```
module BOOTH (ldA, ldQ, ldM, clrA, clrQ, clrff, sftA, sftQ,
                              addsub, decr, ldcnt, data_in, clk, qm1, eqz);
  input ldA, ldQ, ldM, clrA, clrQ, clrff, sftA, sftQ, addsub, clk;
  input [15:0] data_in;
  output qm1, eqz;
  wire [15:0] A, M, Q, Z;
  wire [4:0] count;

  assign eqz = ~&count;

  shiftreg AR (A, Z, A[15], clk, ldA, clrA, sftA);
  shiftreg QR (Q, data_in, A[0], clk, ldQ, clrQ, sftQ);
  dff QM1 (Q[0], qm1, clk, clrff);
  PIPO MR (data_in, M, clk, ldM);
  ALU AS (Z, A, M, addsub);
  counter CN (count, decr, ldcnt, clk);
endmodule
```

THE DATA PATH


```
module ALU (out, in1, in2, addsub);
  input [15:0] in1, in2;
  input addsub;
  output reg [15:0] out;

  always @(*)
    begin
     if (addsub == 0) out = in1 - in2;
     else out = in1 + in2;
    end
endmodule
```

```
module counter (data_out, decr, ldcnt, clk)
  input decr, clk;
  output [4:0] data_out;

  always @(posedge clk)
    begin
      if (ldcnt) data_out < 5'b10000;
      else if (decr) data_out <= data_out - 1;
    end
endmodule
```

```
module shiftreg (data_out,data_in,
          s_in, clk, ld, clr, sft);
  input s_in, clk, ld, clr, sft;
  input [15:0] data_in;
  output reg [15:0] data_out;

  always @(posedge clk)
    begin
      if (clr) data_out <= 0;
      else if (ld)
             data_out <= data_in;
      else if (sft)
     data_out <= {s_in,data_out[15:1]};
    end
endmodule
```

```
module PIPO (data_out,data_in, clk, load);
  input [15:0] data_in;
  input load, clk;
  output reg [15:0] data_out;

  always @(posedge clk)
    if (load) data_out <= data_in;
endmodule

module dff (d, q, clk, clr);
  input d, clk, clr;
  output reg q;

  always @(posedge clk)
    if (clr) q <= 0;
    else q <= d;
endmodule
```