UNIVERSITÉ LIBRE DE BRUXELLES

DÉPARTMENT D'INFORMATIQUE

ELEC-H473
MICROPROCESSOR ARCHITECTURES

Lab 3 - RiSC16: internal processor behaviour

Authors:
Hakim BOULAHYA
Jacky TRINH
Yasin ARSLAN
Youcef BOUHARAOUA

1 Question 1 : Tests

We were asked for the assignment to choose an operator among: $<>$, $\leq$, $\geq$.

We choose the $<$ operator. For the Question 2 we were asked to write a code for the operators, and for this first question a determination of a set of test vectors and a justification of our choice was necessary.

A vector is just a binary representation of a number.

Our algorithm compares the MSB of each vector; if there's a match, it checks the next one. If the bits are different, we have the solution.

To test the correctness of our algorithm, we decided to test some extreme cases of the representation :

- a positive number with a negative one.
  - 1000000000000000 (-32768)
  - 0000000000000001 (1)
- zero with a negative number.
  - 0000000000000000 (0)
  - 1000000000000000 (-32768)
- zero with a positive number.
  - 0000000000000000 (0)
  - 0000000000000001 (1)
- Two numbers which are equal.
  - 0111111111111111 (32767)
  - 0111111111111111 (32767)
- Two positive numbers.
  - 0000000000100010 (34)
  - 0000000001000010 (66)
- Two negative numbers.
  - 1111111111010110 (-42)
  - 1111111110011100 (-100)

We also tested those cases in the other sens.

2 Question 2 : Solution Register

2.1 2.1

The solution is stored in Register 3 (1 if reg 1 ¡ reg 2, else : 0).

2.2 2.2

The solution is stored in Register 3 (1 if reg 1 ¡ reg 2, else : 0).

3 Question 3.3 : Conclusion

With the new instructions of IS[1], it allowed us to simplify the multiplication. We mainly gained a lot of process time to check for overflow because of the new ADD instruction that allows to detect it.

It saved a lot of instruction since the main optimisation had to be done in the overflow process during the 32 bits addition and the shifting of 32 bits values.

We also use less memory, as we didn't had to check for the overflow, we could used more registers to store directly the counter and avoid multiple SW/LW instructions to be able to process the multiplication.

With IS[2] it was even easier because the work was already done for us with the MUL instruction, which already do the multiplication in one simple instruction.

In this case, the new instructions lowered the complexity of the algorithm. IS[2] is more optimised than IS[1] which is more optimised than the standard RiSC16's instructions. As the objective of the project is to lower the number of instructions, having an overflow detection helps us in that way, and having only one instruction for the multiplication also allow use really minimise the CI.