Assignment 3, Question 1
Group 24

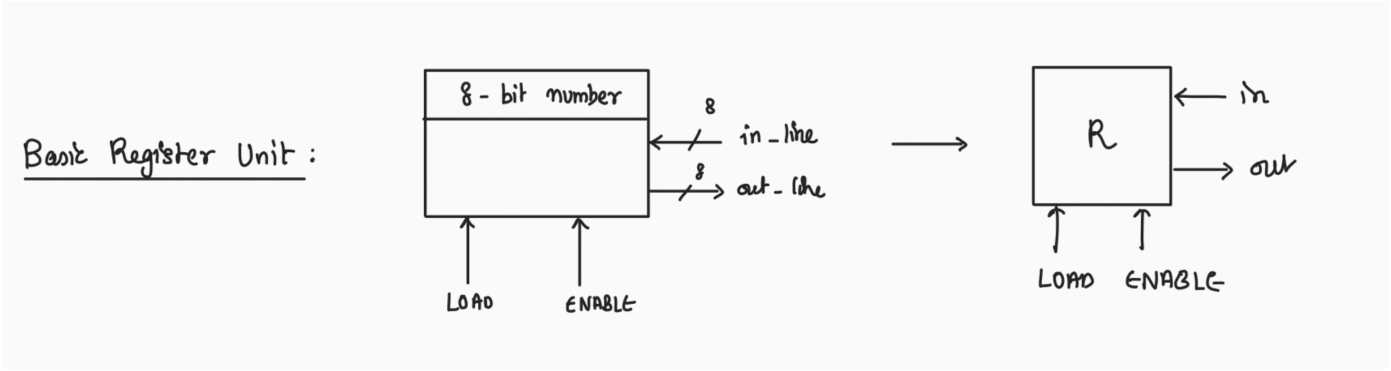
Gorantla Thoyajakshi - 21CS10026
Ashwin Prasanth - 21CS30009

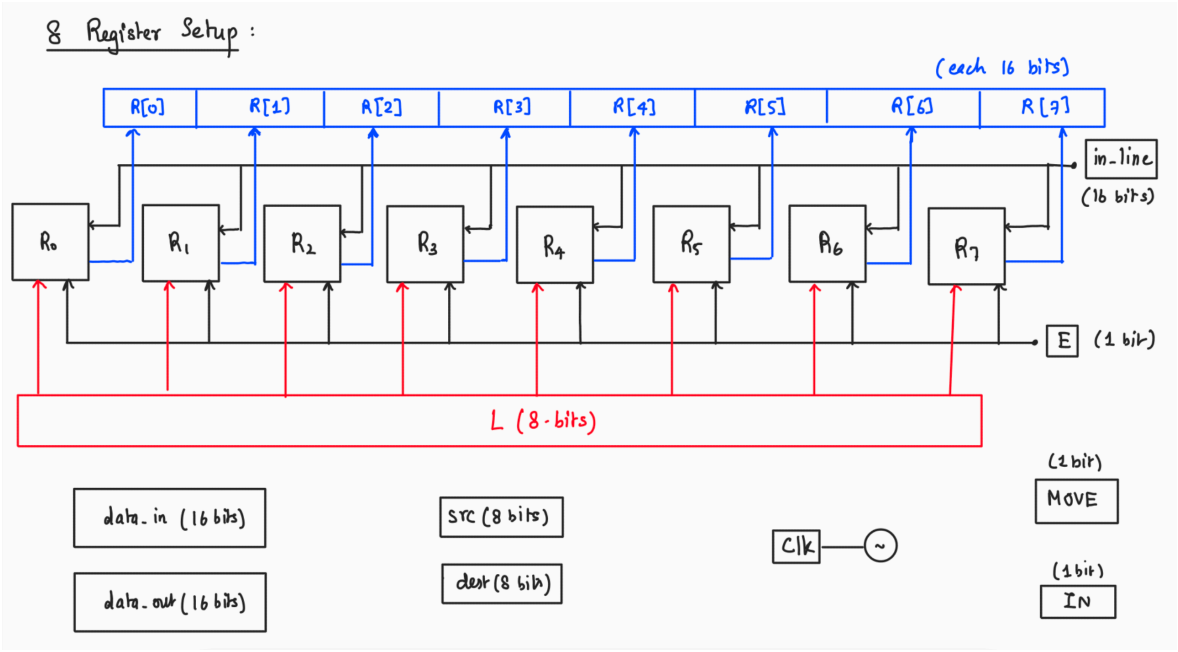
- We defined a base module called `register` in `Q1_Grp_24_reg.v`, of which we made **8 instances**
- Each such register has 3 inputs - `LOAD`, `ENABLE` and `in_line` and 1 output - `out_line`
- Whatever input is fed into the register is done through `in_line`, and the value in the register is read from `out_line`
- When `LOAD` is high, the register assumes the value in `in_line`
- When `ENABLE` is high, the value at the register can be obtained at `out_line`, else, `out_line` is at high impedance state

- In the top-level module in `Q1_Grp_24_control_reg.v`, we have 6 inputs - `src`, `dest`, `MOVE`, `IN`, `data_in`, `clk` and 1 output - `data_out`
- `src` and `dest` are 3 bits each, denoting the register number of the source and destination register
- If `MOVE` is high, the value in `src` register is moved to `dest` register
- If `IN` is high, the value provided at `data_in` is moved to `dest` register
- Both cannot be high at the same time, so error is thrown and program is terminated
- We have made 8 instances of the `register` module, with each of them having their own `LOAD` parameter (in an 8 bit array called `L`), and the same `ENABLE` parameter (`E`) and same input line (`in_line`). Their values are obtained from each of their `out_line`s and stored in an array `R`

- Whenever `MOVE` or `IN` is executed, depending on which register is required, the corresponding `LOAD` value is set to high through `L` and all registers are enabled through `E`. The output, or the value that is moved, is stored in `data_out` (as output). At the end, all `L` and `E` are reset to low
- The whole process is controlled by a clock `clk`

- Here is a simple schematic diagram of the setup:





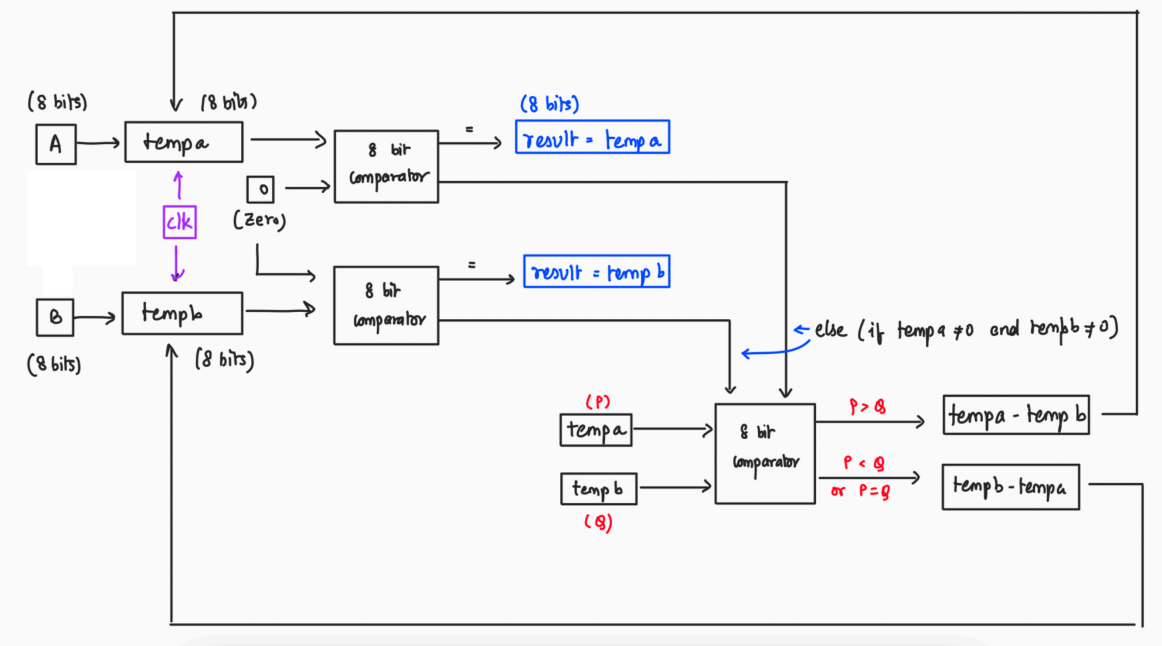
- There would additionally be some logic involved to access the registers based on `src` and `dest` values, to move the corresponding value from `data_in`, to store the final value in `data_out`, to process whether `MOVE` is to be executed or `IN`, and also to use the `clk` to drive the process

- In the testbench in `Q1_Grp_24_tb.v`, we have tested a few cases of `MOVE` and `IN`, and at the end tested the case of when `MOVE` and `IN` are both set to high (which would throw error)

Assignment 3, Question 2
Group 24

Gorantla Thoyajakshi - 21CS10026
Ashwin Prasanth - 21CS30009

- We have done all the computation in a single module called `gcd`, in `Q2_Grp_24_gcd.v`
- We have 3 inputs `a`, `b`, and `clk`, and one output `result`
- Additional registers `tempa` and `tempb` are used to do operations that will lead to the GCD, and they will have initial values of `a` and `b`
- Whenever the value of `a` or `b` changes (ie, new test case is run), the values of `tempa` and `tempb` are set to the new values of `a` and `b`, and `result` is reset to zero
- `tempa` and `tempb` are iteratively reduced based on whichever is greater, and this is done synchronously with the clock cycle of `clk`, and it is done until one of them assumes the value of zero, in which case the result is the other one.
- Here is a simple schematic diagram of the setup:



- Additionally there would be some logic involved in assigning the correct value to result, handling the condition when `tempa` and `tempb` are both not equal to zero, finding `tempa-tempb` or `tempb-tempa` (using subtractor module), and using `clk` to drive the process, etc.

- In the testbench in `Q2_Grp_24_tb.v`, we have tested some simple cases of GCD computation, including corner cases like one of them being zero, or both being equal