COMPUTER ARCHITECTURE LABORATORY,

STUDENT GUIDE

SESSION 3: Control hazards in DLX

Review slides and examples about Control Hazards and the Techniques of Delayed Branch and Static Branch Prediction. In this session we will see some strategies supported by the simulators in the lab to deal with Control Hazards.

Delayed branch

The **DlxvSim (DLXV3.1)** simulator uses the so called Delayed Branch technique. In the **WinDLXV 1.0** simulator, this technique is also used, although it must be selected in the configuration menu.

When we described the DLX pipeline in previous sessions, we saw that, for jump instructions the PC is written in the **ID** stage with the destination address when the branch is taken. In that case, the pipeline has a Delay Slot (DS) as shown in figure below.

n	bnez **r1**, etiq	IF	***ID***	EX	MEM	WB		
n+1	Inst. util o NOP		IF	ID	EX	MEM	WB	
n+2 o Etiq	Inst. X			**IF**	ID	EX	MEM	WB

With the Delayed branch technique, the instruction in the DS is/are **always** executed no matter whether the branch is taken or not. The idea is that the compiler should choose an instruction from before the jump itself, since it will be always executed and completed even when the branch is not taken.

This works for conditional and unconditional branches. Therefore, if there is no useful instruction that can take advantage of the DS, the compiler must place a useless non-harming instruction, typically a NOP. But then, the cycle of the DS is wasted (in a loop, that would be one cycle per iteration).

Question: What will happen if the instruction after the jump placed at the end of a loop is "trap #6"? --Check if what you've supposed happens or not--

Branch Prediction

In the **WinDLXV 1.0** simulator it is possible to select one of two techniques (Menu Configuración/Saltos):

- Delayed branch, as we have seen in the case of DLXV3.1. or
- **Branch Prediction Not Taken**.

In the later case, the assumption is made that the jump will NOT be TAKEN and the immediate instruction to the branch instruction is loaded into the pipeline, which will be executed only if the jump is not taken. Upon branch resolution, then:

- If prediction is wrong, instruction in HR is not allowed to complete and the cycle is not used for good.
- If prediction is correct and the branch is NOT taken, instructions are completed with no stalls.

Please note that if you use Branch Prediction the instruction after the branch (useful or NOP) is NOT executed ALWAYS as with delayed branch so do not use any NOPs anymore

Activities

1. For the optimized version of the "**reverse.s**" program (sum of vectors that appears in reverse order) of the previous practice, make the necessary modifications to take advantage of the delay slot using the delayed branch technique (replace the **NOP** with a useful instruction while keeping no stalls due to RAW data hazards). Recalculate the theoretical gain vs. the initial unordered program (with stalls and NOPs) for N = 200.
2. For N integers input vector A, B C and D, develop a program called **condsuma.s** based on the schema shown below so that:

   if D(i)=0 then S(i) = A(i)+C(i) and if D(i)!)!=0 then S(i) = A(i)+B(i).

   a) In the case of Delayed Slot, design the code causing as many stalls as possible, indicating in which instructions there are RAW data stalls specifying stall cycles (if you did the diagrams on page 2 of the guide of the previous session correctly, you will see that in one case there are two RAW stalls). Use NOPs for DS

      IMPORTANT: theoretically calculate the number of cycles it will take to execute the program (on paper, it is the most important part of the exercise).

      To do this, complete the declaration of the data area and use the following algorithm that corresponds to a code without optimization and that takes **227** cycles with **52** data stalls in DLXV3.1 it should take **exactly that** and work correctly, else something is wrong .

   b) Explain how those 52 stalls and 227 cycles are obtained.

```
R9 <-- M(N)                 A:      .word   .....
R7<-- 0                     B:      .word   .....
loop:                       D:      .word   0, 5, 0, 0, 6, 5, 9, 0, 0, 0, 2, 0, 3. 44, 87
  R1 <-- A(r7)              C:      .word   ....
  R2 <--B(R7)               S:      .space  ...
  R5 <--D(R7)               N:      .word   15
  if (R5 != 0) goto contin
  R2 <-- C(R7)
contin:
   R3 <--R2+R1
   M(S+R7) <-- R3
   R7 <--R7 +4
   R9 <--R9-1
    if (R9 != 0) goto loop
---fin--
```

   c) Optimize it and take full advantage of DS. Note: to eliminate the 2-cycle stall it is necessary to insert two instructions, but in DLXV3.1, when placing only one there are no more stalls: this is a tool error (in WinDLX this is consistent). Find the theoretical Gain (on paper) extrapolating the calculations for N = 500 elements (assume 50% of zeros in D).

3. Starting from the original non-optimized code, make the necessary modifications to run the program using **Branch prediction not taken** in WinDLXV, verifying that the program is executed correctly. Explain what happens, compute cycles taken in execution and compare the performance of both techniques for this specific case, qualitatively and quantitatively.