This is an example to show how to translate programs into RTL.
A simple program to calc gcd.

Algorithm 1 Euclidean algorithm

```
 1: procedure GCD($a, b$)
 2:     while $a \neq b$ do
 3:         if $a > b$ then
 4:             $a := a - b$
 5:         else
 6:             $b := b - a$
 7:         end if
 8:     end while
 9:     return $a$
10: end procedure
```

Segment into basic blocks. Draw a flow chart.

$$\begin{tikzcd}
\#2 \arrow[d] \arrow[dddd, bend right] \\
\#3 \arrow[d] \arrow[dd, bend right] \\
\#4 \arrow[uu, bend right] \\
\#6 \arrow[uuu, bend right] \\
\#9
\end{tikzcd}$$

Map the basic blocks $\{\#2, \#3, \#4, \#6, \#9\}$ to program counter values $PC = \{0, 1, 2, 3, 4\}$.
This is a state machine $< S, \Delta, I, F >$.
State space $S = \{PC \times A \times B\}$
Initial states set $I = \{s \mid s \in S, pc = 0\}$
Final states set $F = \{s \mid s \in S, pc = 4\}$
Write out state shift rule Δ (as APL?!).

$$\begin{array}{lclcll}
pc = 0 & \wedge & a \neq b & \rightarrow & pc := 1 & \\
pc = 0 & \wedge & a = b & \rightarrow & pc := 4 & \\
pc = 1 & \wedge & a > b & \rightarrow & pc := 2 & \\
pc = 1 & \wedge & a \leqslant b & \rightarrow & pc := 3 & \\
pc = 2 & & & \rightarrow & pc := 0, & a := a - b \\
pc = 3 & & & \rightarrow & pc := 0, & b := b - a
\end{array}$$

Describe the state machine in Verilog.

```
module gcd(a, b, reset, return_val, ready, clk)
    input [31:0] a;
    input [31:0] b;
    input reset;
    output [31:0] return_val;
    output ready;
    input clk;

    reg [31:0] mem_a;  // S
    reg [31:0] mem_b;  // S
    reg [31:0] pc;     // S
    reg [31:0] result;
    reg halt;

    always @(posedge clk)
      begin
        if (reset == 0)
        begin
          mem_a <= a;  // I
          mem_b <= b;  // I
          pc <= 0;     // I
          halt <= 0;
        end
        else if (halt == 0)
        begin
               if (pc == 0 && mem_a != mem_b) begin pc <= 1; end // Delta
          else if (pc == 0 && mem_a == mem_b) begin pc <= 4; end // Delta
          else if (pc == 1 && mem_a > mem_b)  begin pc <= 2; end // Delta
          else if (pc == 1 && mem_a <= mem_b) begin pc <= 3; end // Delta
          else if (pc == 2) begin pc <= 0; mem_a <= mem_a - mem_b; end // Delta
          else if (pc == 3) begin pc <= 0; mem_b <= mem_b - mem_a; end // Delta
          else if (pc == 4) begin result <= mem_a; halt <= 1; end // F
        end
      end

    assign return_val = result;
    assign ready = halt;
endmodule
```

It's done.