all code in https://github.com/EugeniusK/ecad-distribution2025/tree/main/riscv/assembly/optimisations

The number of cycles running the code in div.s is defined from

- the cycle that the first line of code inside div.s ran (after `jal` and bubbles from calling `div`)
- the cycle of the final bubble caused by the `return` (after `jalr` and bubbles from calling `return` inside div.s)

By focusing on these lines, there are only 3 "unavoidable" bubbles after return (`jalr`)
All other bubbles are from loops (`jal` - 3 bubbles) or comparisons (`blt` - 3 bubbles).

Version 00 - same as submitted to tick 3, 36 lines of code

start: 24
end: 513
cycles: 490

189 bubbles occured in the divider code (bubbles 10~198)

Branch was not taken 34 times (no bubbles)
Branch was taken 31 times (3 bubbles per)

`JAL` was called 31 times (3 bubbles per)

`JALR` was called 1 time (3 bubbles per)

$$34 * 0 + 1 * 3 + 31 * 3 + 1 * 3 = 189$$

From observation `JAL` from `if (R < D)` caused bubbles almost all the time. I have no clue why I did a `BLT` instead of `BGT` like in the pseudocode provided.

Version 01 - replace `BLT` with `BGE` , 39 lines of code

start: 24
end: 429
cycles: 406

105 bubbles occured in the divider code (bubbles 10~114)

Branch was not taken 62 times (no bubbles)
Branch was taken 3 times (3 bubbles per)

`JAL` was called 31 times (3 bubbles per)

`JALR` was called 1 time (3 bubbles per)

$62 * 0 + 3 * 3 + 31 * 3 + 1 * 3 = 105$

Replacing `BLT` with `BGE` and some rearranging code significantly reduced the number of bubbles (and cycles).

Now, the only possible optimisations are reducing bubbles (only from `JAL` with for loops) or improving the logic.

Unravelling the loop is an option as this reduces the number of `JAL`. Furthermore, the number of iterations of the loop (32) is insignificant so the code can be copied and pasted with minor impact on program size.

Version 02 - unravel loop, ~440 lines of code

start: 24
end: 282
cycles: 259

18 bubbles occured in the divider code (bubbles 10~27)

Branch was not taken 31 times (no bubbles)
Branch was taken 2 times (3 bubbles per)

`JAL` was called 3 times (3 bubbles per)

`JALR` was called 1 time (3 bubbles per)

$31 * 0 + 2 * 3 + 3 * 3 + 1 * 3 = 18$

At this point, it is impossible to reduce the number of branches while maintaining the logic of an `if` statement.

Only optimisation thinkable at the moment is using immediates instead of storing `1 << i` in a register and using it in a later calculation. Due to RISC-V, only immedates of only 12 bits can be used and as is two's complement, negative number has to be used for adding 2**11. So only for i in $[0, 11]$, immediates can be used.

Version 03 - use immediates, ~430 lines of code

start: 24

end: 270

cycles: 247

18 bubbles occured in the divider code (bubbles 10~27)

Branch was not taken 31 times (no bubbles)
Branch was taken 2 times (3 bubbles per)

`JAL` was called 3 times (3 bubbles per)

`JALR` was called 1 time (3 bubbles per)

$$31 * 0 + 2 * 3 + 3 * 3 + 1 * 3 = 18$$

I have absolutely no clue on what can be done further, stop here.