is not taken. Thus, a compiler need not be as conservative when filling the delay slot. For each branch delay slot scheduling scheme, what is the penalty if the branch is taken and if it is not taken, and what condition, if any, must be satisfied to ensure correct execution?
(c) Assume that an instruction set has a delayed branch and a cancel-if-not-taken branch. When should a compiler use each branch instruction and from where should the slot be filled?

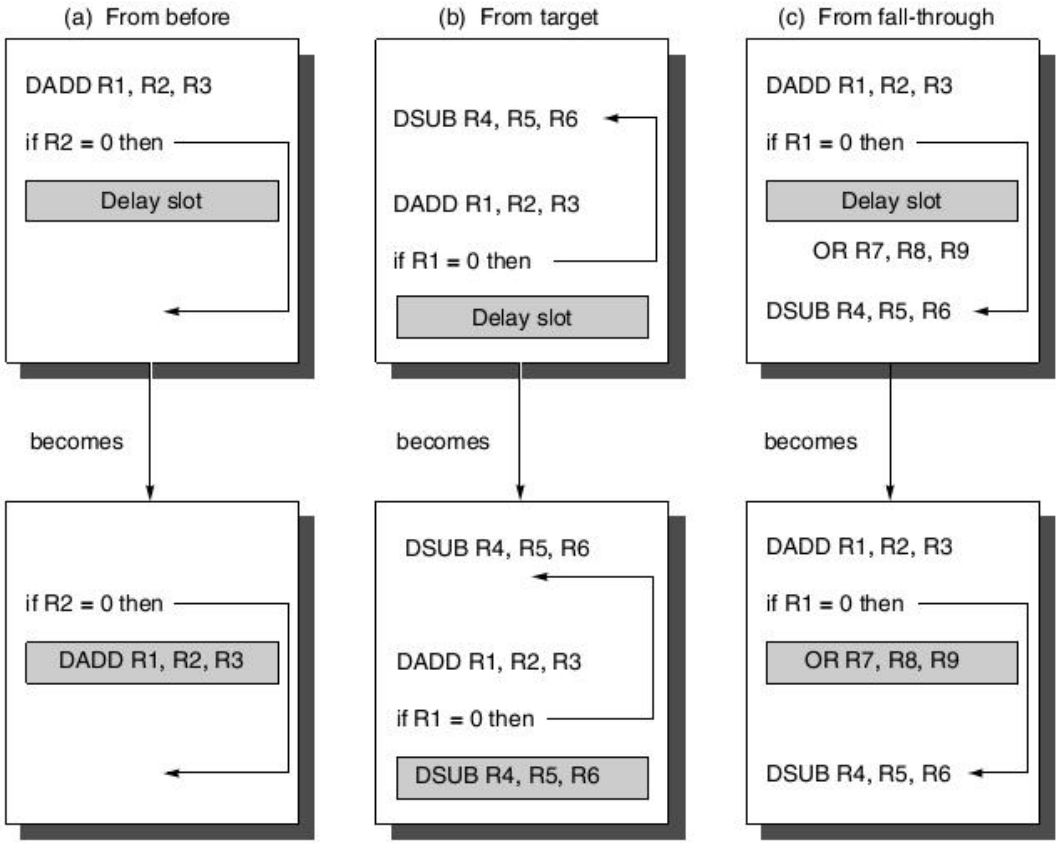

Figure 2.1 Scheduling the branch delay slot. The top box in each pair shows the code before scheduling; the bottom box shows the scheduled code. In (a) the delay slot is scheduled with an independent instruction from before the branch. This is the best choice. Strategies (b) and (c) are used when (a) is not possible. In the code sequences for (b) and (c), the use of R1 in the branch condition prevents the DADD instruction (whose destination is R1) from being moved after the branch. In (b) the branch delay slot is scheduled from the target of the branch; usually the target instruction will need to be copied because it can be reached by another path. Strategy (b) is preferred when the branch is taken with high probability, such as a loop branch. Finally, the branch may be scheduled from the not-taken fall-through as in (c). To make this optimization legal for (b) or (c), it must be OK to execute the moved instruction when the branch goes in the unexpected direction. By OK we mean that the work is wasted, but the program will still execute correctly. This is the case, for example, in (c) if R7 were an unused temporary register when the branch goes in the unexpected direction.

Question 3: Total Points (10+10=20)

Tomasulo's algorithm has a disadvantage: Only one result can complete per clock, per CDB.

(a) Use the hardware configuration from Figure 3.1 and the FP latencies from Figure 3.2. Find a code sequence of no more than 10 instructions where Tomasulo's algorithm must stall due to CDB contention. Indicate where this occurs in your sequence.

(b) Generalize you result from part (a) by describing the characteristic of any code sequence that will eventually experience structural hazard stall given n CDBs.

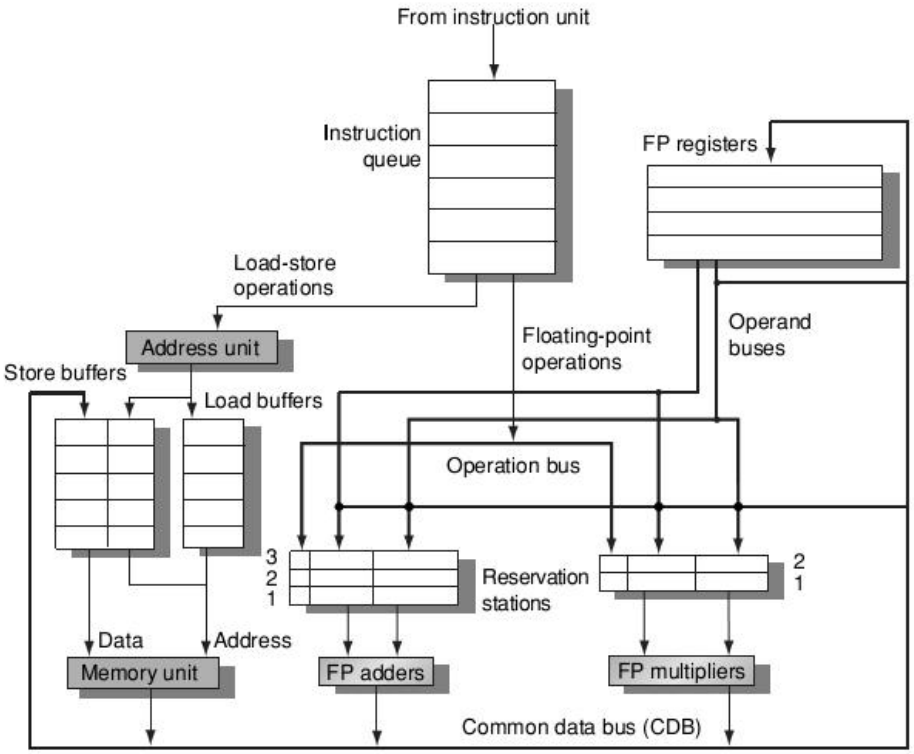

Figure 3.1 The basic structure of a MIPS floating-point unit using Tomasulo's algorithm. Instructions are sent from the instruction unit into the instruction queue from which they are issued in FIFO order. The reservation stations include the operation and the actual operands, as well as information used for detecting and resolving hazards. Load buffers have three functions: hold the components of the effective address until it is computed, track outstanding loads that are waiting on the memory, and hold the results of completed loads that are waiting for the CDB. Similarly, store buffers have three functions: hold the components of the effective address until it is computed, hold the destination memory addresses of outstanding stores that are waiting for the data value to store, and hold the address and value to store until the memory unit is available. All results from either the FP units or the load unit are put on the CDB, which goes to the FP register file as well as to the reservation stations and store buffers. The FP adders implement addition and subtraction, and the FP multipliers do multiplication and division.

Instruction producing result	Instruction using result	Latency in clock cycles
FP multiply	FP ALU op	6
FP add	FP ALU op	4
FP multiply	FP store	5
FP add	FP store	3
Integer operation (including load)	Any	0

Figure 3.2 Pipeline latencies where latency is number of cycles between producing and consuming instruction.

Question 4: Total Points (15)

Consider the following code

```
for (i=1; i<=100; i=i+1)
{
        A[i+1] = A[i] + C[i];          /* S1 */
        B[i+1] = B[i] + A[i+1];        /* S2 */
}
```

Write an optimal assembly code for the loop. Is it safe to use loop un-rolling here? Why or why not?

Question 5: Total Points (15)

Based on the paper "**Instruction Issue Logic for High-Performance Interruptable Pipelined Processors**", explain how the Register Update Unit (RUU) is used to resolve data dependencies and maintain precise interrupts.