## **NATIONAL UNIVERSITY OF SINGAPORE**

## **Department of Electrical and Computer Engineering**

## **CG2028 Computer Organization**

**Tutorial 6 Solutions**

**Pipelining Basics**

1. Insert NOPs in the code below to ensure correct operation in a 5-stage pipelined ARM processor.

L1: AND R8, R9, R11

L2: ADD R1, R2, #8

L3: SUB R2, R5, R7

L4: LDR R3, [R1, #4]

L5: CMP R0, #0

L6: BNE SKIP

Ans:

There is a data hazard when the source register for an instruction is written by (i.e., is the destination register of) an instruction still in the pipeline. If there is a data hazard, to ensure correct operation, we need to insert an appropriate number of NOPs such that the two instructions causing data hazard are spaced by at least 3 instructions.

Control hazards are caused by branch instructions or writes to R15. We need to insert 4 NOPs after each branch instruction to ensure program correctness.

AND R8, R9, R11

ADD R1, R2, #8

SUB R2, R5, R7

NOP

NOP

LDR R3, [R1, #4]

CMP R0, #0

BNE SKIP

NOP

NOP

NOP

NOP

R1 is a source of L4 and the destination for L2. Hence, they need to be spaced by 3 instructions. Only 2 NOPs needed (instead of 3) as L3 is independent and acts as a spacer.

Note that L2 and L3 do not have a hazard as long as L3 follows L2 - L2’s source is L3’s destination, not the other way around.

1. Can the instructions in the code above be rearranged to reduce the number of NOPs to be inserted to eliminate data hazards?

Ans:

We can attempt to use independent instructions instead of NOPs for spacing instructions causing data hazards. However, we need to ensure that the rearrangement does not introduce any hazards.

Here, L1 can be moved down (swap L1 and L2) - the destination register of L1 is not one of the source registers of L2; the destination registers of L2 is not one of the source registers for L1; L1 has a different destination register as compared to L2. There are some other possible complications associated with rearranging instructions, related to flags and memories. We do not have such issues here though.

L5 (swap L4 and L5) can be moved up. Note that moving L5 before L4 is fine since L4 does not affect flags.

ADD R1, R2, #8

SUB R2, R5, R7

AND R8, R9, R11

CMP R0, #0

LDR R3, [R1, #4]

BNE SKIP

NOP

NOP

NOP

NOP

1. What is the total number of cycles required for executing the code you obtained in Q.2 if the branch is not taken?

Ans:

4 cycles are required to fill the pipeline. From cycle 5, one instruction completes every cycle. The total number of cycles = 4 + number of instructions = 4+10 = 14. The time taken is the same even if the branch is taken, as the 4 NOPs are inserted at compile-time and the processor executes them irrespective of whether the branch is taken.

Note for Q2 and Q3:

The number of NOPs can be reduced even further by moving some instructions from before the branch to after the branch. For example, L4 can be moved to follow L6. Apart from the fact that L4 has no data dependencies w.r.t L5 and L6, we should note that L4 is an instruction which should be executed no matter whether the branch is taken or not. Since L4 (after being pushed below L6) enters the pipeline well before L6 changes the PC value (to BTA) if the branch is taken, L4 proceeds to completion normally and will write the correct result to R3. However, we should be mindful of the potential hazards if R3 is a source for an instruction within 3 word addresses of the BTA, or within 3 instructions after the branch in the original program order - we have to check both these possibilities.

Up to 4 independent instructions (L5 is not one of them) can be moved from before the branch to after the branch if the issues similar to that mentioned in the previous sentence does not arise.