DO NOT SHARE
SLIDES AND CLASS MATERIALS
ON ONLINE SITES

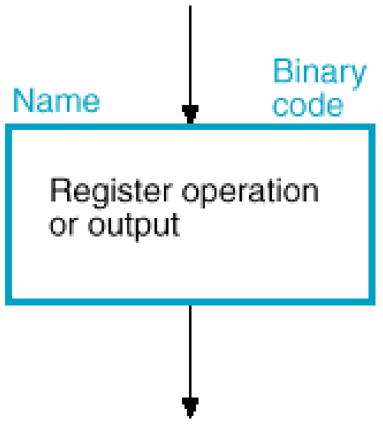
Advanced Logic Design
Algorithmic State Machines (ASM) Method

Mingoo Seok

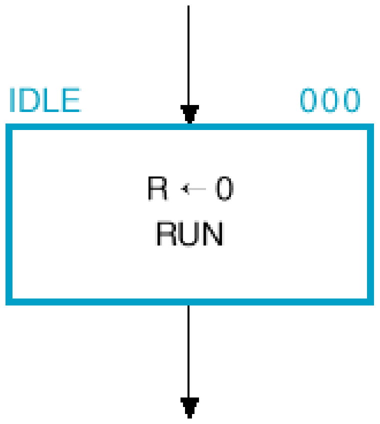
Columbia University

BV: Secs. 6.10-6.13, 7.3-7.4

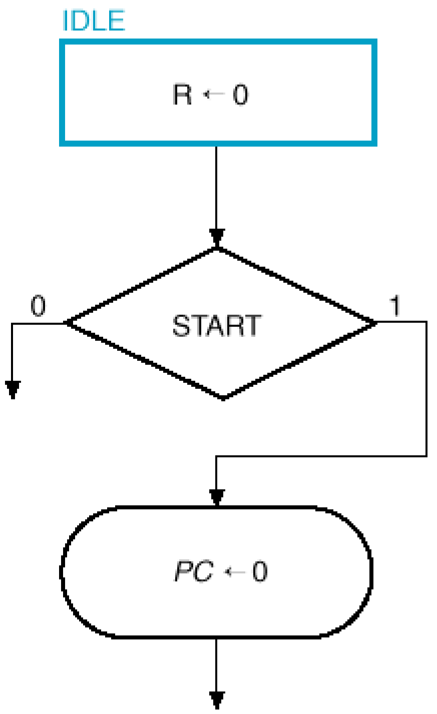
Algorithmic State Machine



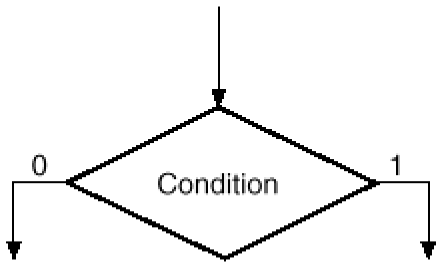
(a) State box



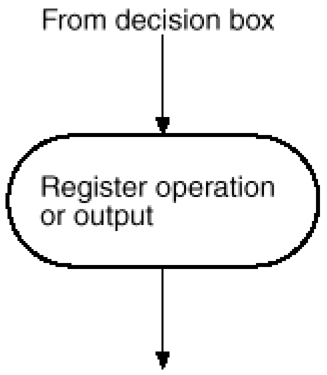
(b) Example of state box



(e) Example of decision and condition output box

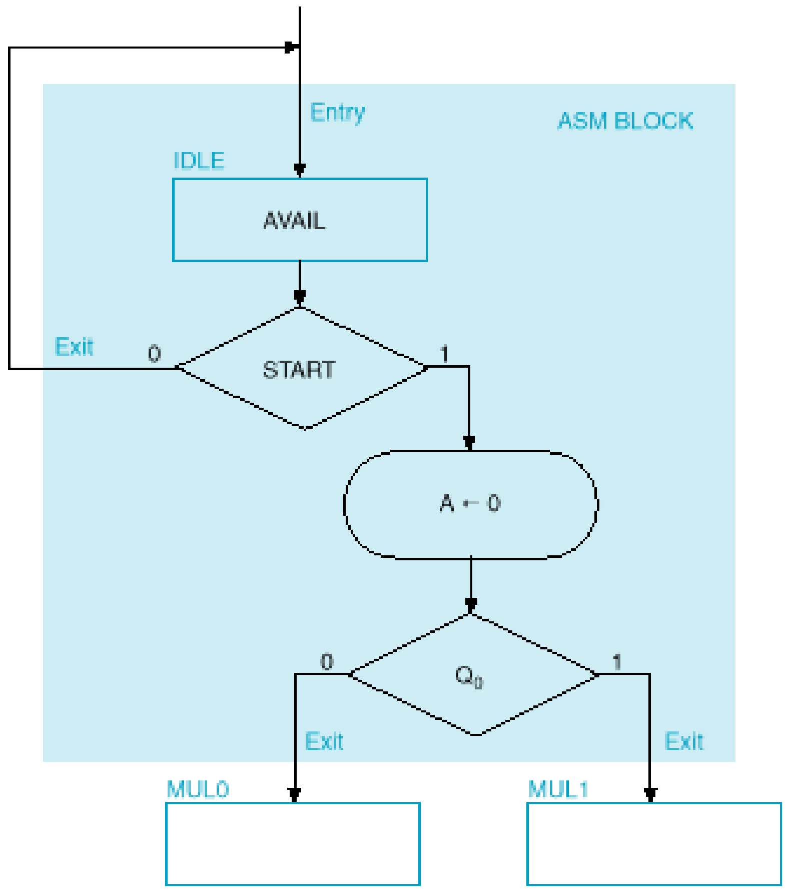


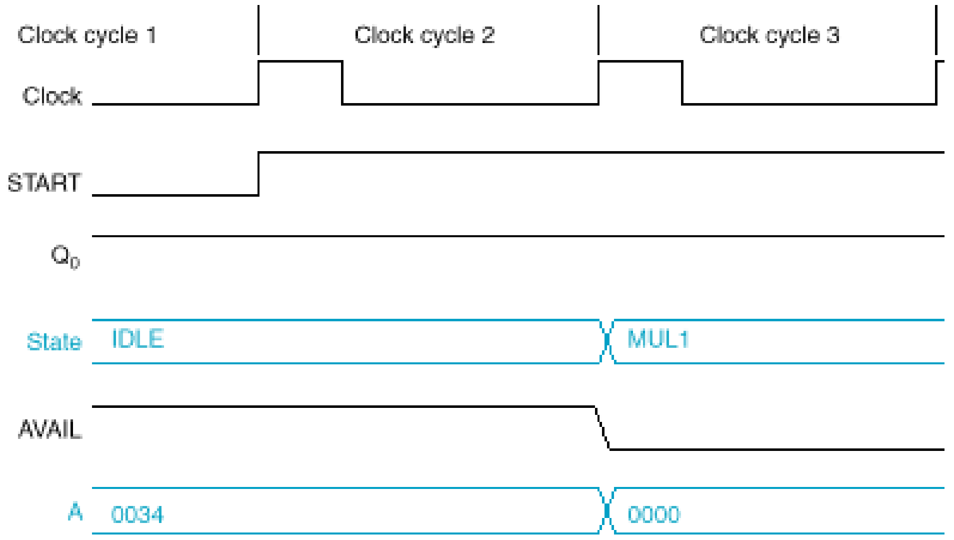
(c) Decision box



(d) Conditional output box

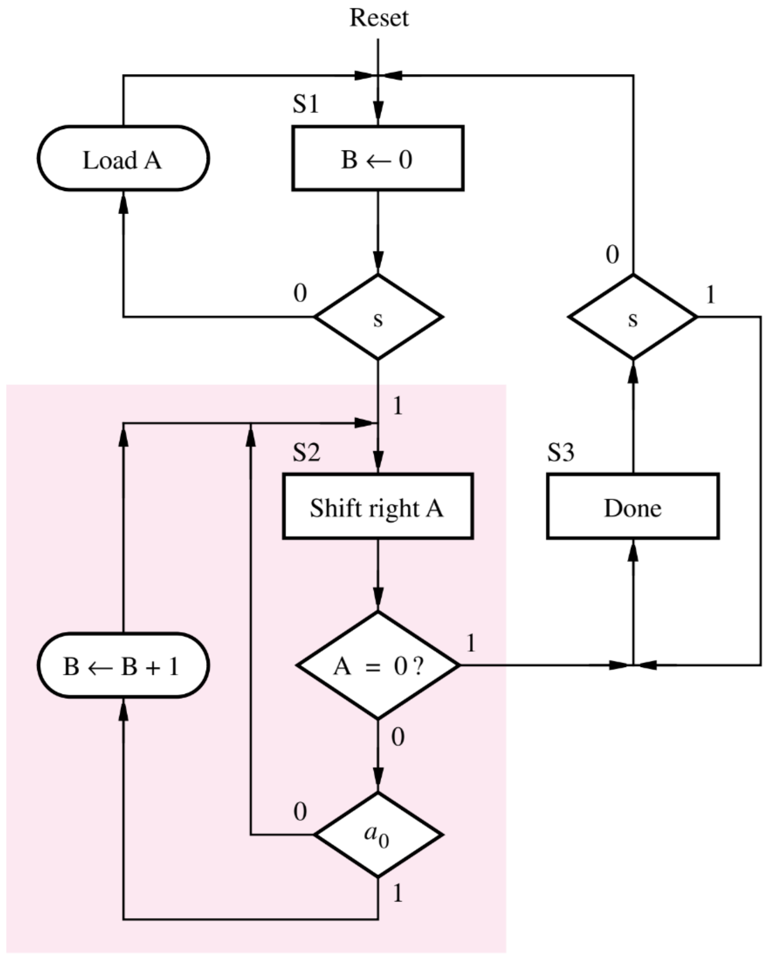
ASM Chart Example



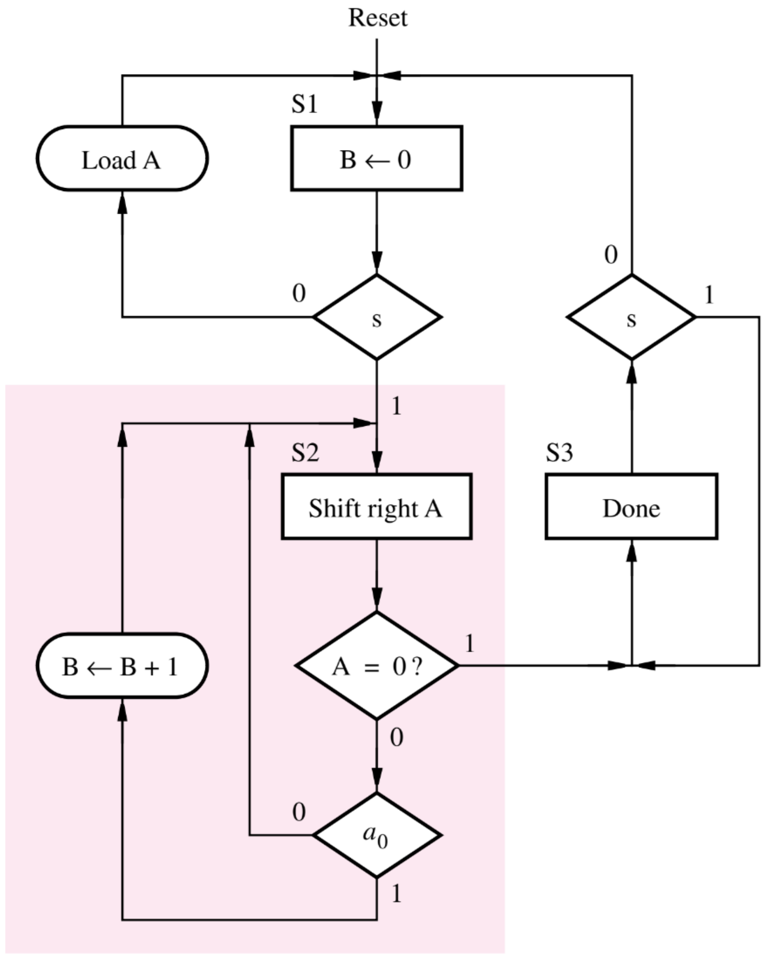


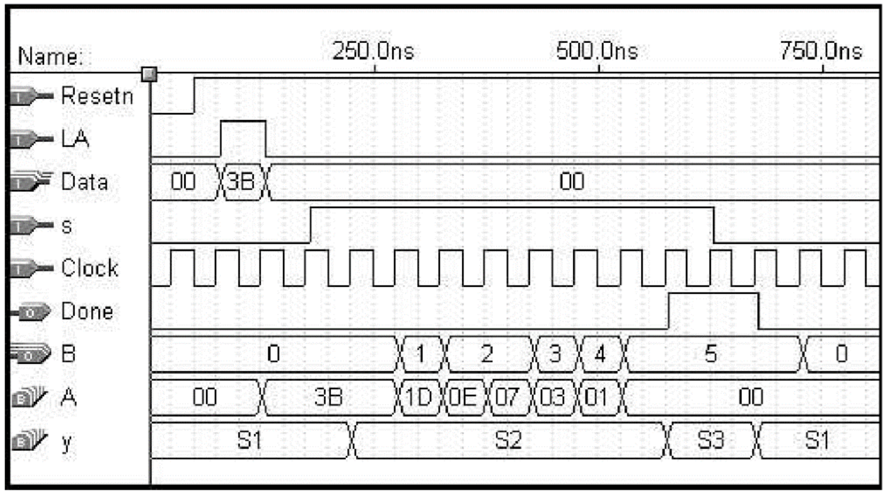
Bit Counting Circuit

```
$B = 0$;
while $A \neq 0$ do
        if $a_0 = 1$ then
                $B = B + 1$;
        end if;
        Right-shift $A$;
end while;
```

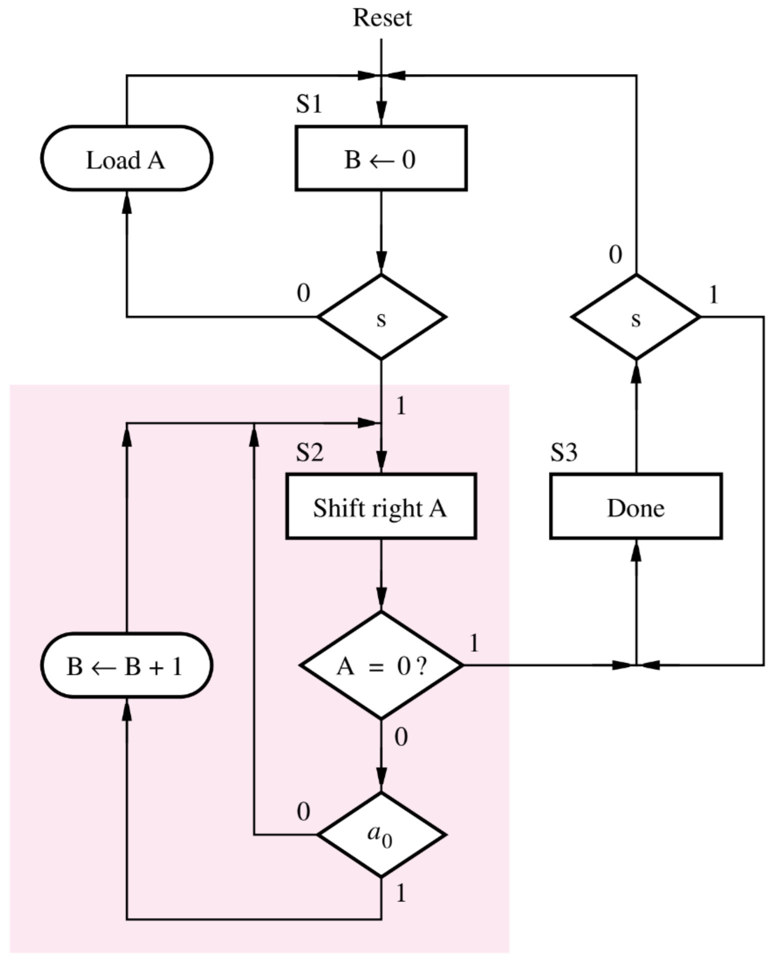


Timing Implied in ASM

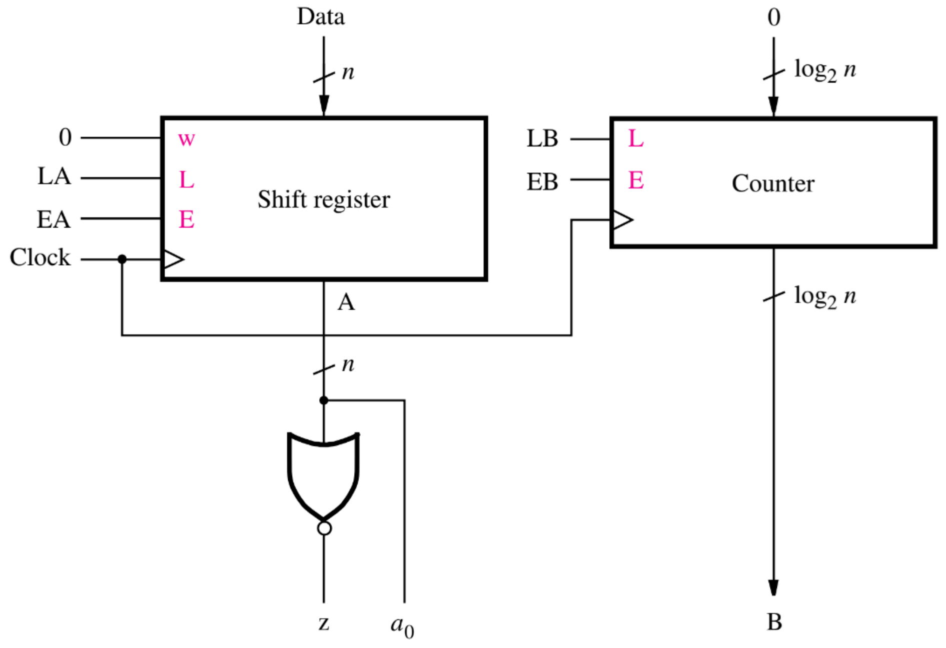




Datapath

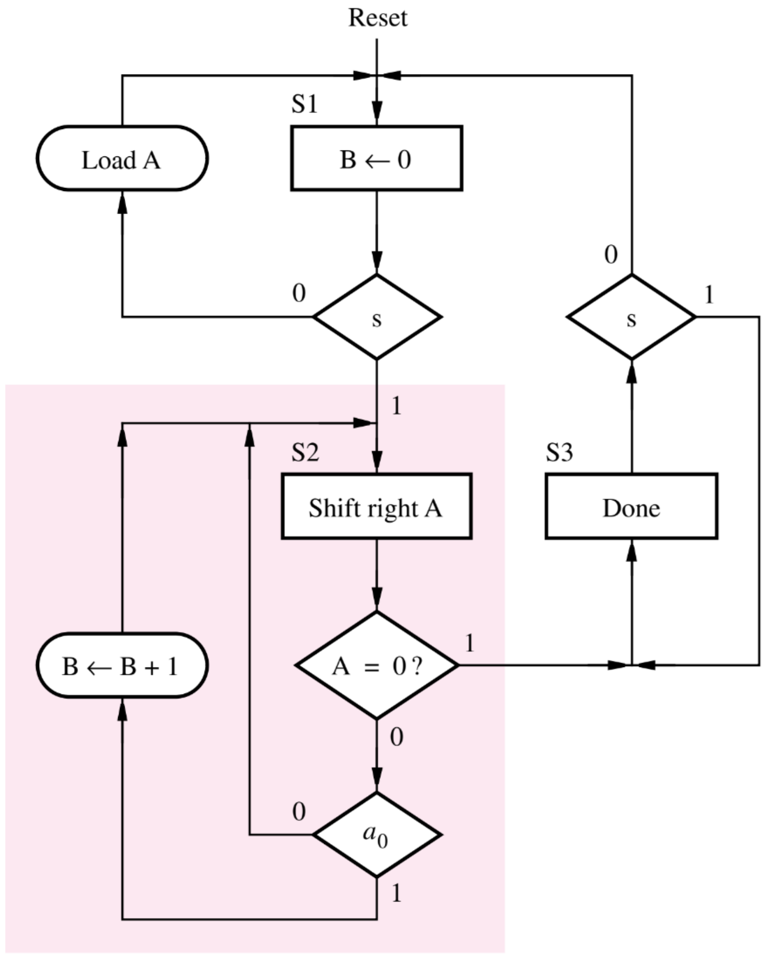


Reset
S1
Load A
B ← 0
0
s
1
0
s
1
S2
Shift right A
S3
Done
B ← B + 1
A = 0 ?
1
0
0
a_0
1

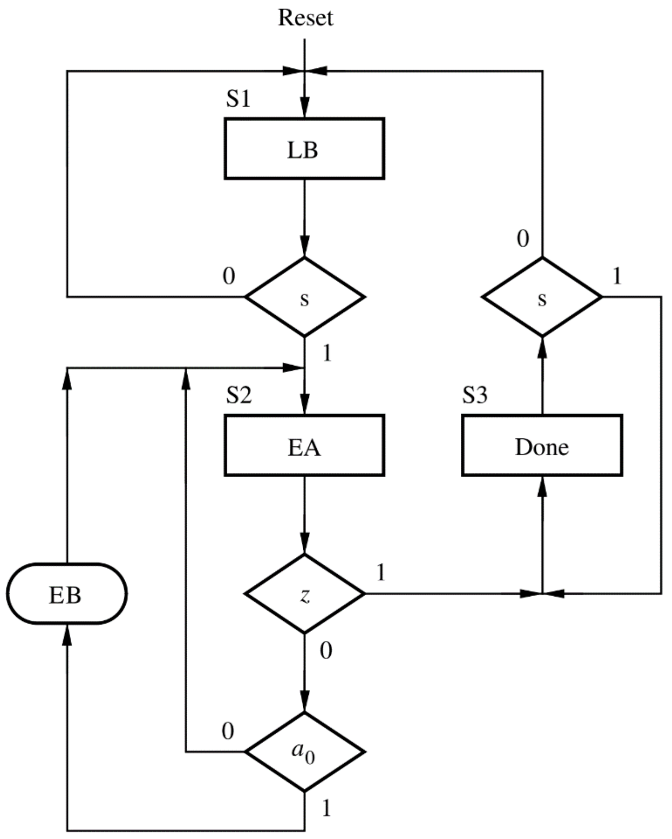


Data
n
0
LA
EA
Clock
w
L
E
Shift register
A
n
z
a_0
0
$\log_2 n$
LB
EB
L
E
Counter
$\log_2 n$
B

Control



Reset
S1
Load A
B ← 0
0
s
1
0
s
1
S2
Shift right A
S3
Done
B ← B + 1
A = 0 ?
1
0
0
a_0
1

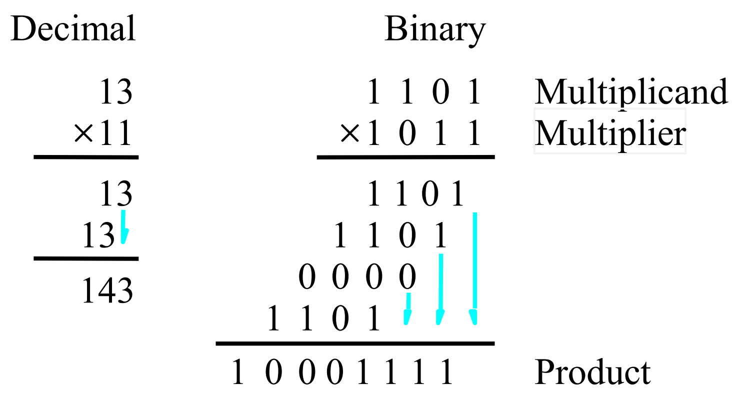


Reset
S1
LB
0
s
1
0
s
1
S2
EA
S3
Done
EB
z
1
0
0
a_0
1

Verilog HDL Code

```
module  bitcount (Clock, Resetn, LA, s, Data, B, Done);
    input  Clock, Resetn, LA, s;
    input  [7:0] Data;
    output  reg  [3:0] B;
    output  reg  Done;
    wire  [7:0] A;
    wire  z;
    reg  [1:0] Y, y;
    reg  EA, EB, LB;

// control circuit

    parameter S1 = 2'b00, S2 = 2'b01, S3 = 2'b10;

    always @(s, y, z)
    begin: State_table
        case (y)
            S1:  if (!s) Y = S1;
                 else Y = S2;
            S2:  if (z == 0) Y = S2;
                 else Y = S3;
            S3:  if (s) Y = S3;
                 else Y = S1;
            default: Y = 2'bxx;
        endcase
    end

    always @(posedge Clock, negedge Resetn)
    begin: State_flipflops
        if (Resetn == 0)
            y <= S1;
        else
            y <= Y;
    end

    always @(y, A[0])
    begin: FSM_outputs
        // defaults
        EA = 0; LB = 0; EB = 0; Done = 0;
        case (y)
            S1:  LB = 1;
            S2:  begin
                     EA = 1;
                     if (A[0]) EB = 1;
                     else EB = 0;
                 end
            S3:  Done = 1;
        endcase
    end

// datapath circuit

    // counter B
    always @(negedge Resetn, posedge Clock)
        if (!Resetn)
            B <= 0;
        else if (LB)
            B <= 0;
        else if (EB)
            B <= B + 1;

    shiftrne  ShiftA (Data, LA, EA, 1'b0, Clock, A);
    assign   z = ~| A;

endmodule
```

Shift-and-Add Multiplier



(a) Manual method

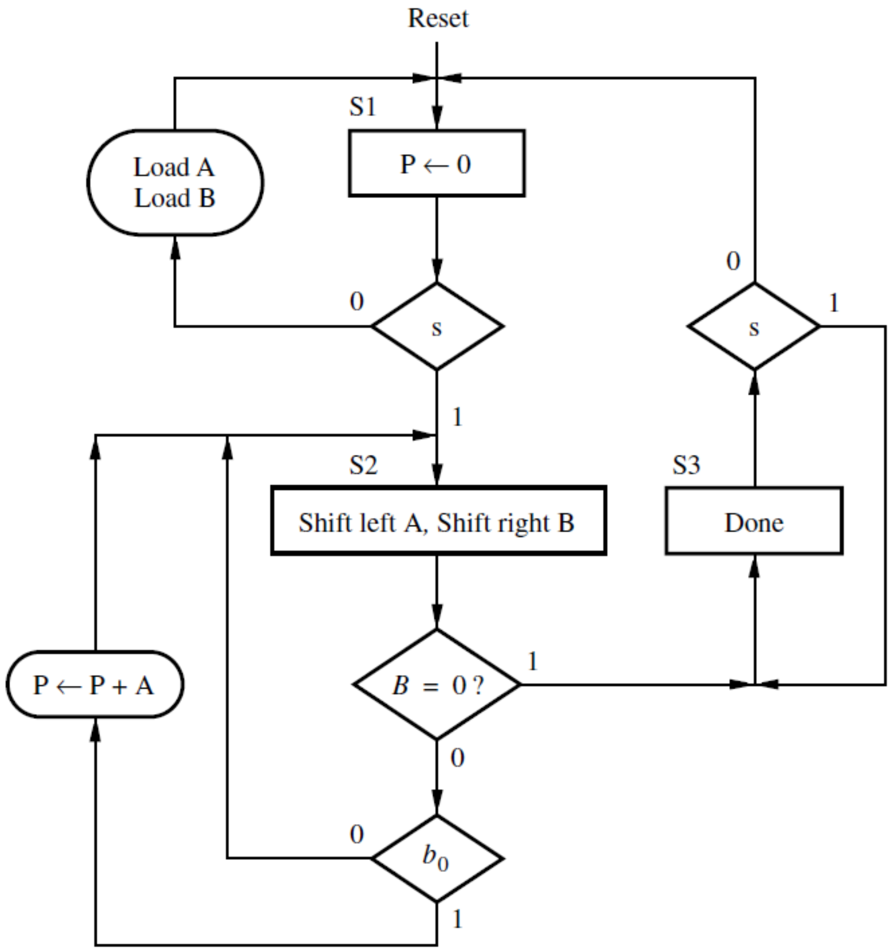
```
$P = 0$ ;
for $i = 0$ to $n - 1$ do
    if $b_i = 1$ then
        $P = P + A$ ;
    end if;
    Left-shift $A$ ;
end for;
```

(b) Pseudo-code

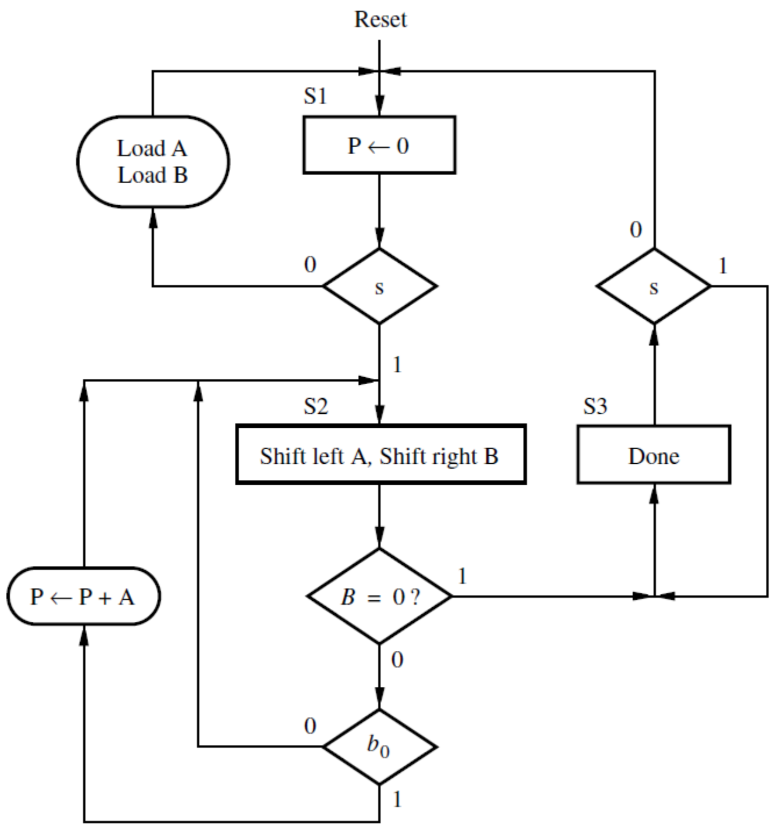
- For large words, the array multiplier can be too hardware intensive
- Implement a *multi-cycle* multiplier with a shifter and an adder

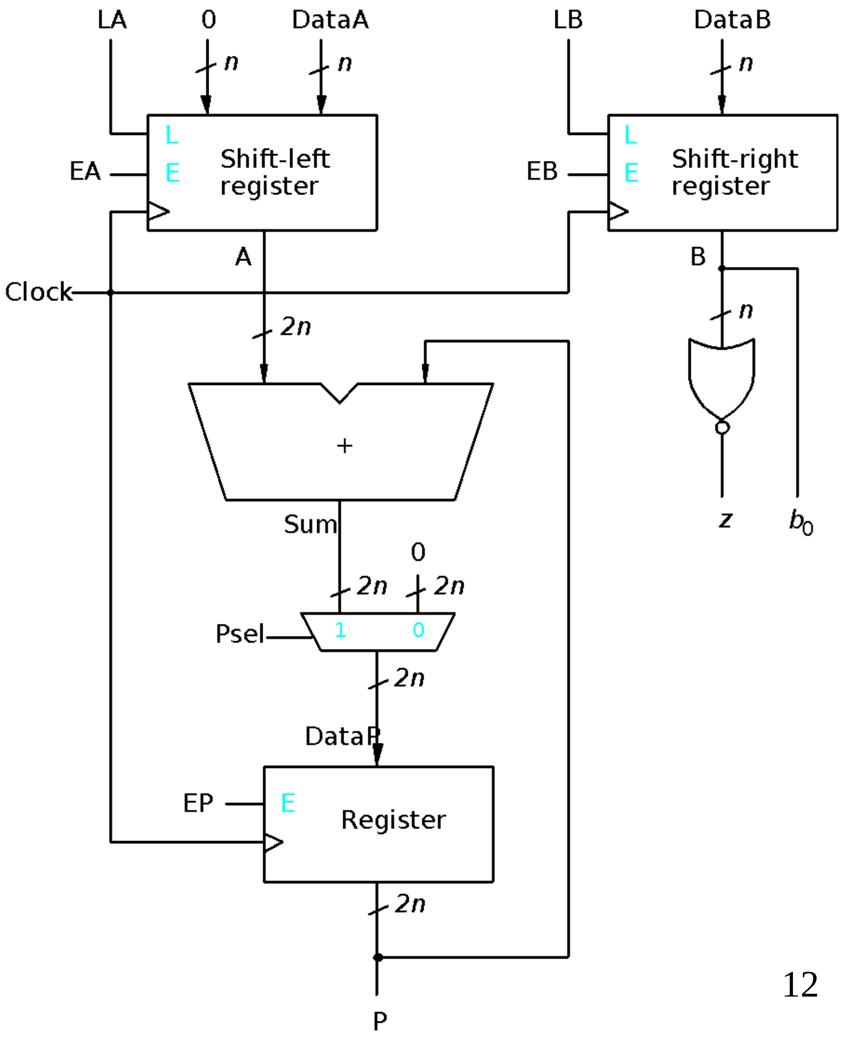
Pseudo Code to ASM Chart

```
$P = 0$ ;
for $i = 0$ to $n - 1$ do
    if $b_i = 1$ then
        $P = P + A$ ;
    end if;
    Left-shift $A$ ;
end for;
```

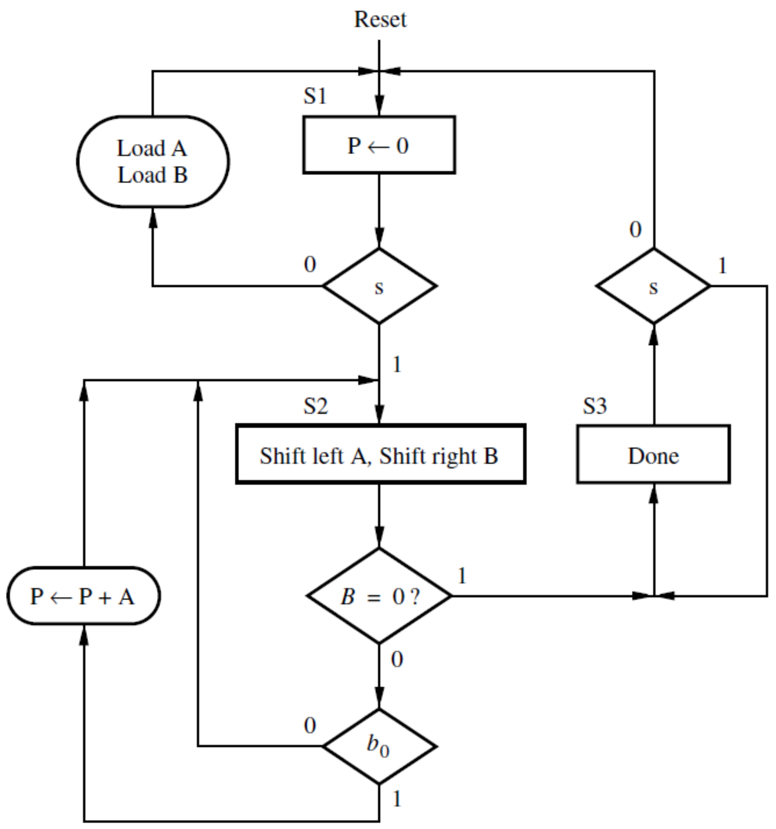


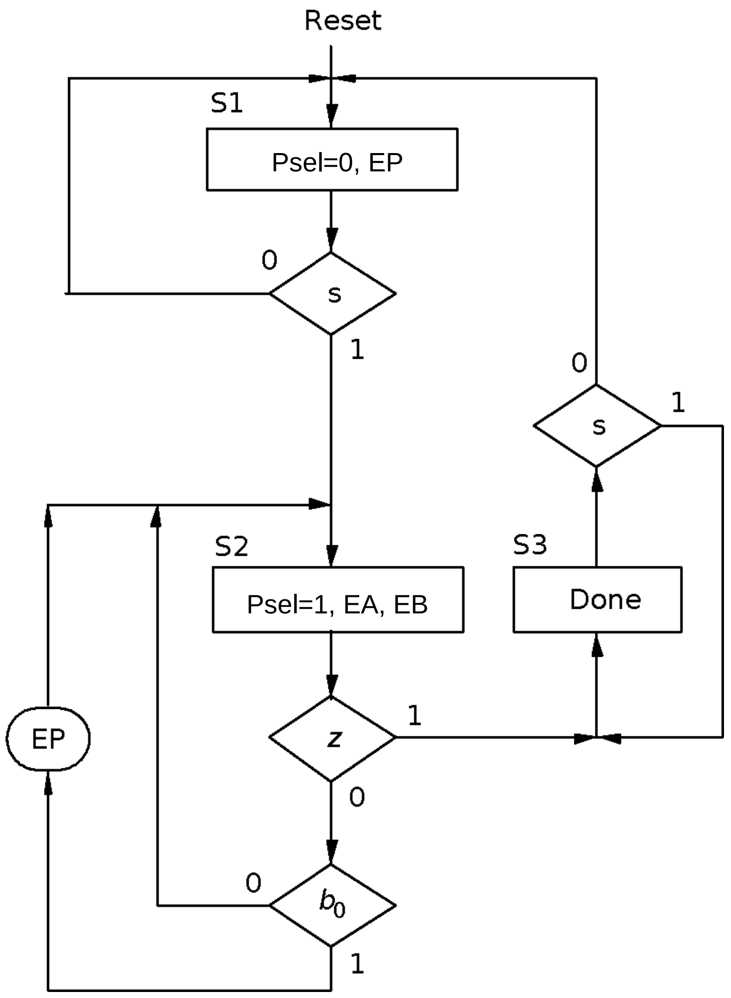
ASM to Datapath





ASM to Control ASM

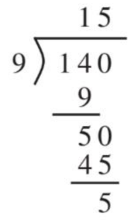




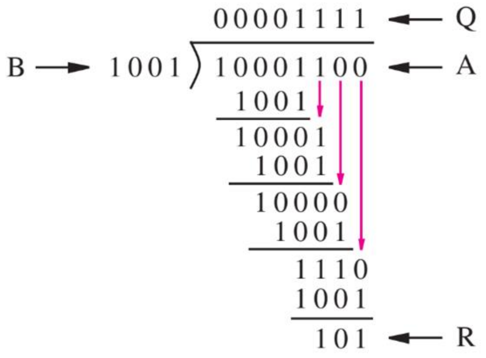
Control ASM to Verilog

```
module multiply (Clock, Resetn, LA, LB, s, DataA, DataB, P, Done);
        parameter n = 8;
        input Clock, Resetn, LA, LB, s;
        input [n-1:0] DataA, DataB;
        output [n+n-1:0] P;
        output reg Done;
        wire z;
        reg [n+n-1:0] A, DataP;
        wire [n+n-1:0] Sum;
        reg [1:0] y, Y;
        reg [n-1:0] B;
        reg EA, EB, EP, Psel;
        integer k;

// control circuit

        parameter S1 = 2'b00, S2 = 2'b01, S3 = 2'b10;

        always @(s, y, z)
        begin: State_table
                case (y)
                S1:             if (s == 0) Y = S1;
                                else Y = S2;
                S2:             if (z == 0) Y = S2;
                                else Y = S3;
                S3:             if (s == 1) Y = S3;
                                else Y = S1;
                                default: Y = 2'bxx;
                endcase
        end

        always @(posedge Clock, negedge Resetn)
        begin: State_flipflops
                if (Resetn = = 0)
                y <= S1;
                else
                y <= Y;
        end

        always @(s, y, B[0])
        begin: FSM_outputs
                // defaults
                EA = 0; EB = 0; EP = 0; Done = 0; Psel = 0;
                case (y)
                S1: EP = 1;
                S2: begin
                        EA = 1; EB = 1; Psel = 1;
                        if (B[0]) EP = 1;
                        else EP = 0;
                  end
                S3: Done = 1;
                endcase
        end

//datapath circuit

        shiftrne ShiftB (DataB, LB, EB, 0, Clock, B);
                defparam ShiftB.n = 8;
        shiftlne ShiftA ({{n{1'b0}}, DataA}, LA, EA, 1'b0, Clock, A);
                defparam ShiftA.n = 16;

        assign z = (B = = 0);
        assign Sum = A + P;

        // define the 2n 2-to-1 multiplexers
        always @(Psel, Sum)
                for (k = 0; k < n+n; k = k+1)
                DataP[k] = Psel ? Sum[k] : 1'b0;
        regne RegP (DataP, Clock, Resetn, EP, P);
                defparam RegP.n = 16;

endmodule
```

Divider Circuits



(a) An example using decimal numbers

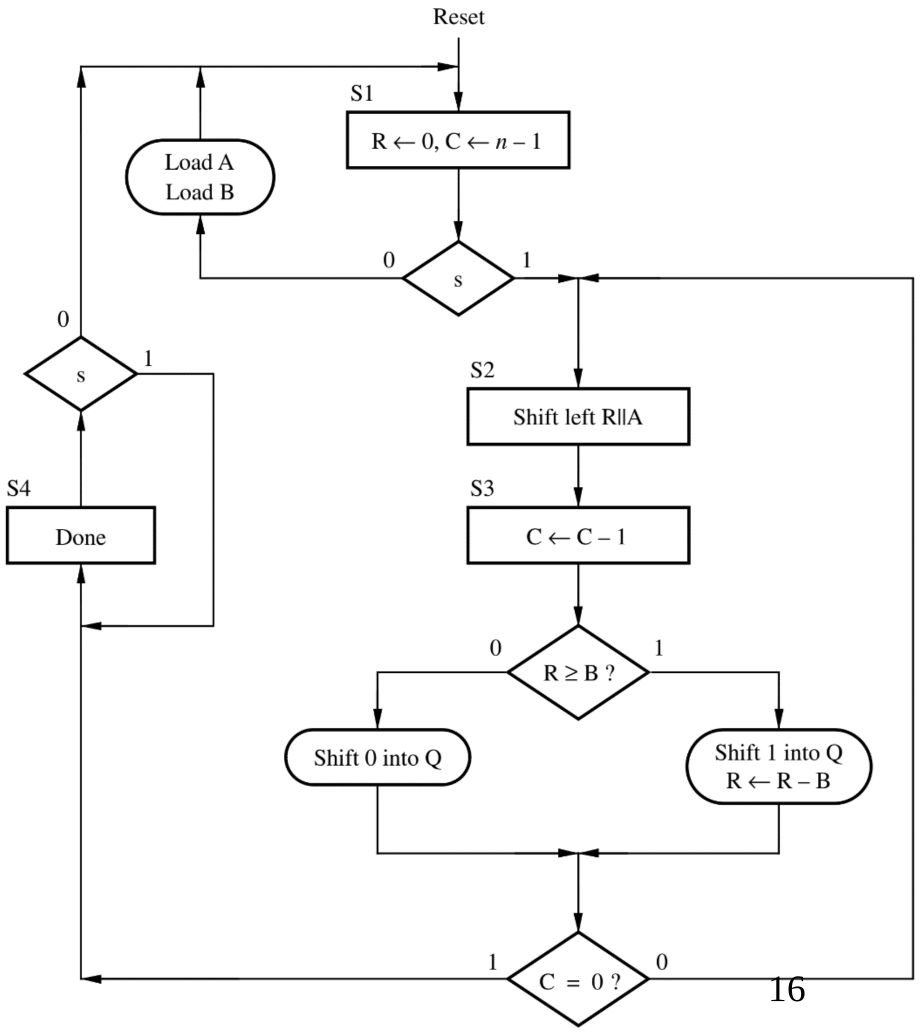


(b) Using binary numbers

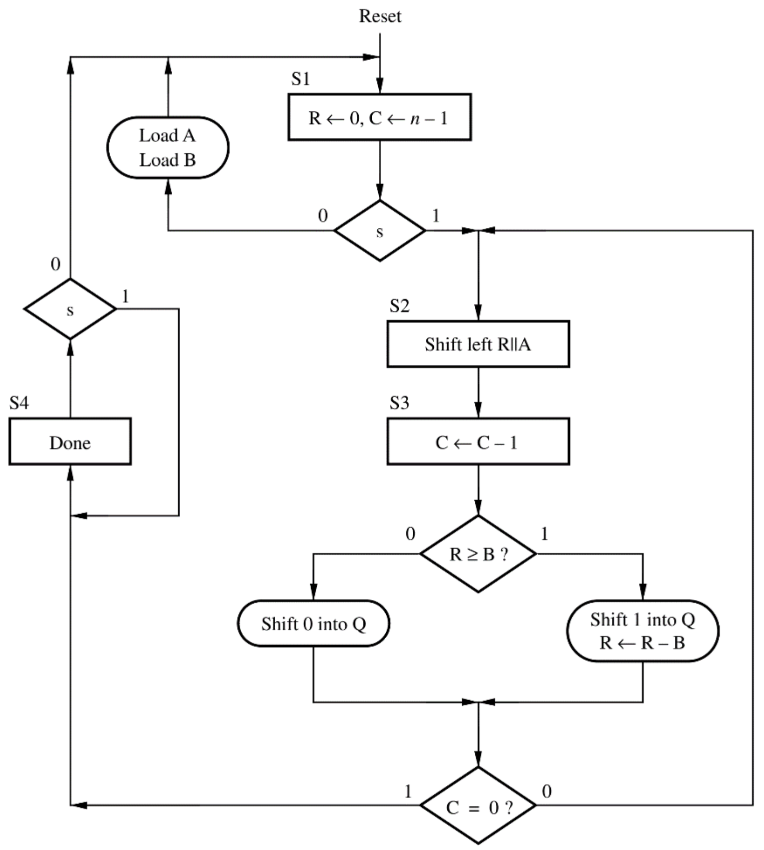
```
$R = 0$;
for $i = 0$ to $n - 1$ do
    Left-shift $R \| A$;
    if R $\geq$ B then
        $q_i = 1$;
        $R = R - B$;
    else
        $q_i = 0$ ;
    end if;
end for;
```

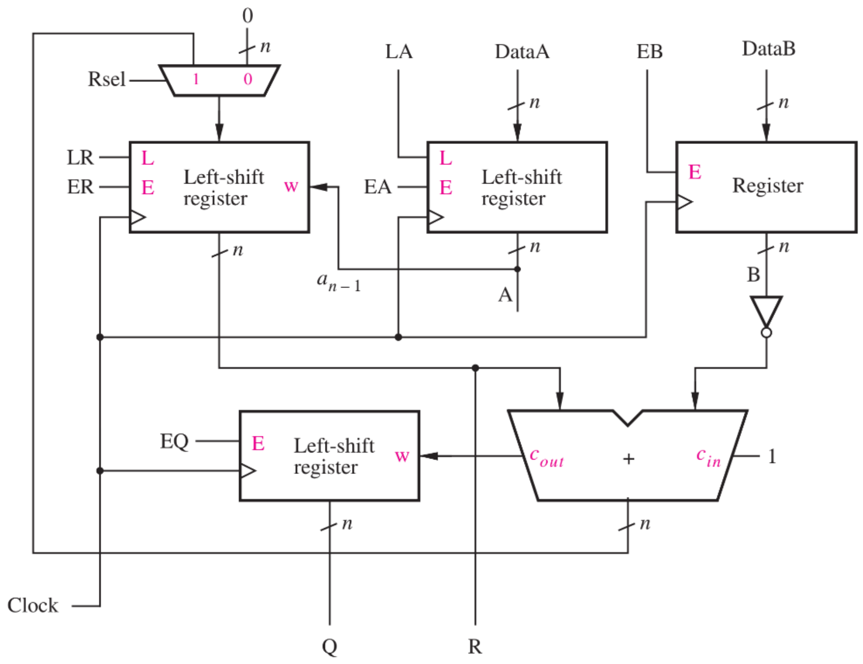
Algorithm to ASM

```
$R = 0$;
for $i = 0$ to $n - 1$ do
        Left-shift $R \| A$;
        if R $\geq$ B then
                $q_i = 1$;
                $R = R - B$;
        else
                $q_i = 0$ ;
        end if;
end for;
```

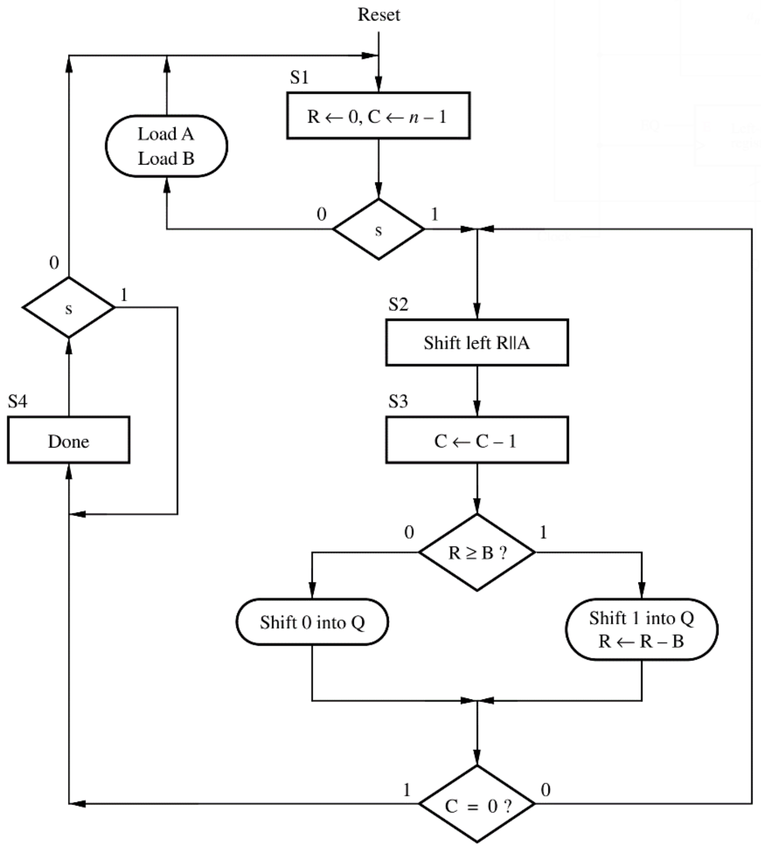


ASM to Datapath Circuit

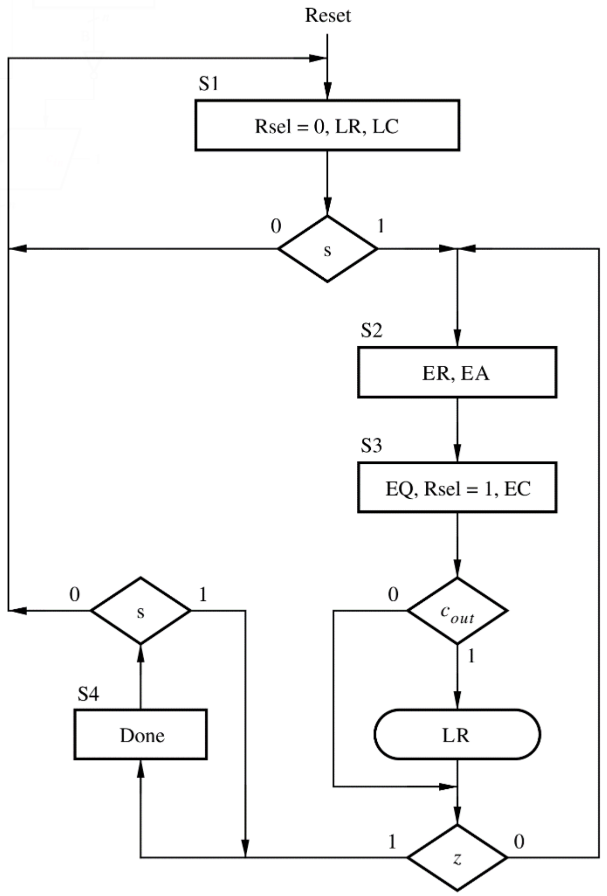




ASM to Control ASM



Reset
S1
R ← 0, C ← n – 1
Load A
Load B
0
s
1
0
s
1
S2
Shift left R||A
S4
Done
S3
C ← C – 1
0
R ≥ B ?
1
Shift 0 into Q
Shift 1 into Q
R ← R – B
1
C = 0 ?
0



Reset
S1
Rsel = 0, LR, LC
0
s
1
S2
ER, EA
S3
EQ, Rsel = 1, EC
0
s
1
0
c_out
1
S4
Done
LR
1
z
0