

CS2022 One Flip-Flop per State

► Alternative Design

- A flip-flop is assigned to each state
- Only one flip-flop may be true
- Each flip-flop represents a state
- The next four slides give:
 - Symbol substitution rules that:
 - Change an ASM chart into:
 - A sequential circuit with one flip-flop per state.

CS2022 State Box Transformation

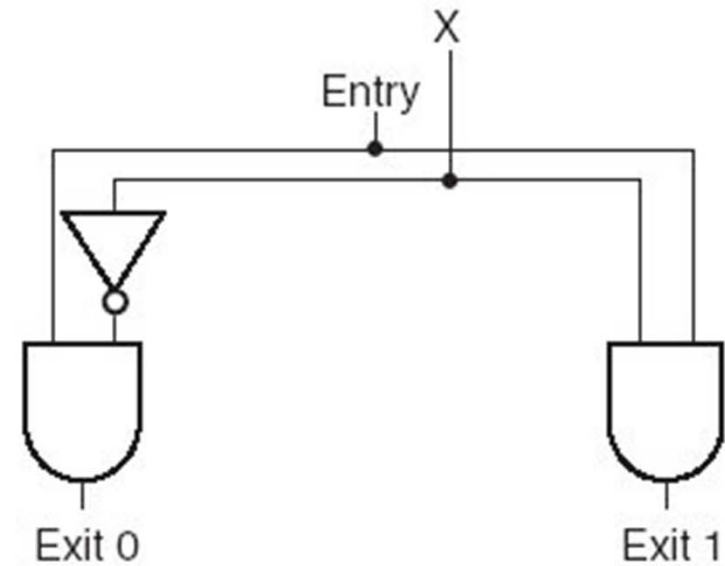
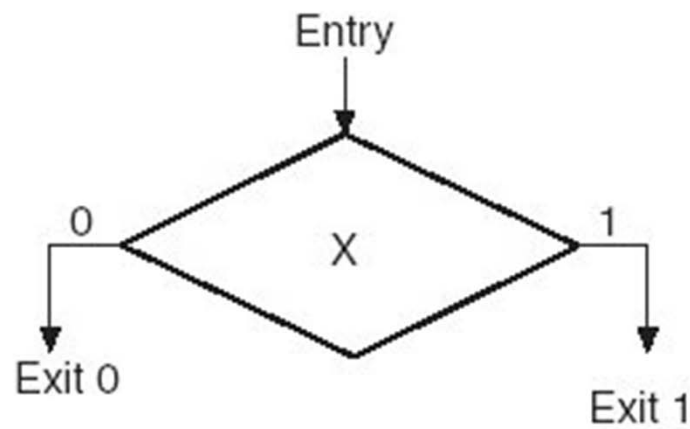
-> D flip-flop

The diagram illustrates the transformation of a State Box into a D flip-flop. The top part shows a State Box with an Entry input, an Exit output, and an unlabeled State input/output. The bottom part shows the equivalent D flip-flop implementation, where the State input is connected to the D input, the Exit output is connected to the Q output, and the unlabeled input is connected to the clock (C) input.

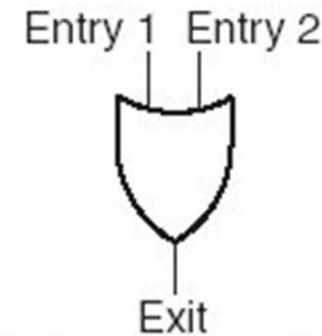
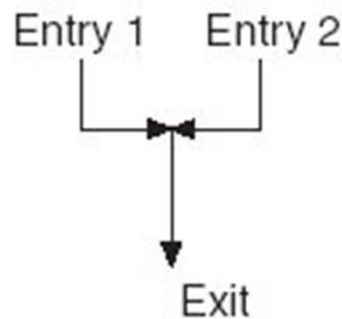


CS2022 Decision Box Transformation

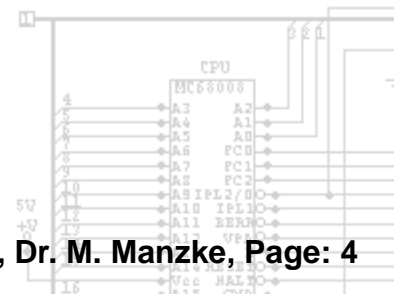
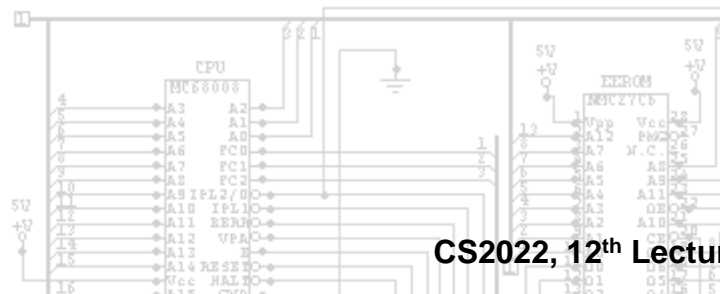
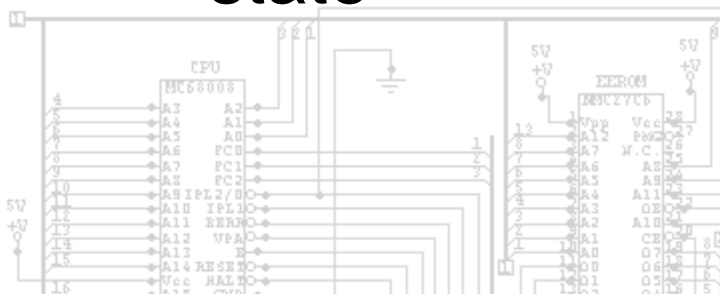
-> Demultiplexer



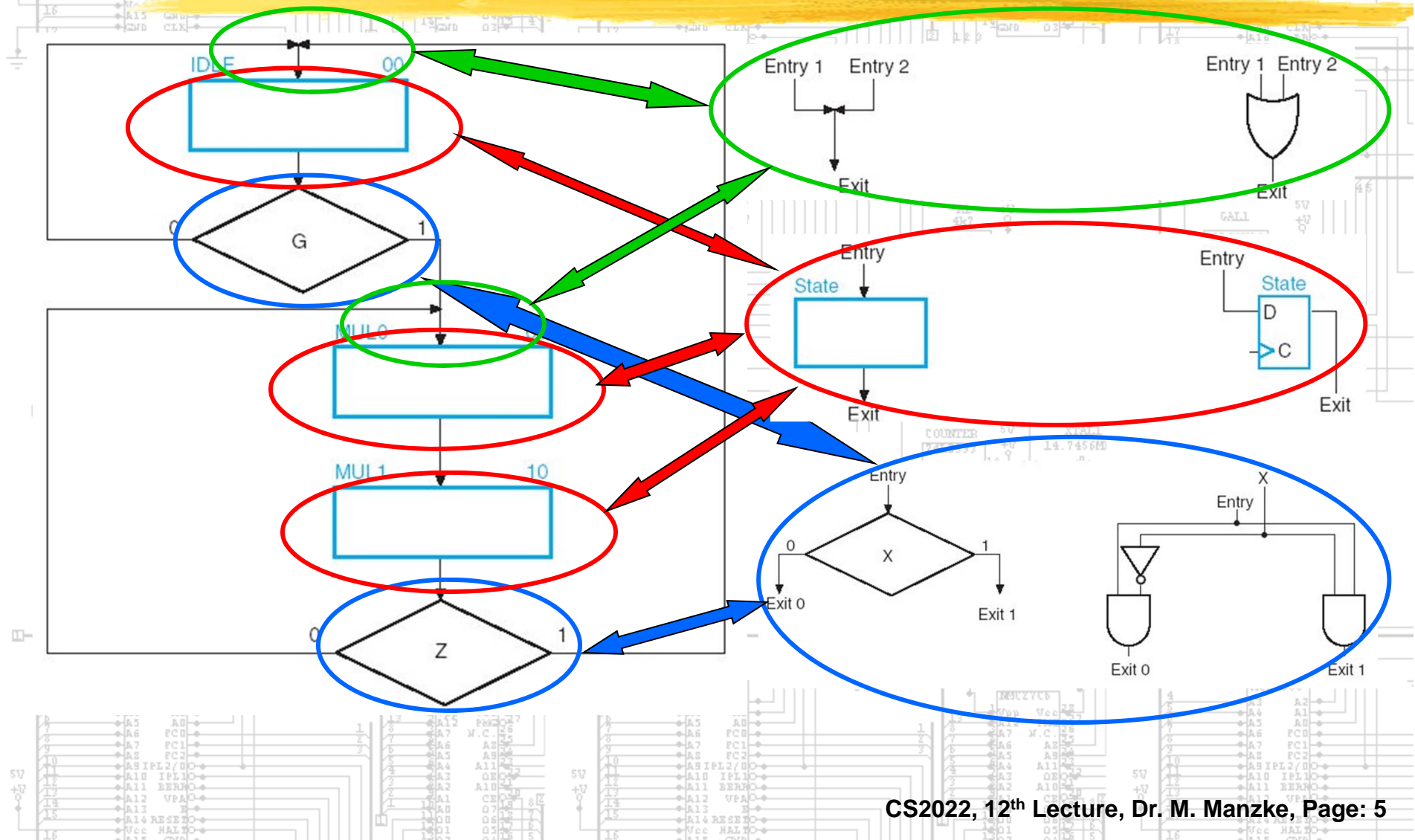
CS2022 Junction Box Transformation -> OR gate



▶ The previous three transformations may be used to transform the sequencing part of a ASM chart into a circuit with one flip-flop per state

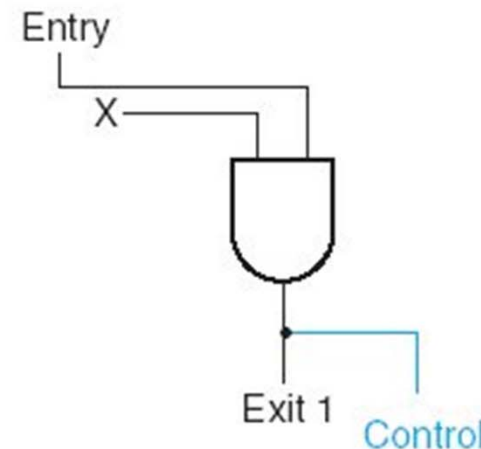
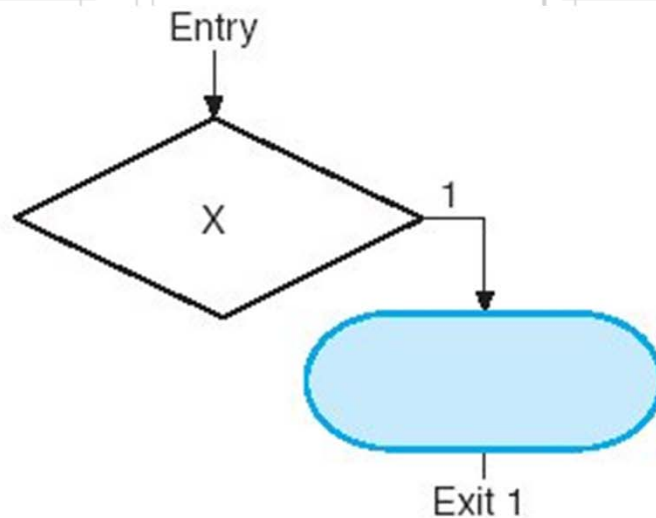


CS2022 Sequencing Part of ASM Chart

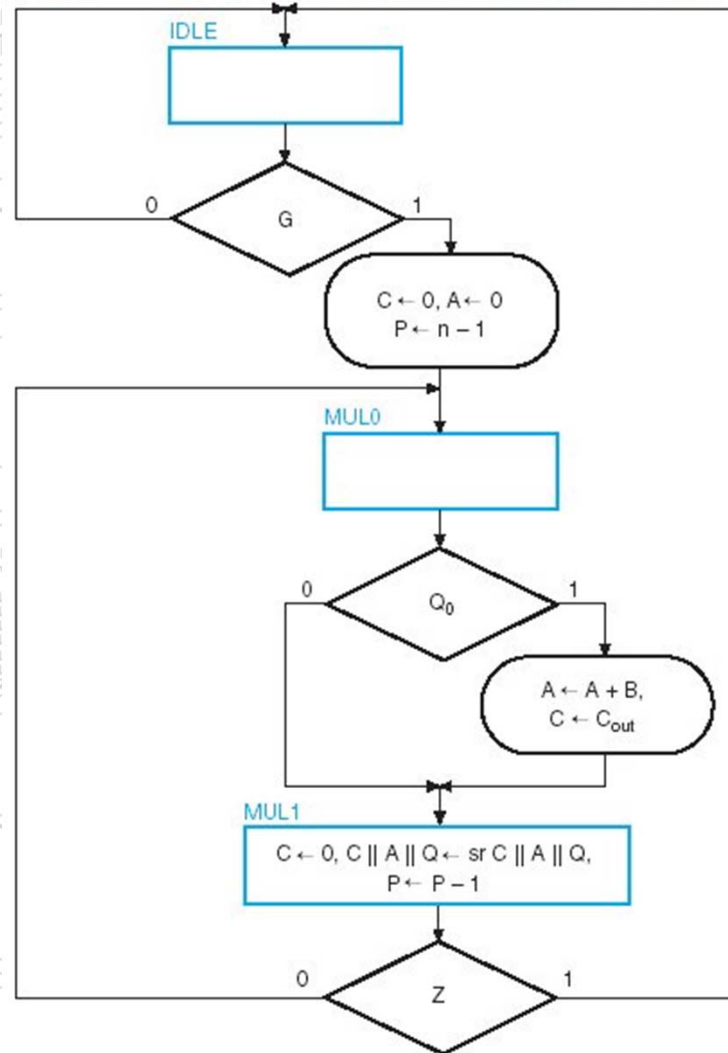


CS2022 Conditional Output Box Transformation

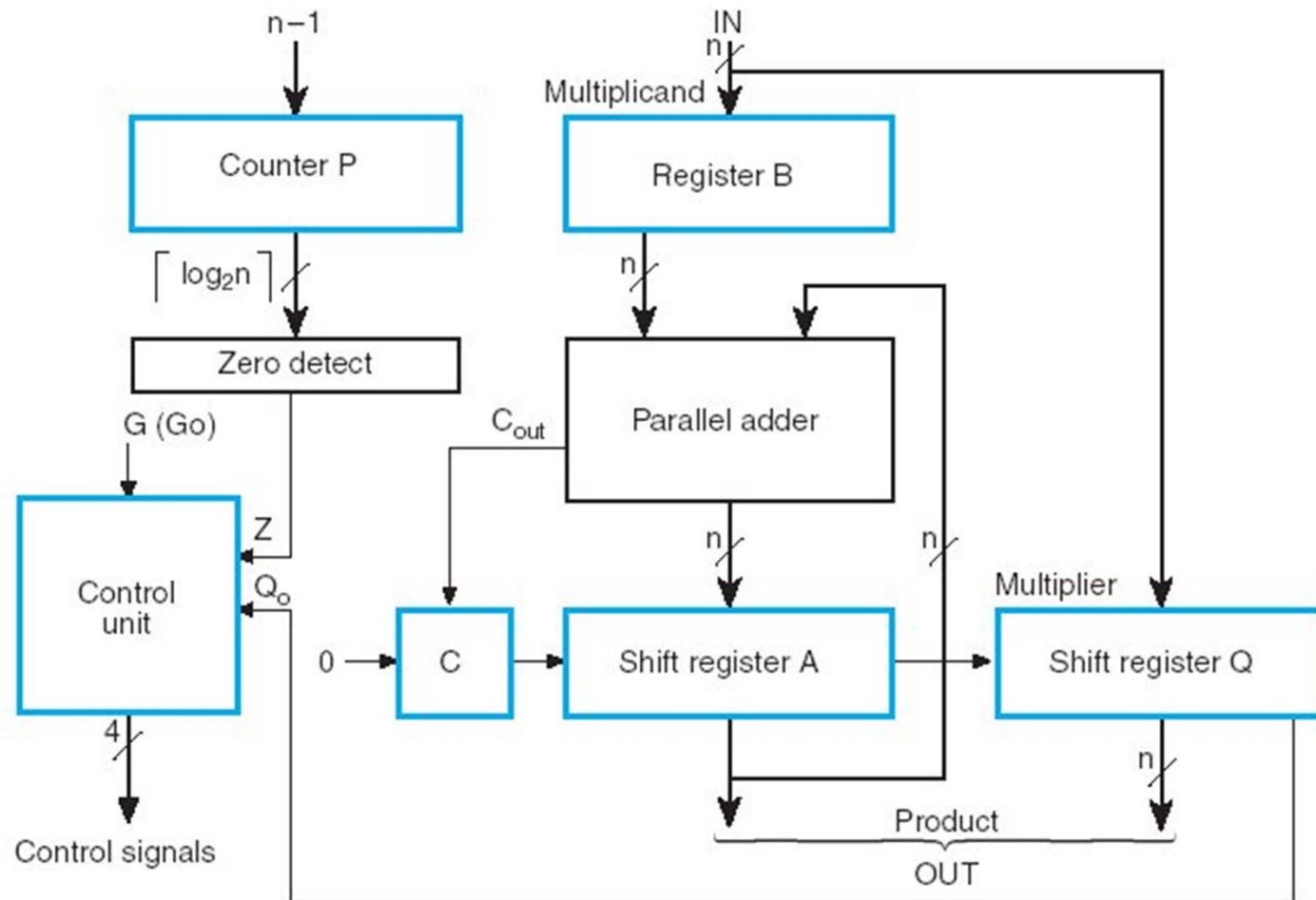
- ▶ Control output is generated by:
 - ▶ Attaching Control line in the right location
 - ▶ Adding output logic
- ▶ The Original ASM is used for the control



CS2022 Binary Multiplier ASM



CS2022 Binary Multiplier Diagram



CS2022 Transformation

► Replace:

1. State boxes with D flip-flops
2. Decision boxes with Demultiplexers
3. Junctions with OR gates
4. Add output signals

► Use table on the following slide

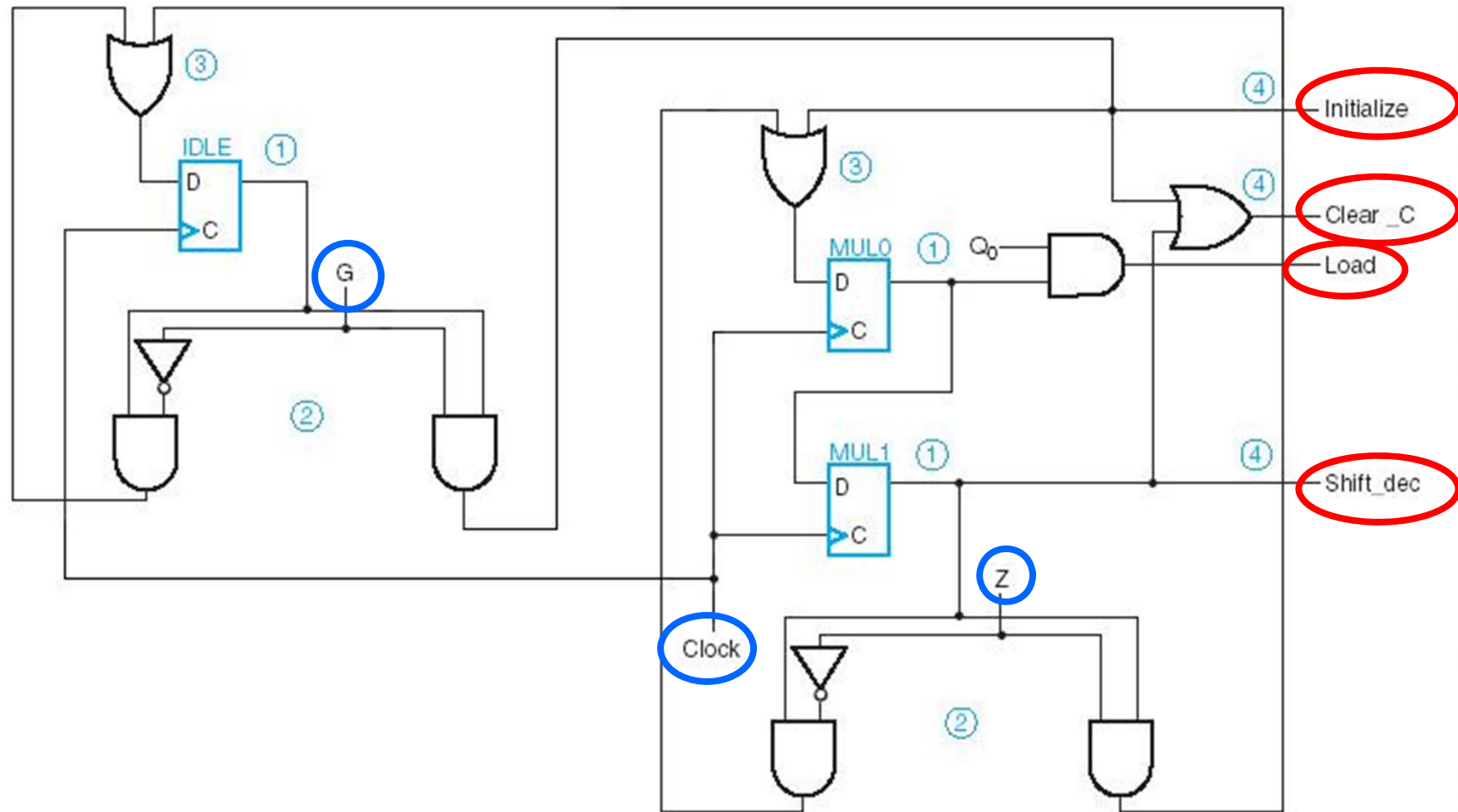
CS2022 Control Signals for Binary Multiplier

Block Diagram Module	Microoperation	Control Signal Name	Control Expression
Register A:	$A \leftarrow 0$ $A \leftarrow A + B$ $C A Q \leftarrow \text{sr } C A Q$	Initialize Load Shift_dec	$\text{IDLE} \cdot G$ $\text{MUL0} \cdot Q_0$ MUL1
Register B:	$B \leftarrow IN$	Load_B	LOADB
Flip-Flop C:	$C \leftarrow 0$ $C \leftarrow C_{\text{out}}$	Clear_C Load	$\text{IDLE} \cdot G + \text{MUL1}$ —
Register Q:	$Q \leftarrow IN$ $C A Q \leftarrow \text{sr } C A Q$	Load_Q Shift_dec	LOADQ —
Counter P:	$P \leftarrow n - 1$ $P \leftarrow P - 1$	Initialize Shift_dec	— —



CS2022 Binary Multiplier Control Unit

One Flip-Flop per State



CS2022 Binary Multiplier (VHDL)

Entity

- Binary Multiplier with $n = 4$: VHDL Description
- See Figures 8-6 and 8-7 for block diagram and ASM Chart
- in Mano and Kime

```
library ieee;  
use ieee.std_logic_1164.all;  
use ieee.std_logic_unsigned.all;  
entity binary_multiplier is  
    port(CLK, RESET, G, LOADB, LOADQ: in std_logic;  
         MULT_IN: in std_logic_vector(3 downto 0);  
         MULT_OUT: out std_logic_vector(7 downto 0));  
end binary_multiplier;
```


CS2022

Binary Multiplier (VHDL) architecture

```
architecture behavior_4 of binary_multiplier is
    type state_type is (IDLE, MUL0, MUL1);
    signal state, next_state : state_type;
    signal A, B, Q: std_logic_vector(3 downto 0);
    signal P: std_logic_vector(1 downto 0);
    signal C, Z: std_logic;
begin
    Z <= P(1) NOR P(0);
    MULT_OUT <= A & Q;
```


CS2022 Binary Multiplier (VHDL)

state_register: process (CLK, RESET)

state_register: process (CLK, RESET)

begin

if (RESET = '1') then

state <= IDLE;

elsif (CLK'event and CLK = '1') then

state <= next_state;

end if;

end process;

CS2022 Binary Multiplier (VHDL)

next_state_func: process (G, Z, state)

```
next_state_func: process (G, Z, state)
begin
    case state is
        when IDLE =>
            if G = '1' then
                next_state <= MUL0;
            else
                next_state <= IDLE;
            end if;
        when MUL0 =>
            next_state <= MUL1;
        when MUL1 =>
            if Z = '1' then
                next_state <= IDLE;
            else
                next_state <= MUL0;
            end if;
        end case;
    end process;
```

CS2022 Binary Multiplier (VHDL)

datapath_func: process (CLK) Part 1

```
datapath_func: process (CLK)
  variable CA: std_logic_vector(4 downto 0);
begin
  if (CLK'event and CLK = '1') then
    if LOADB = '1' then
      B <= MULT_IN;
    end if;
    if LOADQ = '1' then
      Q <= MULT_IN;
    end if;
```

CS2022 Binary Multiplier (VHDL)

datapath_func: process (CLK) Part 2

```
case state is
when IDLE =>
    if G = '1' then
        C <= '0';
        A <= "0000";
        P <= "11";
    end if;
when MUL0 =>
    if Q(0) = '1' then
        CA := ('0' & A) + ('0' & B);
    else
        CA := C & A;
    end if;
    C <= CA(4);
    A <= CA(3 downto 0);
when MUL1 =>
    C <= '0';
    A <= C & A(3 downto 1);
    Q <= A(0) & Q(3 downto 1);
    P <= P - "01";
end case;
end if;
```

end process;

end behavior_4;