
Logic and Computer Design Fundamentals

Chapter 8 – Sequencing and Control

Charles Kime & Thomas Kaminski

© 2004 Pearson Education, Inc.

[Terms of Use](#)

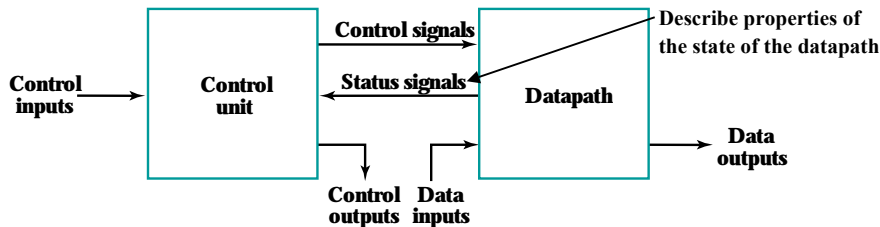
(Hyperlinks are active in View Show mode)

Overview

- **Datapath and Control**
- **Algorithmic State Machines (ASM)**
 - ASM chart
 - Timing considerations
- **ASM chart examples**
 - Binary multiplier
- **Hardwired Control**
 - Control design methods
 - Sequence register and decoder
 - One flip-flop per state
- **Microprogrammed control**

Datapath and Control

- **Datapath** - performs data transfer and processing operations
- **Control Unit** - Determines the enabling and sequencing of the operations



- **The control unit receives:**
 - External control inputs
 - Status signals
- **The control unit sends:**
 - Control signals
 - Control outputs

Logic and Computer Design Fundamentals
PowerPoint® Slides
© 2004 Pearson Education, Inc.

Chapter 8 3

Control Unit Types

- **Two distinct classes:**
 - Programmable
 - Non-programmable.
- **A programmable control unit has:**
 - A program counter (PC) or other sequencing register with contents that points to the next instruction to be executed
 - An external ROM or RAM array for storing instructions and control information
 - Decision logic for determining the sequence of operations and logic to interpret the instructions
- **A non-programmable control units does not fetch instructions from a memory and is not responsible for sequencing instructions**
 - This type of control unit is our focus in this chapter

Logic and Computer Design Fundamentals
PowerPoint® Slides
© 2004 Pearson Education, Inc.

Chapter 8 4

Algorithmic State Machines

- The function of a state machine (or sequential circuit) can be represented by a state table or a state diagram.
- A flowchart is a way of showing actions and control flow in an algorithm.
- An Algorithmic State Machine (ASM) is simply a flowchart-like way to specify state diagrams for sequential logic and, optionally, actions performed in a datapath.
- While flowcharts typically do not specify “time”, an ASM explicitly specifies a sequence of actions and their timing relationships.

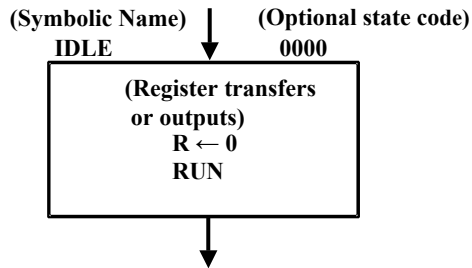
ASM Primitives

- | | |
|---|---|
| 1.State Box
(a rectangle) | ▪ The State Box is a rectangle, marked with the symbolic state name, containing register transfers and output signals activated when the control unit is in the state. |
| 2.Scalar
Decision Box
(a diamond) | ▪ The Scalar Decision Box is a diamond that describes the effects of a specific input condition on the control. It has one input path and two exit paths, one for TRUE (1) and one for FALSE (0). |
| 3.Vector
Decision Box
(a hexagon) | ▪ The Vector Decision Box is a hexagon that describes the effects of a specific n-bit ($n > 2$) vector of input conditions on the control. It has one input path and up to 2^n exit paths, each corresponding to a binary vector value. |
| 4.Conditional
Output Box
(oval). | ▪ The Conditional Output Box is an oval with entry from a decision block and outputs activated for the decision conditions being satisfied. |

State Box

- A rectangle with:

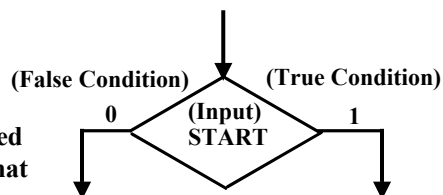
- The symbolic name for the state marked outside the upper left top
- Containing register transfer operations and outputs activated within or while leaving the state
- An optional state code, if assigned, outside the upper right top



Scalar Decision Box

- A diamond with:

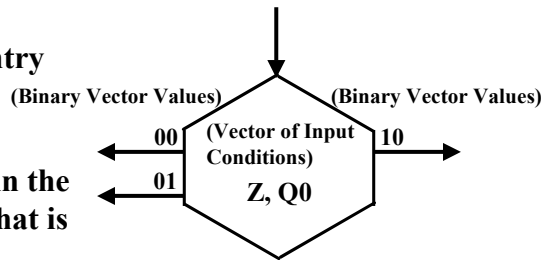
- One input path (entry point).
- One input condition, placed in the center of the box, that is tested.
- A TRUE exit path taken if the condition is true (logic 1).
- A FALSE exit path taken if the condition is false (logic 0).



Vector Decision Box

- A hexagon with:

- One Input Path (entry point).
- A vector of input conditions, placed in the center of the box, that is tested.
- Up to 2^n output paths. The path taken has a binary vector value that matches the vector input condition

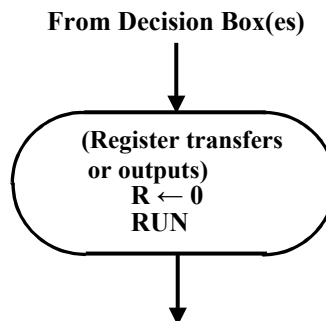


Conditional Output Box

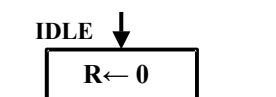
- An oval with:

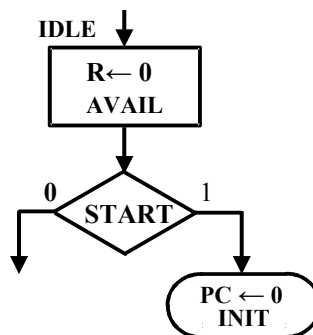
- One input path from a decision box or decision boxes.
- One output path
- Register transfers or outputs that occur only if the conditional path to the box is taken.

- Transfers and outputs in a state box are Moore type - dependent only on state
- Transfers and outputs in a conditional output box are Mealy type - dependent on both state and inputs



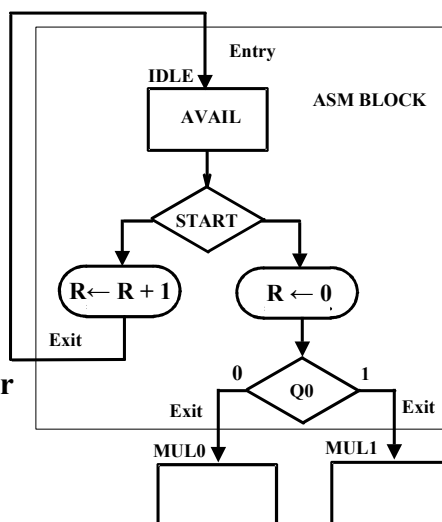
Connecting Boxes Together

- By connecting boxes together, we begin to see the power of expression.
 - What are the:
 - Inputs?
 - Outputs?
 - Conditional Outputs?
 - Transfers?
 - Conditional Transfers?
- 
- ```
graph TD; Entry(()) -- IDLE --> Init[R ← 0
AVAIL]; Init --> Start{START}; Start -- 0 --> Exit(()); Start -- 1 --> PC[PC ← 0
UNTIL];
```
- The flowchart illustrates a simple program structure. It begins with an entry point labeled 'IDLE' with a downward arrow. This leads to a rectangular process box containing the assignments 'R ← 0' and 'AVAIL'. An arrow from this box points down to a diamond-shaped decision box labeled 'START'. From the 'START' box, two paths emerge: a left path labeled '0' that leads to an exit arrow pointing downwards, and a right path labeled '1' that leads to another rectangular process box containing the assignments 'PC ← 0' and 'UNTIL'.



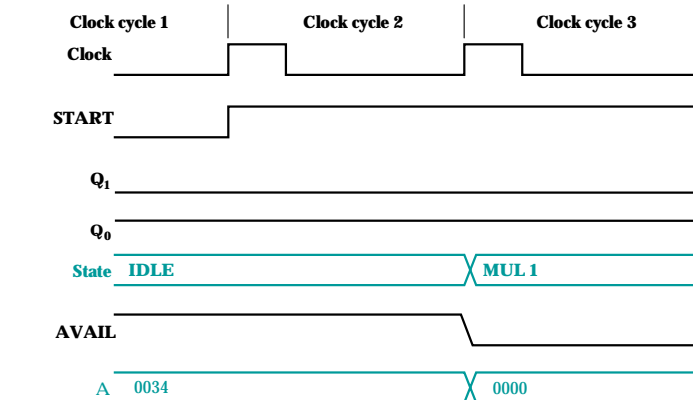
## ASM Blocks

- One state box along with all decision and conditional output boxes connected to it is called an ASM Block.
- The ASM Block includes all items on the path from the current state to the same or other states.



# ASM Timing

- Outputs appear while in the state
- Register transfers occur at the clock while exiting the state - New value occur in the next state!



Logic and Computer Design Fundamentals  
PowerPoint® Slides  
© 2004 Pearson Education, Inc.

Chapter 8 13

## Multiplier Example

- Example:  $(101 \times 011)$  Base 2
- Note that the partial product summation for  $n$  digits, base 2 numbers requires adding up to  $n$  digits (with carries) in a column.
- Note also  $n \times m$  digit multiply generates up to an  $m + n$  digit result (same as decimal).
- Partial products are:  
 $101 \times 0$ ,  $101 \times 1$ , and  $101 \times 1$

$$\begin{array}{r}
 \phantom{00}101 \\
 \times \phantom{00}011 \\
 \hline
 \phantom{00}101 \\
 \phantom{00}010 \\
 000 \\
 \hline
 001111
 \end{array}$$

Logic and Computer Design Fundamentals  
PowerPoint® Slides  
© 2004 Pearson Education, Inc.

Chapter 8 14

### Example (1 0 1) x (0 1 1) Again

- Reorganizing example to follow hardware algorithm:

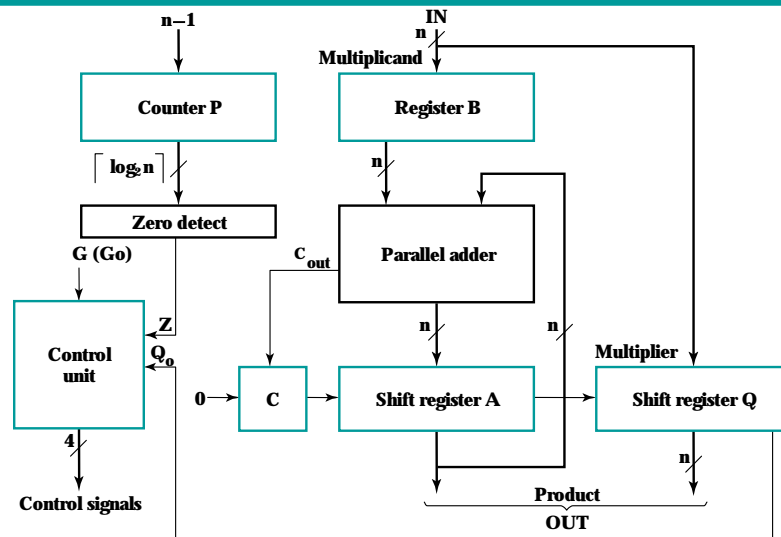
|   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|
|   | 1 | 0 | 1 |   |   |   |   |
| x | 0 | 1 | 1 |   |   |   |   |
|   | 0 | 0 | 0 | 0 |   |   |   |
| + | 1 | 0 | 1 |   |   |   |   |
|   | 0 | 1 | 0 | 1 |   |   |   |
|   | 0 | 0 | 1 | 0 | 1 |   |   |
| + | 1 | 0 | 1 |   |   |   |   |
|   | 0 | 1 | 1 | 1 | 1 |   |   |
|   | 0 | 0 | 1 | 1 | 1 | 1 |   |
|   | 0 | 0 | 0 | 1 | 1 | 1 | 1 |

Clear C || A  
Multiplier<sub>0</sub> = 1 => Add B  
**Addition**  
Shift Right (Zero-fill C)  
Multiplier<sub>1</sub> = 1 => Add B  
**Addition**  
Shift Right  
Multiplier<sub>2</sub> = 0 => No Add,  
**Shift Right**

Logic and Computer Design Fundamentals  
PowerPoint® Slides  
© 2004 Pearson Education, Inc.

Chapter 8 15

## Multiplier Example: Block Diagram



Logic and Computer Design Fundamentals  
PowerPoint® Slides  
© 2004 Pearson Education, Inc.

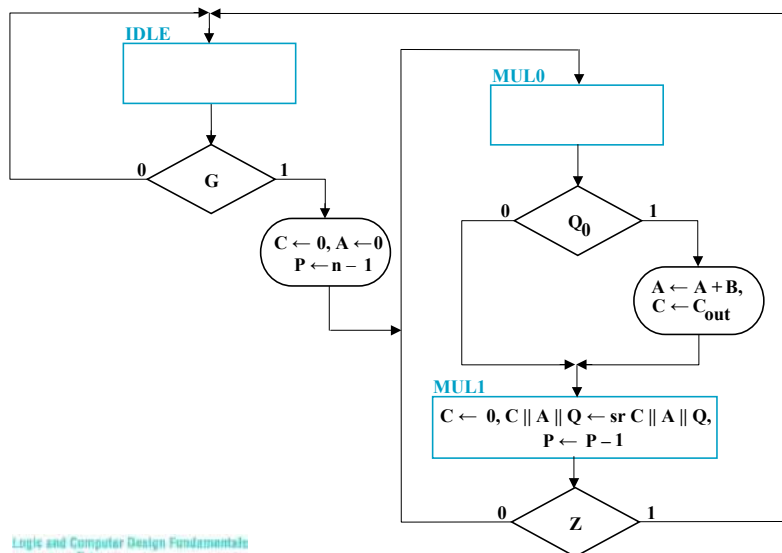
Chapter 8 16



## Multiplexer Example: Operation

1. The multiplicand (top operand) is loaded into register B.
2. The multiplier (bottom operand) is loaded into register Q.
3. Register C|| Q is initialized to 0 when G becomes 1.
4. The partial products are formed in register C||A||Q.
5. Each multiplier bit, beginning with the LSB, is processed (if bit is 1, use adder to add B to partial product; if bit is 0, do nothing)
6. C||A||Q is shifted right using the shift register
  - Partial product bits fill vacant locations in Q as multiplier is shifted out
  - If overflow during addition, the outgoing carry is recovered from C during the right shift
7. Steps 5 and 6 are repeated until Counter P = 0 as detected by Zero detect.
  - Counter P is initialized in step 4 to  $n - 1$ ,  $n$  = number of bits in multiplier

## Multiplier Example: ASM Chart



## Multiplier Example: ASM Chart (continued)

- Three states are employed using a combined Mealy - Moore output model:
  - IDLE - state in which:
    - the outputs of the prior multiply is held until Q is loaded with the new multiplicand
    - input G is used as the condition for starting the multiplication, and
    - C, A, and P are initialized
  - MUL0 - state in which conditional addition is performed based on the value of  $Q_0$ .
  - MUL1 - state in which:
    - right shift is performed to capture the partial product and position the next bit of the multiplier in  $Q_0$
    - the terminal count of 0 for down counter P is used to sense completion or continuation of the multiply.

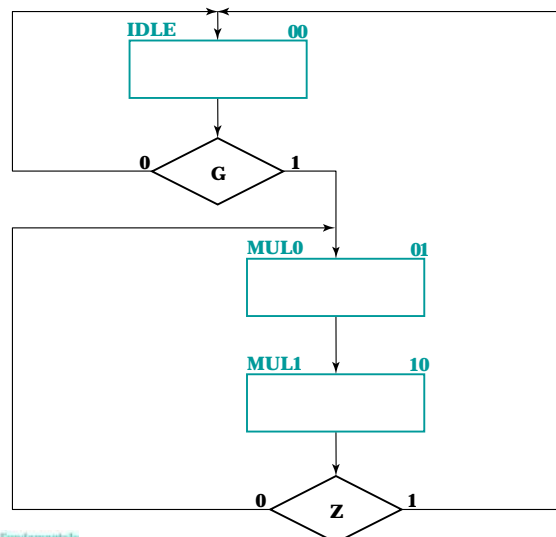
## Multiplier Example: Control Signal Table

| 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           |
|                                       | $C \parallel A \parallel Q \leftarrow \text{sr } C \parallel A \parallel Q$ | Shift_dec           | MUL1               |
| Register B:                           | $B \leftarrow IN$                                                           | Load_B              | LOADB              |
| Flip-Flop C:                          | $C \leftarrow 0$                                                            | Clear_C             | IDLE · G + MUL1    |
|                                       | $C \leftarrow C_{\text{out}}$                                               | Load                | —                  |
| Register Q:                           | $Q \leftarrow IN$                                                           | Load_Q              | LOADQ              |
|                                       | $C \parallel A \parallel Q \leftarrow \text{sr } C \parallel A \parallel Q$ | Shift_dec           | —                  |
| Counter P:                            | $P \leftarrow n - 1$                                                        | Initialize          | —                  |
|                                       | $P \leftarrow P - 1$                                                        | Shift_dec           | —                  |

## Multiplier Example: Control Table (continued)

- Signals are defined on a register basis
- LOADQ and LOADB are external signals controlled from the system using the multiplier and will not be considered a part of this design
- Note that many of the control signals are “reused” for different registers.
- These control signals are the “outputs” of the control unit
- With the outputs represented by the table, they can be removed from the ASM giving an ASM that represents only the sequencing (next state) behavior

## Multiplier Example - Sequencing Part of ASM



# Hardwired Control

## ■ Control Design Methods

- The procedure from Chapter 6
- Procedure specializations that use a single signal to represent each state
  - Sequence Register and Decoder
    - Sequence register with encoded states, e.g., 00, 01, 10, 11.
    - Decoder outputs produce “state” signals, e.g., 0001, 0010, 0100, 1000.
  - One Flip-flop per State
    - Flip-flop outputs as “state” signals, e. g., 0001, 0010, 0100, 1000.

## Multiplier Example: Sequencer and Decoder Design

- Initially, use sequential circuit design techniques from Chapter 4.
- First, define:
  - States: IDLE, MUL0, MUL1
  - Input Signals: G, Z,  $Q_0$  ( $Q_0$  affects outputs, not next state)
  - Output Signals: Initialize, LOAD, Shift\_Dec, Clear\_C
  - State Transition Diagram (Use Sequencing ASM on Slide 22)
  - Output Function: Use Table on Slide 20
- Second, find
  - State Assignments (two bits required)
  - We will use two state bits to encode the three state IDLE, MUL0, and MUL1.

| State  | M1 | M0 |
|--------|----|----|
| IDLE   | 0  | 0  |
| MUL0   | 0  | 1  |
| MUL1   | 1  | 0  |
| Unused | 1  | 1  |

## Multiplier Example: Sequencer and Decoder Design (continued)

- Assuming that state variables M1 and M0 are decoded into states, the next state part of the state table is:

| Current State | Input<br>G Z | Next State<br>M1 M0 | Current State<br>M1 M0 | Input<br>G Z | Next State<br>M1 M0 |
|---------------|--------------|---------------------|------------------------|--------------|---------------------|
| IDLE          | 0 0          | 0 0                 | MUL1                   | 0 0          | 0 1                 |
| IDLE          | 0 1          | 0 0                 | MUL1                   | 0 1          | 0 0                 |
| IDLE          | 1 0          | 0 1                 | MUL1                   | 1 0          | 0 1                 |
| IDLE          | 1 1          | 0 1                 | MUL1                   | 1 1          | 0 0                 |
| MUL0          | 0 0          | 1 0                 | Unused                 | 0 0          | d d                 |
| MUL0          | 0 1          | 1 0                 | Unused                 | 0 1          | d d                 |
| MUL0          | 1 0          | 1 0                 | Unused                 | 1 0          | d d                 |
| MUL0          | 1 1          | 1 0                 | Unused                 | 1 1          | d d                 |

## Multiplier Example: Sequencer and Decoder Design (continued)

- Finding the equations for M1 and M0 is easier due to the decoded states:

$$M1 = MUL0$$

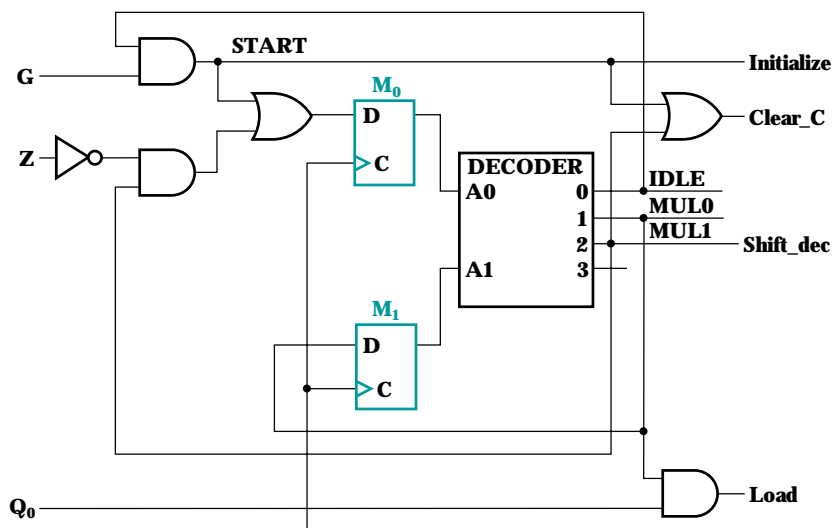
$$M0 = IDLE \cdot G + MUL1 \cdot \overline{Z}$$

- Note that since there are five variables, a K-map is harder to use, so we have directly written reduced equations.
- The output equations using the decoded states:
  - Initialize = IDLE · G
  - Load = MUL0 · Q<sub>0</sub>
  - Clear\_C = IDLE · G + MUL1
  - Shift\_dec = MUL1

## Multiplier Example: Sequencer and Decoder Design (continued)

- Doing multiple level optimization, extract  $IDLE \cdot G$ :  
 $START = IDLE \cdot G$   
 $M1 = MUL0$   
 $M0 = START + MUL1 \cdot \overline{Z}$   
Initialize = START  
Load =  $MUL0 \cdot Q_0$   
Clear\_C =  $START + MUL1$   
Shift\_dec = MUL1
- The resulting circuit using flip-flops, a decoder, and the above equations is given on the next slide.

## Multiplier Example: Sequencer and Decoder Design (continued)

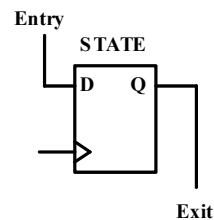
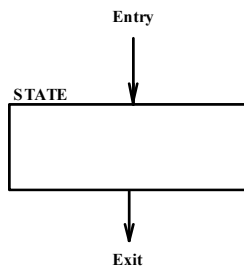


# One Flip-Flop per State

- This method uses one flip-flop per state and a simple set of transformation rules to implement the circuit.
- The design starts with the ASM chart, and replaces
  1. State Boxes with flip-flops,
  2. Scalar Decision Boxes with a demultiplexer with 2 outputs,
  3. Vector Decision Boxes with a (partial) demultiplexer
  4. Junctions with an OR gate, and
  5. Conditional Outputs with AND gates.
- Each is discussed detail below.
  - Figure 8-11 is the end result.

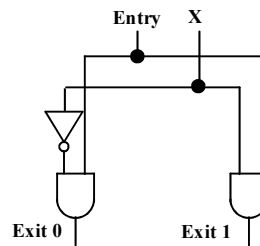
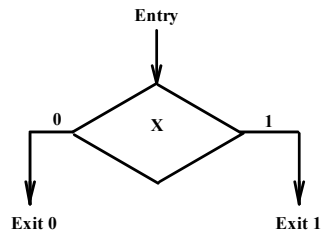
## State Box Transformation Rules

- Each state box transforms to a D Flip-Flop
- Entry point is connected to the input.
- Exit point is connected to the Q output.



# Scalar Decision Box Transformation Rules

- Each Decision box transforms to a Demultiplexer
- Entry points are "Enable" inputs.
- The Condition is the "Select" input.
- Decoded Outputs are the Exit points.

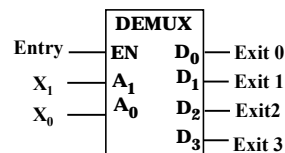
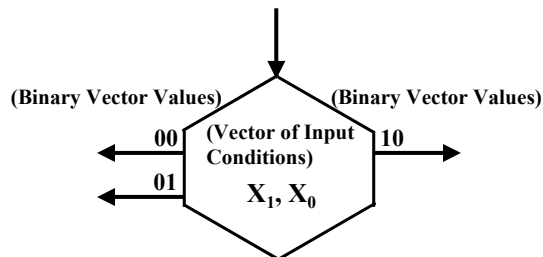


Logic and Computer Design Fundamentals  
PowerPoint® Slides  
© 2004 Pearson Education, Inc.

Chapter 8 31

# Vector Decision Box Transformation Rules

- Each Decision box transforms to a Demultiplexer
- Entry point is Enable inputs.
- The Conditions are the Select inputs.
- Demultiplexer Outputs are the Exit points.



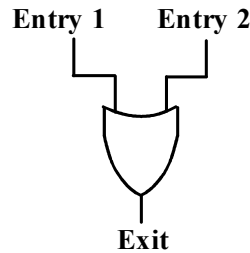
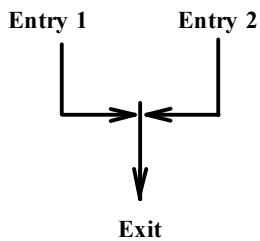
Logic and Computer Design Fundamentals  
PowerPoint® Slides  
© 2004 Pearson Education, Inc.

Chapter 8 32



## Junction Transformation Rules

- Where two or more entry points join, connect the entry variables to an OR gate
- The Exit is the output of the OR gate

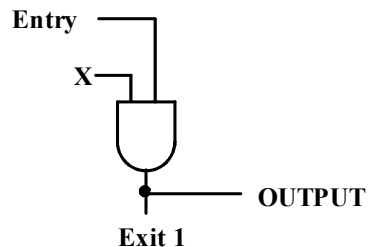
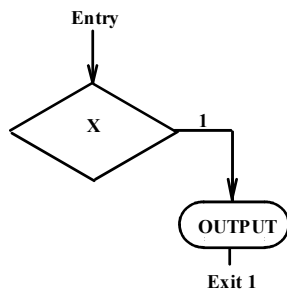


Logic and Computer Design Fundamentals  
PowerPoint® Slides  
© 2004 Pearson Education, Inc.

Chapter 8 33

## Conditional Output Box Rules

- Entry point is Enable input.
- The Condition is the "Select" input.
- Demultiplexer Outputs are the Exit points.
- The Control OUTPUT is the same signal as the exit value.



Logic and Computer Design Fundamentals  
PowerPoint® Slides  
© 2004 Pearson Education, Inc.

Chapter 8 34

---



Chapter 8 35

---

- Logic and Computer Design Fundamentals  
PowerPoint® Slides  
© 2004 Pearson Education, Inc.

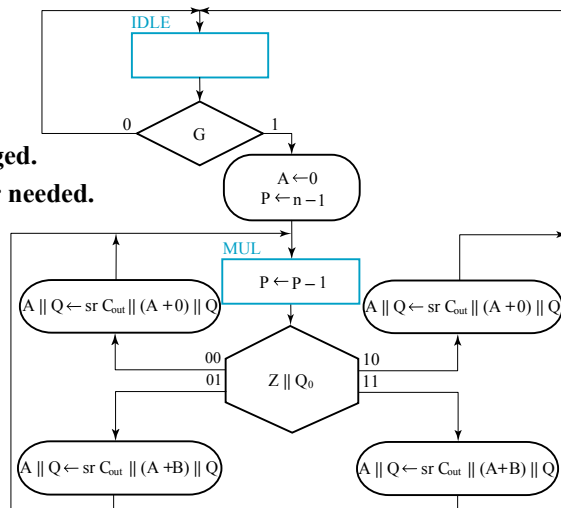
Chapter 8 36

## Speeding Up Multiply (continued)

- **Examining the operations in MUL0 and MUL1:**
  - In MUL0, a conditional add of B is performed, and
  - In MUL1, a right shift of  $C \parallel A \parallel Q$  in a shift register, the decrementing of P, and a test for  $P = 0$  (on the old value of P) are all performed in MUL1
- Any solution that uses one state must combine all of the operations listed into one state
- The operations involving P are already done in a single state, so are not a problem.
- The right shift, however, depends on the result of the conditional addition. So these two operations must be combined!

## Speeding Up Multiply (continued)

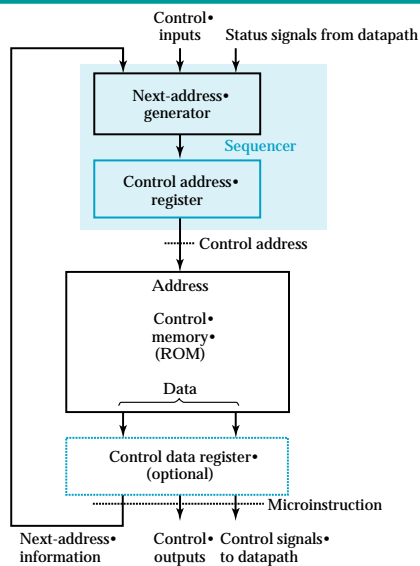
- By replacing the shift register with a combinational shifter and combining the adder and shifter, the states can be merged.
- The C-bit is no longer needed.
- In this case, Z and  $Q_0$  have been made into a vector. This is not essential to the solution.
- The ASM chart =>



# Microprogrammed Control

- **Microprogrammed Control** — a control unit with binary control values stored as words in memory.
- **Microinstructions** — words in the control memory.
- **Microprogram** — a sequence of microinstructions.
- **Control Memory** — RAM or ROM memory holding the microinstructions.
- **Writeable Control Memory** — RAM Memory into which microinstructions may be written

## Microprogrammed Control (continued)



# Terms of Use

---

- © 2004 by Pearson Education, Inc. All rights reserved.
- The following terms of use apply in addition to the standard Pearson Education [Legal Notice](#).
- Permission is given to incorporate these materials into classroom presentations and handouts only to instructors adopting Logic and Computer Design Fundamentals as the course text.
- Permission is granted to the instructors adopting the book to post these materials on a protected website or protected ftp site in original or modified form. All other website or ftp postings, including those offering the materials for a fee, are prohibited.
- You may not remove or in any way alter this Terms of Use notice or any trademark, copyright, or other proprietary notice, including the copyright watermark on each slide.
- [Return to Title Page](#)