# Modeling Digital Systems with Verilog

Prof. Chien-Nan Liu
TEL: 03-4227151 ext:34534
Email: jimmy@ee.ncu.edu.tw



## Composition of Digital Systems

- Most digital systems can be partitioned into two types of modules:
  - Datapath: perform data-processing operations between registers
  - Controller: determine the sequence of those operations

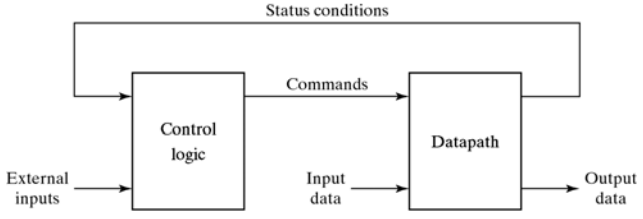
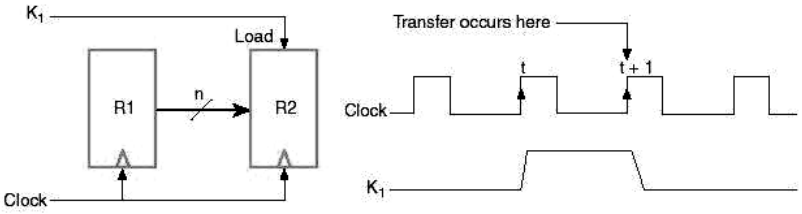



Fig. 8-2 Control and Datapath Interaction

# Register Transfer Operations

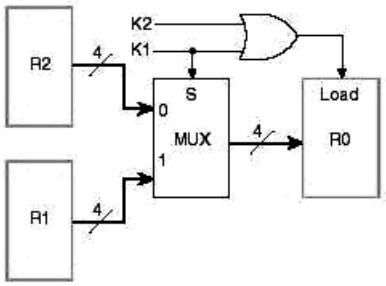
- The movement of the data stored in registers and the processing performed on the data are referred to as ***register transfer operations***
- Ex: If (K1 = 1) then (R2 ← R1)




6-3

# Multiplexer-Based Transfer

- When a register receives data from two or more different sources at different times, a ***multiplexer*** can be used
- Ex: If (K1 = 1) then (R0 ← R1)
  else if (K2 = 1) then (R0 ← R2)




6-4

# Bus-Based Transfer

- Each register has its own multiplexers
  - May be too complex for large systems
- Use shared transfer path instead
  - Often called a ***bus***
  - Can have only one source but multiple destinations at a time
  - More hardware-efficient

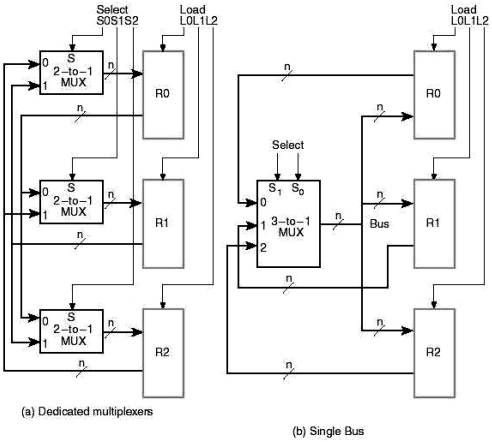
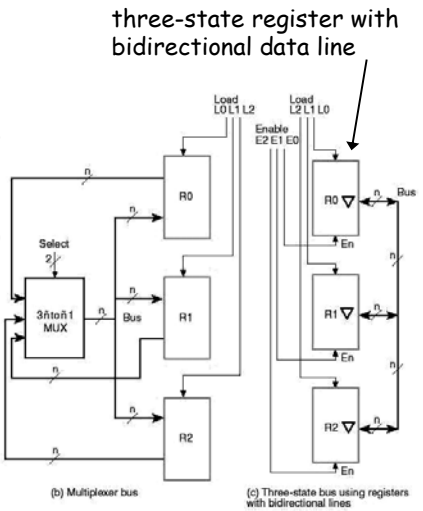



Fig. 7-6 Single Bus versus Dedicated Multiplexers

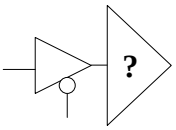


# Three-State Bus

- A bus can be constructed with the three-state buffers
- Many three-state buffer outputs can be connected together
  - Avoid the high-fanin OR in multiplexers
  - Delay time and logic complexity can be reduced
- The signals can travel in two directions on a three-state bus
  - EN=1: output, EN=0: input
  - Simplify the interconnections



# Disadv. of Three-State Bus




- Bus connection problems
  - May reduce reliability
  - One and only one active bus driver at a time
  - Limited driving strength
- Bus floating problems
  - Floating nets with ambiguous logic values
  - Solution: **bus keeper** or pull up/down resistances
- ATPG problem
- FPGA prototyping problem

6-7

# Modeling Three-State Registers

- To declare a bidirectional data port, ***inout*** type is used instead of *input* or *output*
- Ex:

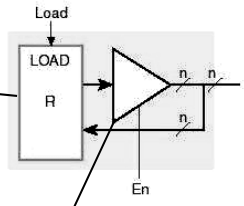
```
module TriReg(CLK, Rst, EN, Load, Data);
  input CLK, Rst, EN, Load;
  inout Data;
  reg int_data, Data;

  always @(posedge CLK) begin
    if (Rst) int_data = 0;
    else if (Load) int_data = Data;
  end

  always @(int_data or EN) begin
    if (EN) Data = int_data;
    else Data = 1`bz;
  end
endmodule
```

separated to ensure correct circuits

register




three-state control

6-8

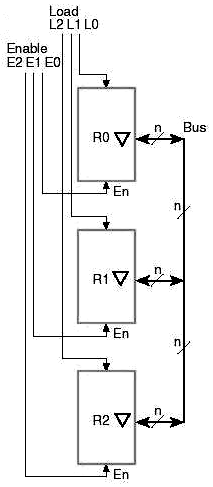
# HDL Modeling for Buses

- To model the behavior of a three-state bus, ***tri*** type is used instead of *wire*
  - tri: has the same properties of *wire* but indicates more than one drivers may connect to it
- Ex:

```
module TriBus(CLK, Rst, E2, E1, E0, L2, L1, L0);
  input CLK, Rst, E2, E1, E0, L2, L1, L0;
  tri databus;

  TriReg R0(CLK, Rst, E0, L0, databus);
  TriReg R1(CLK, Rst, E1, L1, databus);
  TriReg R2(CLK, Rst, E2, L2, databus);
endmodule
```

all connected together

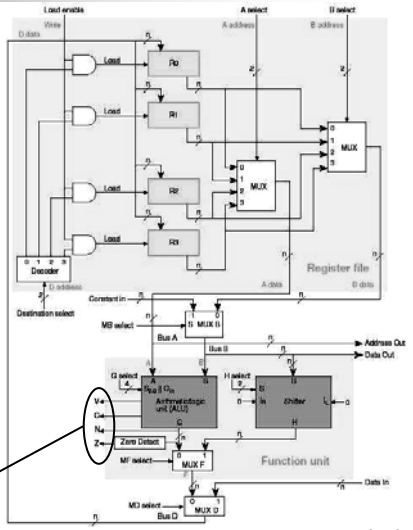





# Datapaths

- A typical datapath often consists of:
  - Register file
  - Arithmetic/logic unit (ALU)
  - Shifter (may be implemented in ALU)
  - Status signals that feedback to controller

status bits



# Arithmetic/Logic Unit

ALU = Arithmetic Circuit + Logic Circuit

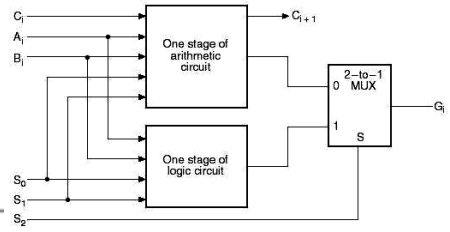



Fig. 7-15 One Stage of ALU

**TABLE 7-8**
**Function Table for ALU**

| Operation Select | | | | | |
|---|---|---|---|---|---|
| $S_2$ | $S_1$ | $S_0$ | $C_{in}$ | Operation | Function |
| 0 | 0 | 0 | 0 | $G = A$ | Transfer $A$ |
| 0 | 0 | 0 | 1 | $G = A + 1$ | Increment $A$ |
| 0 | 0 | 1 | 0 | $G = A + B$ | Addition |
| 0 | 0 | 1 | 1 | $G = A + \underline{B} + 1$ | Add with carry input of 1 |
| 0 | 1 | 0 | 0 | $G = A + \underline{B}$ | $A$ plus 1's complement of $B$ |
| 0 | 1 | 0 | 1 | $G = A + \overline{B} + 1$ | Subtraction |
| 0 | 1 | 1 | 0 | $G = A - 1$ | Decrement $A$ |
| 0 | 1 | 1 | 1 | $G = A$ | Transfer $A$ |
| 1 | 0 | 0 | X | $G = A \wedge B$ | AND |
| 1 | 0 | 1 | X | $G = A \vee B$ | OR |
| 1 | 1 | 0 | X | $G = \underline{A} \oplus B$ | XOR |
| 1 | 1 | 1 | X | $G = A$ | NOT (1's complement) |

6-11

# Arithmetic Circuit

Eight arithmetic functions can be performed by setting the three control signals: S1, S0, Cin.

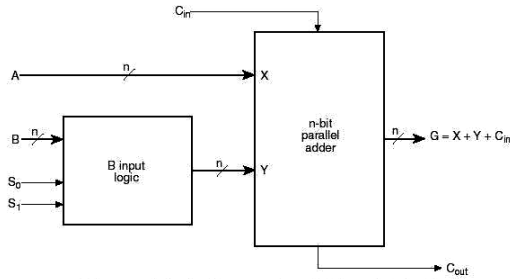


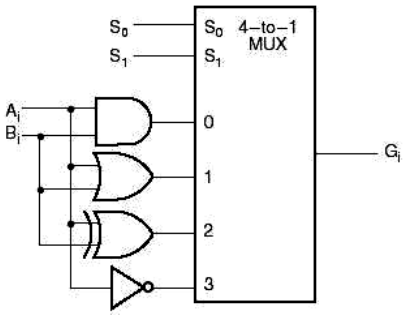

Fig. 7-11 Block Diagram of an Arithmetic Circuit

**TABLE 7-7**
**Function Table for Arithmetic Circuit**

| Select | | Input | $G = A + Y + C_{in}$ | |
|---|---|---|---|---|
| $S_1$ | $S_0$ | Y | $C_{in} = 0$ | $C_{in} = 1$ |
| 0 | 0 | all 0's | $G = A$ (transfer) | $G = A + 1$ (increment) |
| 0 | 1 | $B$ | $G = A + \underline{B}$ (add) | $G = A + B + 1$ |
| 1 | 0 | $\overline{B}$ | $G = A + \overline{B}$ | $G = A + \overline{B} + 1$ (subtract) |
| 1 | 1 | all 1's | $G = A - 1$ (decrement) | $G = A$ (transfer) |

6-12

# Logic Circuit




(a) Logic Diagram

| $S_1$ | $S_0$ | Output | Operation |
|---|---|---|---|
| 0 | 0 | $G = A \wedge B$ | AND |
| 0 | 1 | $G = A \vee B$ | OR |
| 1 | 0 | $G = A \oplus B$ | XOR |
| 1 | 1 | $G = \overline{A}$ | NOT |

(b) Function Table

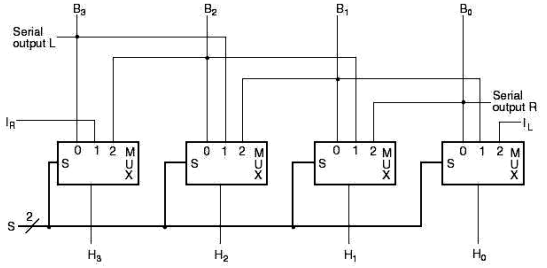
Fig. 7-14 One Stage of Logic Circuit

6-13

# The Shifter

- A bidirectional shift register can meet the basic requirement for the shift operations
  - Shift left and shift right
- One shift operation per clock cycle for shift registers
  - A faster method is often required
- Combinational shifter can be used instead

S=00 : keep unchanged
S=01 : shift right
S=10 : shift left
S=11 : undefined




6-14

# Barrel Shifter

- In some datapaths, the data must be shifted more than one bit position in a single clock cycle
  - Barrel shifter is used
- A barrel shifter with $2^n$ input and output lines requires $2^n$ multiplexers and n selection inputs

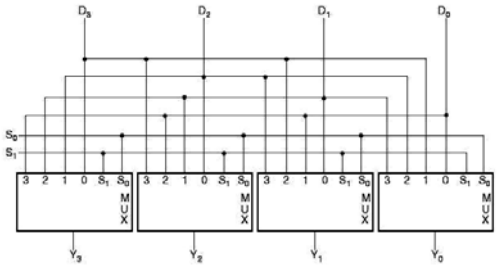



TABLE 7-9
Function Table for 4-Bit Barrel Shifter

| Select | | Output | | | | |
|---|---|---|---|---|---|---|
| $S_1$ | $S_0$ | $Y_3$ | $Y_2$ | $Y_1$ | $Y_0$ | Operation |
| 0 | 0 | $D_3$ | $D_2$ | $D_1$ | $D_0$ | No rotation |
| 0 | 1 | $D_2$ | $D_1$ | $D_0$ | $D_3$ | Rotate one position |
| 1 | 0 | $D_1$ | $D_0$ | $D_3$ | $D_2$ | Rotate two positions |
| 1 | 1 | $D_0$ | $D_3$ | $D_2$ | $D_1$ | Rotate three positions |

6-15

# Modeling the Barrel Shifter

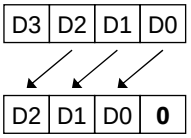
- Describe one by one

```
case (select)
  2`b00:
    Y = D;
  2`b01:
    Y = {D[2:0], D[3]};
  2`b10:
    Y = {D[1:0], D[3:2]};
  2`b10:
    Y = {D[0], D[3:1]};
endcase
```

- Incorrect description

```
case (select)
  2`b00:
    Y = D;
  2`b01:
    Y = D << 1;
  ……
endcase
```

the vacated bits are filled with 0, not really rotate




6-16

## The Control Unit

- Two primary types of control units:
  - Programmable
    - Determine the performed operations according to the pre-stored *instructions* and associated operands
  - Non-programmable
    - Determine the performed operations and their sequence by only its inputs and the status bits

6-17

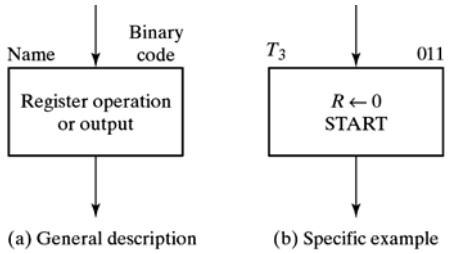
## Design of Control Unit

- Design of control unit: the most challenging and creative part of digital design
  - Formulate hardware algorithms for achieving required objectives
- A special flowchart, *algorithmic state machine* (ASM), is often used to define hardware algorithms
  - Convenient to specify the procedural steps and decision paths with timing relationship
  - Easier to understand
  - Lead directly to hardware realization

6-18

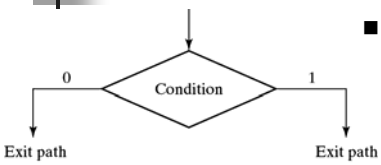
# Basic Elements of ASM Chart (1/2)

- The ASM chart contains three basic elements:
  - State box (rectangle shape)
  - Decision box (diamond shape)
  - Conditional output box (oval shape)
- State box: containing register transfer operations or activated output signals

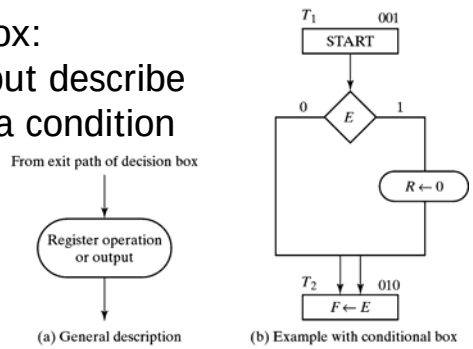



6-19

# Basic Elements of ASM Chart (2/2)




- Decision box: describe the effect of inputs on the control
  - Two exit paths (true & false)
- Conditional output box: similar to state box but describe the operations after a condition is satisfied




6-20

# An Example of ASM Block

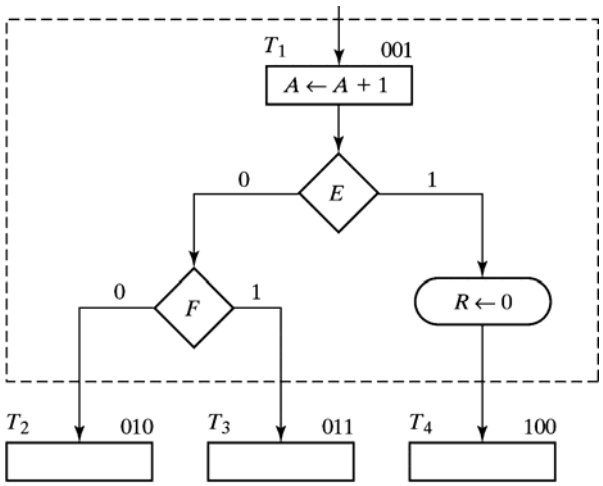



Fig. 8-6 ASM Block

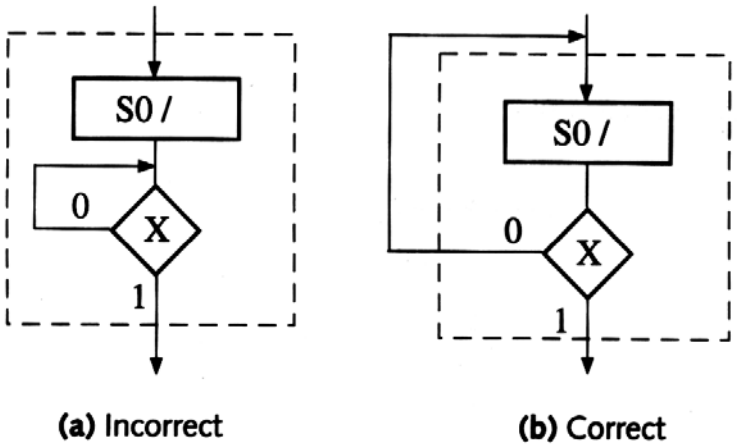
6-21

# ASM Block

- ASM chart is constructed from ASM blocks
- Contains exactly one state box
- One entrance path
- n exit paths
- Every valid input combination defines one exit path
- No internal feedback

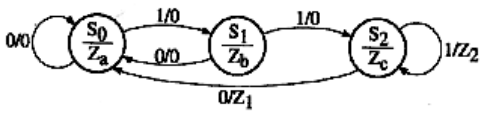
6-22

# ASM Block with Feedback



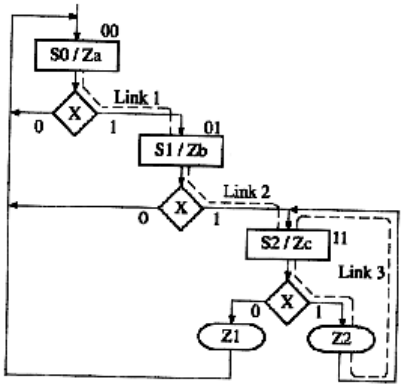

6-23

# STG to ASM Chart




original state transition graph

equivalent ASM chart




6-24

# Timing Considerations

- Assume positive-edge triggering of all flip-flops in Fig. 8-6
- The ASM chart considers the entire block as one unit
  - All operations in the block must occur in the same clock cycle
- The following operations occur simultaneously after T1
  - Register A is incremented
  - If E=1, register R is cleared
  - Control transfer to the next state as specified in Fig. 8-7

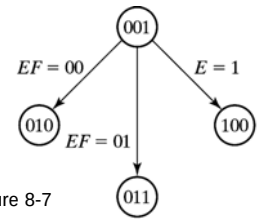



Figure 8-7

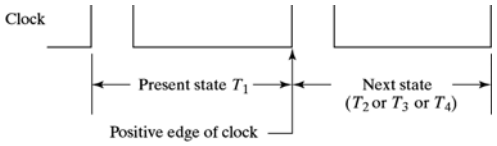



Fig. 8-8 Transition Between States

6-25

# Binary Multiplication

| | | |
|---|---|---|
| 23 | 10111 | Multiplicand |
| 19 | 10011 | Multiplier |
| | 00000 | Initial partial product |
| | 10111 | Add multiplicand, since multiplier bit is 1 |
| | 10111 | Partial product after add and before shift |
| | 010111 | Partial product after shift |
| | 10111 | Add multiplicand, since multiplier bit is 1 |
| | 1000101 | Partial product after add and before shift[a] |
| | 1000101 | Partial product after shift |
| | 01000101 | Partial product after shift |
| | 001000101 | Partial product after shift |
| | 10111 | Add multiplicand, since multiplier bit is 1 |
| | 110110101 | Partial product after add and before shift |
| 437 | 0110110101 | Product after final shift |

a. Note that overflow temporarily occurred.

6-26

# Serial Binary Multiplier

ex: 3 bits
for 8 counts

datapath

controller

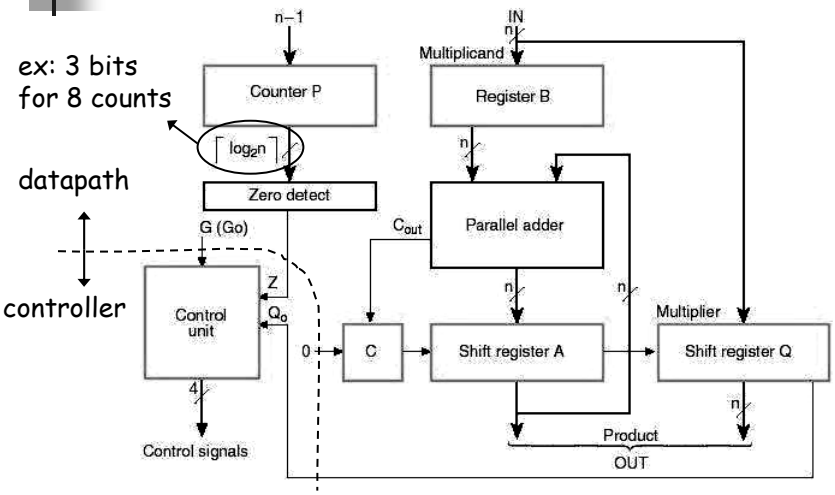
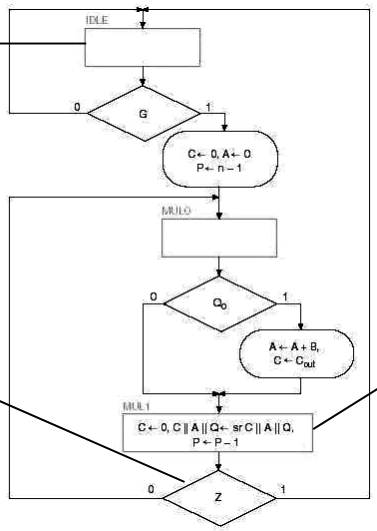



Fig. 8-6 Block Diagram for Binary Multiplier

6-27

# ASM Chart for Multiplier

Multiplicand has been in B and multiplier has been in Q




Z: detect whether the count P is zero

C||A||Q : A composition register made up of other registers

6-28

# ASM Chart to Control Circuits

- Two design approaches are introduced for the control unit:
  - Hardwired control: dedicated circuits for generating the control signals
    - Sequence register and decoder approach
    - One flip-flop per state approach
  - Microprogrammed control: store its binary control values as words in memory
    - Execute the pre-stored *microprogram* according to the current control address

6-29

# Hardwired Control

- Two primary parts:
  - Generate the control signals
  - Determine what happens next
- Analyze the required control signals:

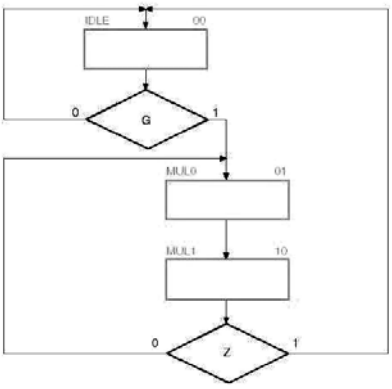
**TABLE 8-1**
**Control Signals for Binary Multiplier**

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

6-30

# The Execution Sequence

- Determine the execution sequence by removing
  - The information of microoperations
  - All conditional output boxes
  - The decision boxes not affecting the next state
- This simplified ASM chart is similar to the traditional state diagram




6-31

# Sequence Register and Decoder

- A sequence register to determine the next states
  - Designed from the simplified ASM chart
- A decoder to generate required control signals for each state

**TABLE 8-2**
**State Table for Sequence Register and Decoder Part of Multiplier Control Unit**

| Present state | | | Inputs | | Next state | | Decoder Outputs | | |
|---|---|---|---|---|---|---|---|---|---|
| Name | $M_1$ | $M_0$ | G | Z | $M_1$ | $M_0$ | IDLE | MUL0 | MUL1 |
| IDLE | 0 | 0 | 0 | × | 0 | 0 | 1 | 0 | 0 |
| | 0 | 0 | 1 | × | 0 | 1 | 1 | 0 | 0 |
| MUL0 | 0 | 1 | × | × | 1 | 0 | 0 | 1 | 0 |
| MUL1 | 1 | 0 | × | 0 | 0 | 1 | 0 | 0 | 1 |
| | 1 | 0 | × | 1 | 0 | 0 | 0 | 0 | 1 |
| — | 1 | 1 | × | × | × | × | × | × | × |

6-32

# The Final Circuits

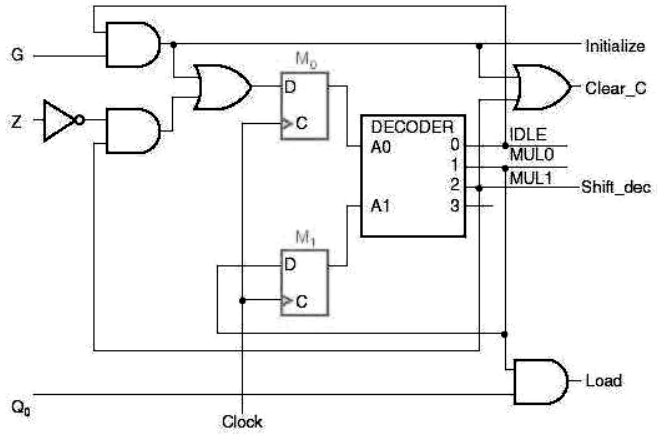



Fig. 8-9 Control Unit for Binary Multiplier Using a Sequence Register and a Decoder

6-33

# Design with Multiplexers

- The sequence register and decoder control consists of three components:
  - Flip-flops: hold the binary value
  - Decoder: generates the control outputs
  - Some gates: determine the next state and the values of output signals
    - Can be replaced by multiplexers !!
- Using multiplexers results in a regular pattern of three levels of components
  - MUXs → flip-flops → decoder

6-34

# ASM Chart for the Example

Inputs = (Next State) & (Input Conditions)

Table 8-7
*Multiplexer Input Conditions*

| Present State $G_1$ | Present State $G_0$ | Next State $G_1$ | Next State $G_0$ | Input Conditions | Inputs MUX1 | Inputs MUX2 |
|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | $w'$ | | |
| 0 | 0 | 0 | 1 | $w$ | 0 | $w$ |
| 0 | 1 | 1 | 0 | $x$ | | |
| 0 | 1 | 1 | 1 | $x'$ | 1 | $x'$ |
| 1 | 0 | 0 | 0 | $y'$ | | |
| 1 | 0 | 1 | 0 | $yz'$ | $yz' + yz = y$ | $yz$ |
| 1 | 0 | 1 | 1 | $yz$ | | |
| 1 | 1 | 0 | 1 | $y'z$ | | |
| 1 | 1 | 1 | 0 | $y$ | | |
| 1 | 1 | 1 | 1 | $y'z'$ | $y + y'z' = y + z'$ | $y'z + y'z' = y'$ |

used as MUX selection

required conditions to enable this transition

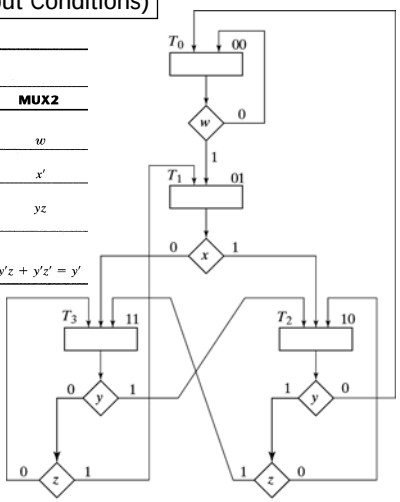



Fig. 8-19 Example of ASM Chart with Four Control Inputs

6-35

# Implementation with MUXs

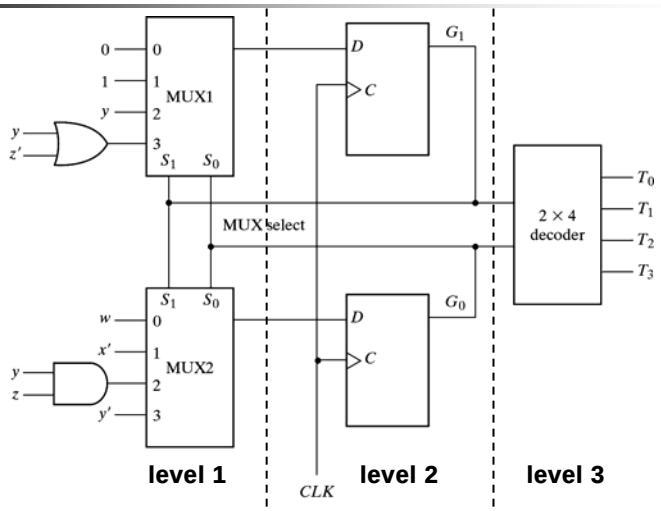



Fig. 8-20 Control Implementation with Multiplexers

6-36

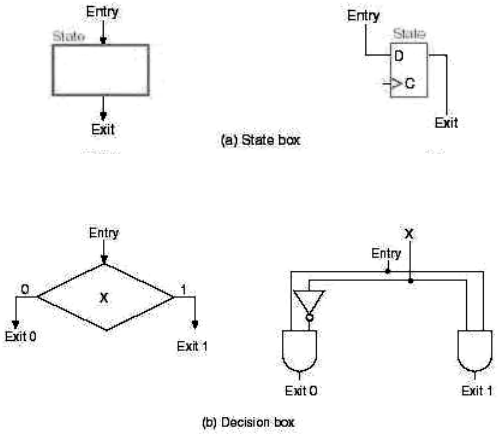
# One Flip-Flop per State

- Each state is assigned a different flip-flop
  - Flip-flop is 1: currently in its corresponding state
- Only one flip-flop contains a 1 at any time
  - The single 1 propagates from one flip-flop to another under the control of decision logic
- Maximum number of flip-flops are used
  - n vs. $\log_2 n$
- Simplify the decision logic and design procedure
  - Can be directly transformed from the ASM chart

6-37

# Transformation Rules (1/2)

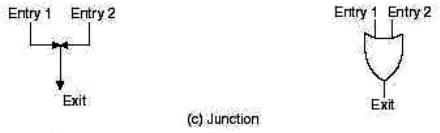
- State box
  - Entry = 1:
    going into the state
  - Exit = 1:
    currently in the state
- Decision box
  - X = 0: send signal to the exit 0
  - X = 1: send signal to the exit 1



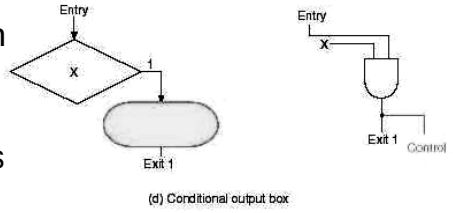

6-38

# Transformation Rules (2/2)

- Junction
  - Wired-OR in nature
  - OR all input signals



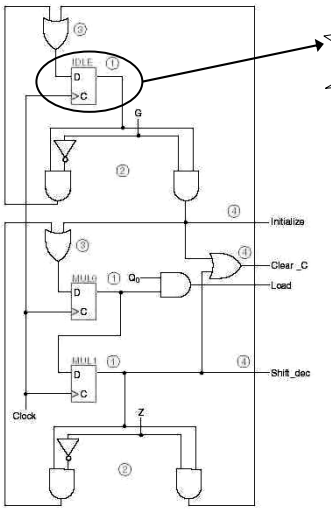

- Conditional output box
  - Replaced by a connection only
  - Attached a control line to trigger the output actions




6-39

# The Final Circuit

① : state box
② : decision box
③ : junction
④ : conditional output




should be 1 at initialization

Solutions:
1. Use flip-flop with PRESET
2. Add inverters at both its input and output

Fig. 8-11 Control Unit with One Flip-Flop per State for the Binary Multiplier

6-40

# Verilog Modeling (1/2)

```
// Binary Multiplier with n = 4: Verilog Description
// See Figures 8-6 and 8-7 for block diagram and ASM Chart

module binary_multiplier_v (CLK, RESET, G, LOADB, LOADQ,
      MULT_IN, MULT_OUT);
input CLK, RESET, G, LOADB, LOADQ;
input [3:0] MULT_IN;
output [7:0] MULT_OUT;
reg [1:0] state, next_state, P;
parameter IDLE = 2'b00, MUL0 = 2'b01, MUL1 = 2'b10;
reg [3:0] A, B, Q;
reg C;
wire Z;

assign Z = ~| P;
assign MULT_OUT = {A,Q};

//state register
always@(posedge CLK or posedge RESET)
begin
  if (RESET == 1)
    state <= IDLE;
  else
  state <= next_state;
end
```

6-41

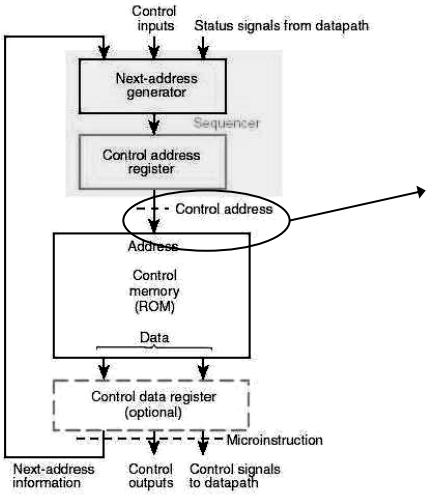
# Verilog Modeling (2/2)

```
//next state function
always@(G or Z or state)
begin
  case (state)
    IDLE:
      if (G == 1)
        next_state <= MUL0;
      else
        next_state <= IDLE;
    MUL0:
      next_state <= MUL1;
    MUL1:
      if (Z == 1)
        next_state <= IDLE;
      else
        next_state <= MUL0;
  endcase
end

//datapath function
always@(posedge CLK)
begin
  if (LOADB == 1)
    B <= MULT_IN;
  if (LOADQ == 1)
    Q <= MULT_IN;
  case (state)
    IDLE:
      if (G == 1)
        begin
          C <= 0;
          A <= 4'b0000;
          P <= 2'b11;
        end
    MUL0:
      if (Q[0] == 1)
        {C, A} = A + B;
    MUL1:
      begin
        C <= 1'b0;
        A <= {C, A[3:1]};
        Q <= {A[0], Q[3:1]};
        P <= P - 2'b01;
      end
  endcase
end
endmodule
```

6-42

# Microprogrammed Control

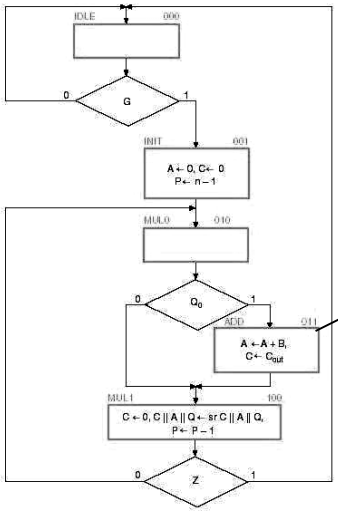



Determine the output actions by only state information (Moore type)

Fig. 8-16 Microprogrammed Control Unit Organization



# Modify the ASM Chart




No conditional output boxes are allowed !! (replaced by a state box)

Fig. 8-17 ASM Chart for Microprogrammed Binary Multiplier Control Unit

# Control Signal Analysis

**TABLE 8-3**
**Control Signals for Microprogrammed Multiplier Control**

| Control Signal | Register Transfers | States in Which Signal is Active | Micro-instruction Bit Position | Symbolic Notation |
|---|---|---|---|---|
| Initialize | $A \leftarrow 0, P \leftarrow n-1$ | INIT | 0 | IT |
| Load | $A \leftarrow A+B, C \leftarrow C_{out}$ | ADD | 1 | LD |
| Clear_C | $C \leftarrow 0$ | INIT, MUL1 | 2 | CC |
| Shift_dec | $C\|A\|Q \leftarrow \text{sr } C\|A\|Q, P \leftarrow P-1$ | MUL1 | 3 | SD |



# Microinstruction Format

total: 12 bits

the four control signals


next address for FALSE

next address for TRUE

5 possible states require 3 bits for an address

**TABLE 8-4**
**SEL Field Definition for Binary Multiplier Control Sequencing**

| SEL | | |
|---|---|---|
| Symbolic notation | Binary Code | Sequencing Microoperations |
| NXT | 00 | $CAR \leftarrow NXTADD0$ |
| DG | 01 | $\overline{G}$: $CAR \leftarrow NXTADD0$<br>$G$: $CAR \leftarrow NXTADD1$ |
| DQ | 10 | $\overline{Q_0}$: $CAR \leftarrow NXTADD0$<br>$Q_0$: $CAR \leftarrow NXTADD1$ |
| DZ | 11 | $\overline{Z}$: $CAR \leftarrow NXTADD0$<br>$Z$: $CAR \leftarrow NXTADD1$ |

# The Final Circuit

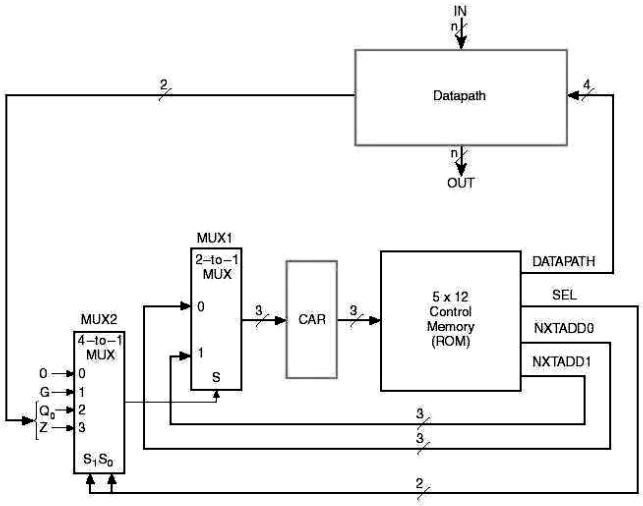



Fig. 8-19 Microprogrammed Control Unit for Multiplier



# Microprogram Design

**TABLE 8-5**
**Register Transfer Description of Binary Multiplier Microprogram**

| Address | Symbolic transfer statement |
|---|---|
| IDLE | $G$: $CAR \leftarrow$ INIT, $\overline{G}$: $CAR \leftarrow$ IDLE |
| INIT | $C \leftarrow 0, A \leftarrow 0, P \leftarrow n-1, CAR \leftarrow$ MUL0 |
| MUL0 | $Q_0$: $CAR \leftarrow$ ADD, $\overline{Q_0}$: $CAR \leftarrow$ MUL1 |
| ADD | $A \leftarrow A+B, C \leftarrow C_{out}, CAR \leftarrow$ MUL1 |
| MUL1 | $C \leftarrow 0, C\|A\|Q \leftarrow$ sr $C\|A\|Q$, $Z$: $CAR \leftarrow$ IDLE, $\overline{Z}$: $CAR \leftarrow$ MUL0, $P \leftarrow P-1$ |

**TABLE 8-6**
**Symbolic Microprogram and Binary Microprogram for Multiplier**

| Address | NXTADD1 | NXTADD0 | SEL | DATAPATH | Address | NXTADD1 | NXTADD0 | SEL | DATAPATH |
|---|---|---|---|---|---|---|---|---|---|
| IDLE | INIT | IDLE | DG | None | 000 | 001 | 000 | 01 | 0000 |
| INIT | — | MUL0 | NXT | IT, CC | 001 | 000 | 010 | 00 | 0101 |
| MUL0 | ADD | MUL1 | DQ | None | 010 | 011 | 100 | 10 | 0000 |
| ADD | — | MUL1 | NXT | LD | 011 | 000 | 100 | 00 | 0010 |
| MUL1 | IDLE | MUL0 | DZ | CC, SD | 100 | 000 | 010 | 11 | 1100 |

# Why Pipelined System ?

- Conventional:
  - Max delay = 12ns
  - Clock rate = 83.3 MHz
  - Required 1 clock cycle to finish a operation
- Pipelined:
  - Max delay = 5ns
  - Clock rate = 200 MHz
  - Required 3 clock cycles (15ns) for a operation
  - Longer latency but higher throughput

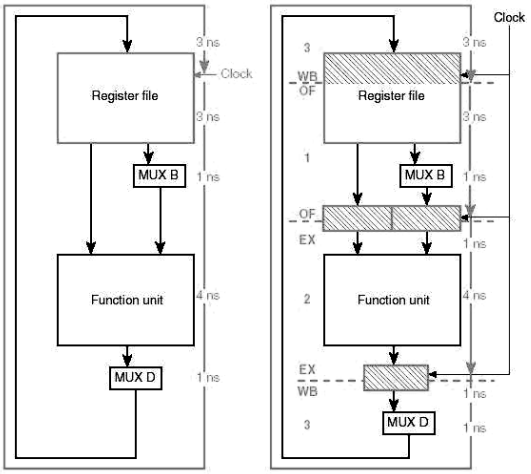



Fig. 7-21 Datapath Timing

# Analogy to Pipelined Operations

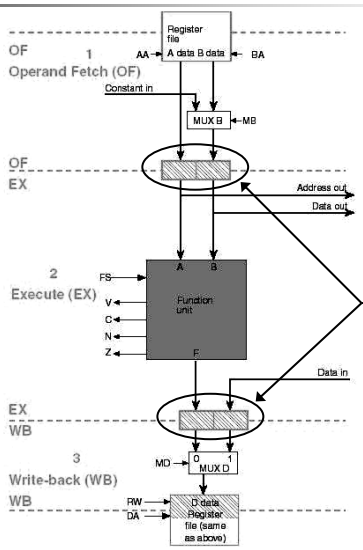
- Can process next operation when current operation is sent to another stage
  - Every worker only finish a part of the product in the assembly line
- Can have almost n times improvements on total speed if multiple operations are processed

Fig. 7-22 Assembly Line Analogy to Datapath Pipeline

6-50

# Pipelined Datapath Design

the number of stages can be decided by designers




registers are inserted between different stages to store the partial results (extra delay will be incurred for storing values in registers)

6-51

# Execution in Pipelined Datapath

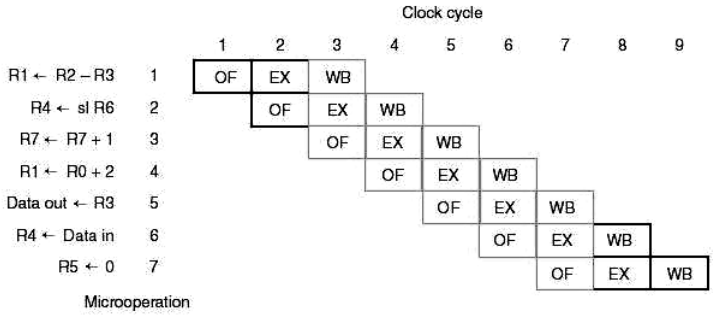



Fig. 7-24 Pipeline Execution Pattern for Microoperation Sequence in Table 7-13

Controller design for piplelined datapath will be more complex !!

6-52

# Pipelined Control

store the fetched instruction

store the control signals

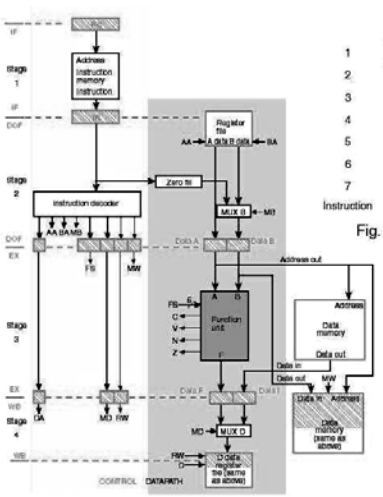



Fig. 8-34 Block Diagram of Pipelined Computer

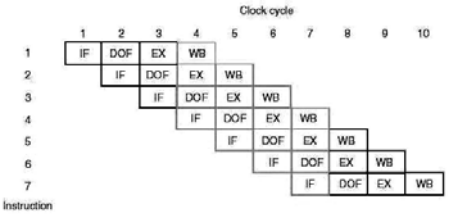



Fig. 8-35 Pipeline Execution Pattern of Register Number Program

registers are inserted between each stage in both controller and datapath

6-53