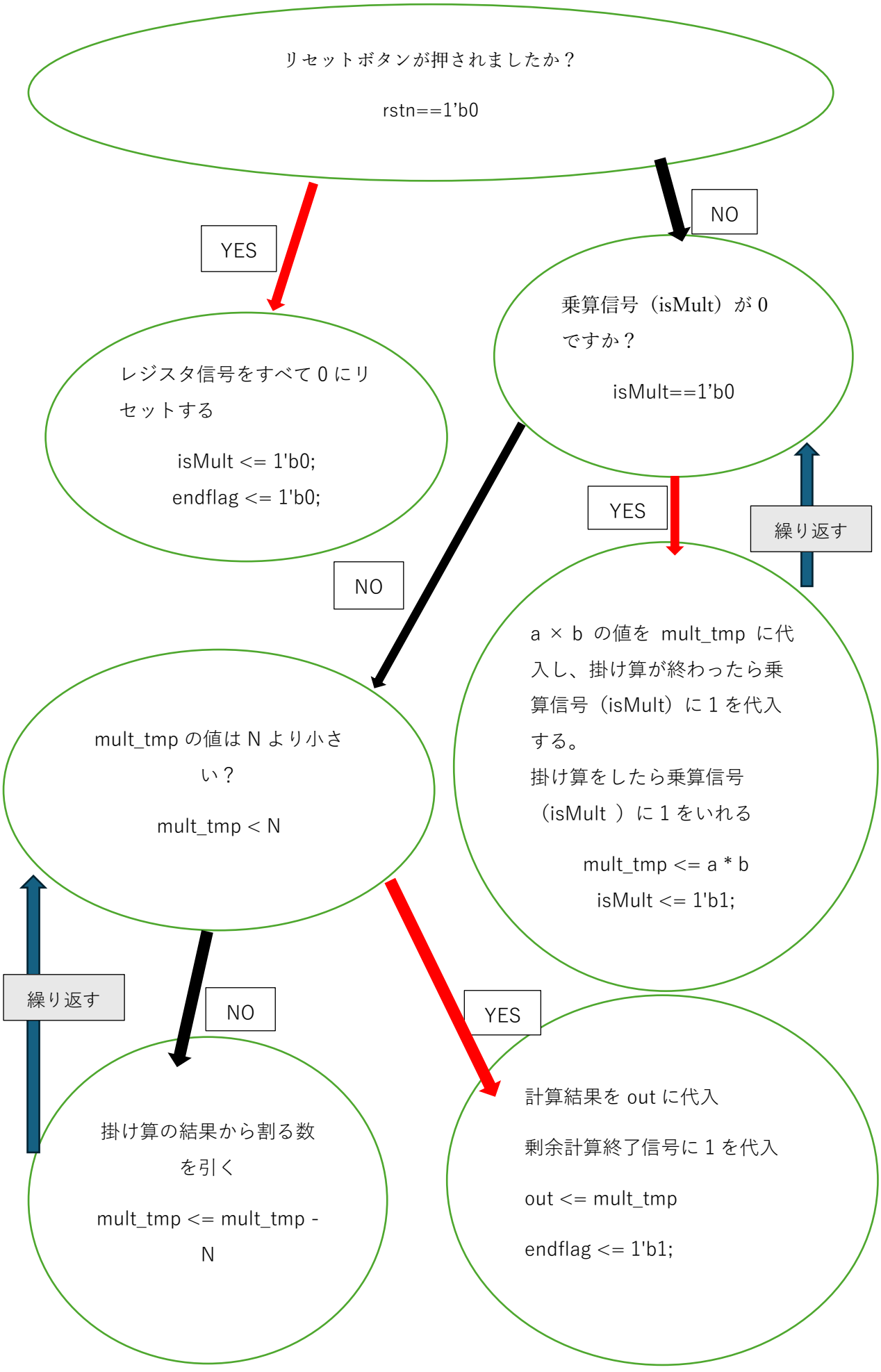
8bit プログラム


リセットボタンが押されましたか？
rstn==1'b0
YES
NO
レジスタ信号をすべて 0 にリセットする
isMult <= 1'b0;
endflag <= 1'b0;
乗算信号（isMult）が 0 ですか？
isMult==1'b0
YES
繰り返す
NO
a × b の値を mult_tmp に代入し、掛け算が終わったら乗算信号（isMult）に 1 を代入する。
掛け算をしたら乗算信号（isMult ）に 1 をいれる
mult_tmp <= a * b
isMult <= 1'b1;
mult_tmp の値は N より小さい？
mult_tmp < N
繰り返す
NO
YES
掛け算の結果から割る数を引く
mult_tmp <= mult_tmp - N
計算結果を out に代入
剰余計算終了信号に 1 を代入
out <= mult_tmp
endflag <= 1'b1;


次は解答ですが、
開く前にもう一度
考えてみませんか?


あと一息だけ
考えてみ
ませんかか?

8bit のプログラム

```
always @(posedge clk or negedge rstn) begin
        if ( !rstn ) begin
            isMult <= 1'b0;
            endflag <= 1'b0;
            out <= 9'd0;
            mult_tmp <= 18'd0;
        end else if ( !isMult ) begin // 乗算をする
            mult_tmp <= a * b;
            isMult <= 1'b1;

end else if ( isMult ) begin // 剰余算をする
            if ( mult_tmp < N ) begin
                out <= mult_tmp[8:0];
                endflag <= 1'b1;
            end else begin
                mult_tmp <= mult_tmp - N;
            end
        end
    end
```

初期化

リセットボタンが押された場合、次のレジスタ信号をすべて 0 に初期化します：

isMult = 0 （掛け算を行わない状態）
endflag = 0 （計算が終了していない状態）
out = 0 （出力の初期化）
mult_tmp = 0 （掛け算の一時結果を初期化）

isMult が 0 のとき、a * b（a_multiplication * b_multiplication）を計算して、mult_tmp に代入します。

前の ModularExponentiationSimple_9bit で、どちらも入力された 8 ビットの値が入っています。つまり、平文同士の掛け算をしています。

mult_tmp（掛け算の結果）が N（割る数）より小さくなったら、計算終了します。

isMult（掛け算終了信号）が 1 のとき、つまり平文を所定の回数掛け終わったので、次はいよいよ余りを求めます。このプログラムでは、引き算を繰り返して余りを計算しています。

16bit のプログラム

基本的にはビット数を変えるだけで対応できます。ただし、Simple のプログラムを動かした際に、暗号文や復号結果の表示がおかしいことがあります。これはプログラムの誤りではなく、計算がまだ完了していないだけです。もう一度リセットボタンを押すと正しく表示されます。つまり、Simple 法では処理に時間がかかることを理解しておけば大丈夫です。

```
always @(posedge clk or negedge rstn) begin

    if ( !rstn ) begin
    isMult <= 1'b0;
    endflag <= 1'b0;
    out <= 17'd0;
    mult_tmp <= 34'd0;
  end
      else if ( !isMult ) begin // multiplication
        mult_tmp <= a * b;
        isMult <= 1'b1;
  end
      else if ( isMult ) begin // Remainder calculation
      if ( mult_tmp < N ) begin
            out <= mult_tmp[33:0];
            endflag <= 1'b1;
      end
                  else begin
            mult_tmp <= mult_tmp - N;
      end
end
end
```