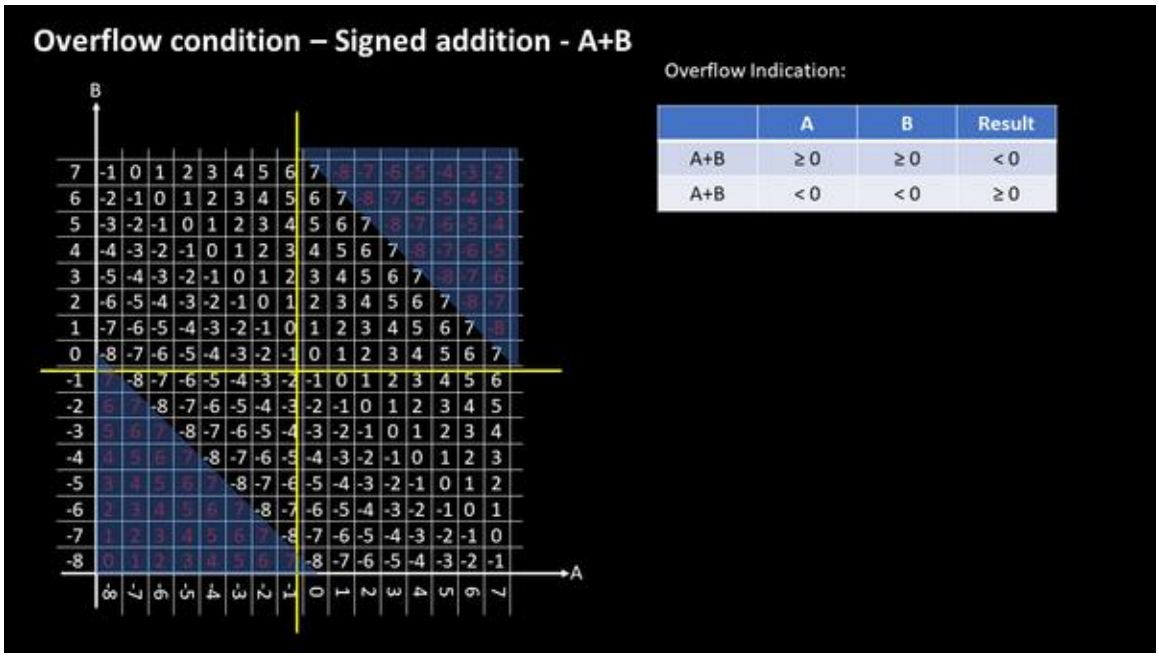
Wanna quick solution to identify overflows? – Use RISC-V branches

KUNAL GHOSH

Nothing more powerful than RISC-V branches. And that's exactly the intent of newly pre-launched RISC-V ISA Part 1b below –

https://www.udemy.com/vsd-riscv-instruction-set-architecture-isa-part-1b/

Look at below matrix graph to understand the conditions of overflow for signed addition –

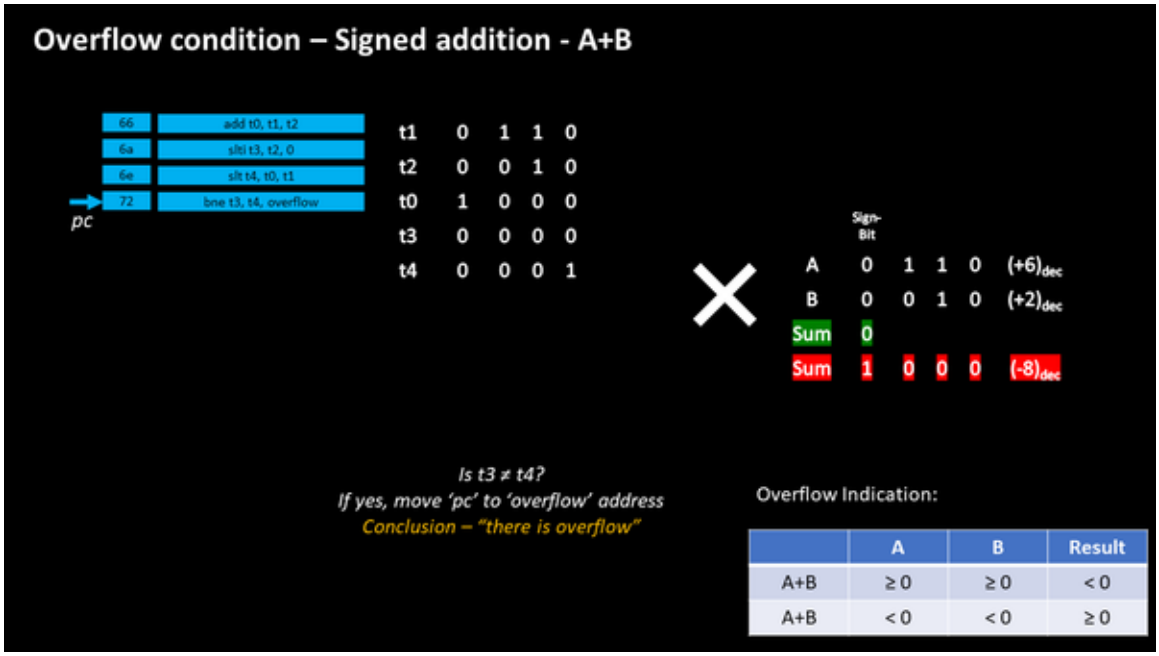


For example, if we are talking about of a 4-bit processor, where msb represents 'sign' ***('1' represents -ve number and '0' represents +ve number)***, then we are left with 3 bits, which can be used to represent numbers from range -8 to +7. From the above image, the x-axis and y-axis represents numbers from -8 to 7 (which is nothing but the range of numbers which can be represented by 4-bit processor).

The contents of matrix shown in middle in white is the 'correct' sum of 2 numbers. For example, take a number '+6' (0110) from x-axis, and '-4' (1100) from y-axis. Now find the intersection of these numbers, and you will see the number '+2' in middle in white area. This says that '6' + '-4' = '+2' (0010). Try that for couple of more numbers whose result lie in white area

Next, take number say '+6' (0110) from x-axis and '+2' (0010) from y-axis. '6' + '2' = '+8' (1000). Ooops. The binary representation of '+8' which 1000 represents a negative number '-8' (remember from above highlighted line, i.e. '1' represents -ve number). (Look

for my RISC-V Part 1a course to find out how to convert a binary pattern to its equivalent -ve numbers using 2's complement method)
Just imagine your personal calculator on your laptop (which has a 4-bit processor in it) giving a value of -8 for 6 + 2…. Grrrr….Annoying…Right…..That brings us to the heading of this blog. Wanna quick solution to detect overflow?
Consider below RISC-V assembly program to identify 'overflows' and take some action (like print an error message)



Let's keep '+6' in register t1 and '+2' is register t2. The first command 'add t0, t1, t2' will **add** t1+t2 and **store result** in t0. So t0 gets '1000' which is -8. The next command 'slti' which is '**set if less than immediate**'which means if t2 < 0 , store logic '1' in register t3 else store logic '0'. t2 which holds +2 is not less '0'. So t3 is set to logic '0' Next, 'slt' is just like 'slti', only it deals with register i.e. if t0 < t1, then store logic '1' in register t4 else store logic '0' in register t4.
The final instruction which is '**bne**'is '**branch if not equal**' which is essentially means to branch to a different address location pointed by 'overflow', if t3 is not equal to t4, which is the case here. So, here's where the overflow is detected, and an error message will be printed out, informing 'you are out of range' (does this ring a bell? You must have seen this error message while working on windows or Linux machine)

Wasn't that interesting? If you are looking for some more cool examples of how your laptop or computer functions, look for below course on RISC-V ISA part 1b which is just pre-launched for $12 only today…

https://www.udemy.com/vsd-riscv-instruction-set-architecture-isa-part-1b/

Part 1a is here:

https://www.udemy.com/vsd-riscv-instruction-set-architecture-isa-part-1a/

The initial response has been great and one of the reviewer says below –

"***The author has given a clear cut definition of all points. The summary of instructions makes it very interesting for me. Eagerly awaiting the next part of the course.***"

Here's last 5 coupons for TCL scripting Part 1 and Part 2 course –

Part 1:

https://www.udemy.com/vsd-tcl-programming-from-novice-to-expert/

Part 2:

https://www.udemy.com/vsd-tcl-programming-from-novice-to-expert-part-2/

Enjoy the course and happy Learning…