ECE290 Fall 2012
Lecture 18

Dr. Zbigniew Kalbarczyk

Today

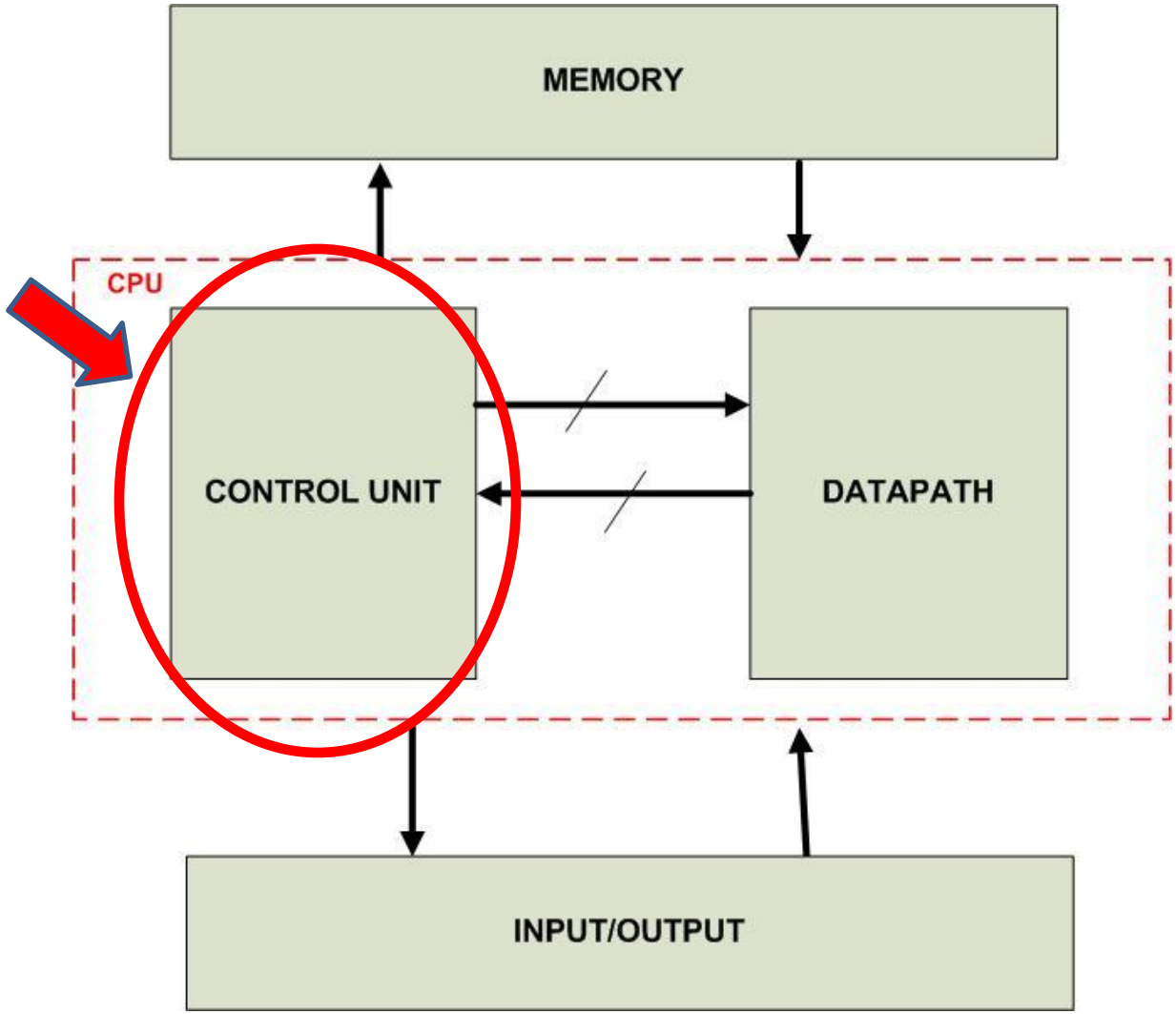
- Introduction to Control Unit
- Binary Multiplier Example
- Hardwire Control Unit for Binary Multiplier

Today

- Introduction to Control Unit
- Binary Multiplier Example
- Hardwire Control Unit for Binary Multiplier

Components of a Simple Computer

- Focus on designing a **control unit** of a CPU in a simple computer



A Control Unit

- A **control unit** generates control input and control output signals that
  - activate micro-operations within data path to perform specified processing tasks
  - determine its own next state

Requirements on Control Units

- In **programmable** systems (use control memory and control program), a control unit consists of
  - Program counter (PC) and its decision logic
  - Logic to interpret instructions fetched from RAM or ROM

- In **non-programmable** systems (use gates + connections), a control unit determines
  - the operations to be performed and
  - the sequence of operations;
    based on inputs and the status bits from data path.

Design Exercise: A Control Unit for Multiplication

- **Multiplication of Unsigned Numbers**
  - If $0<r<2^n$ and $0<s<2^n$ then 2n bits suffice to hold the product of two n-bit binary number multiplication
  - **Example:** Multiplicand **s=111**; Multiplier **r=110**

			1	**1**	**1**
		X	1	1	0
			0	0	0
		1	1	1	0
	1	1	1	0	0
1	0	1	0	1	0

Design Considerations

- Needed: **2n bit adder**
  - Multiplicand: register B of size=n
  - Multiplier: register Q of size=n
  - Products: register A of size=2n
- **Can we accomplish the same with n bit adder?**
- ***Design Idea:***
  - Repeatedly add s (multiplicand) to partial product
  - Keep shifting partial products to right so addition of s always occur in same place

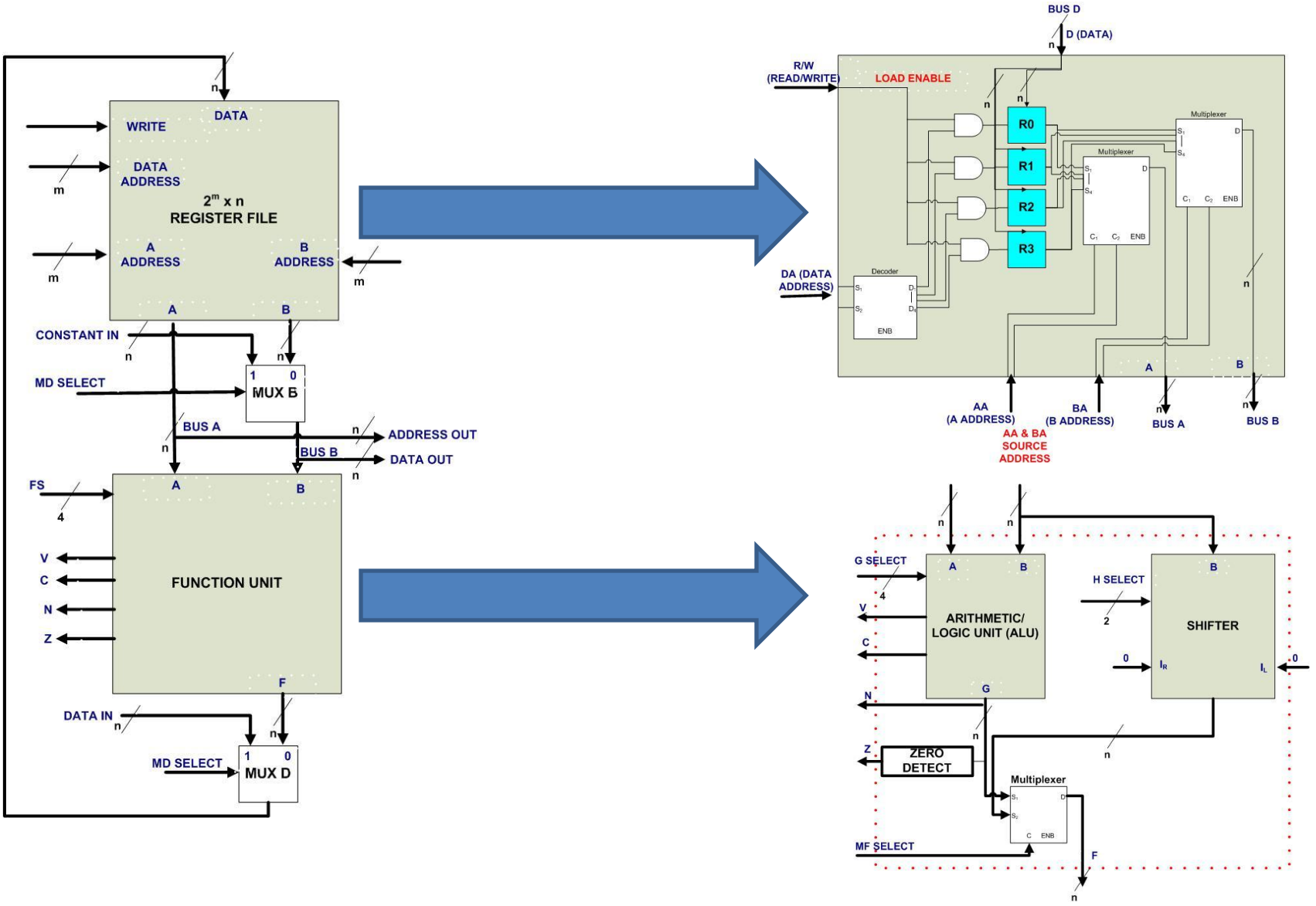
Multiplication Execution with A, B and Q Registers

Init	Register A	=product		Register Q	=multiplier	
C	p2	p1	p0	r2	r1	r0
0	0	0	0	1	1	0
	0	**0**	**0**	Add and then shift right		
0	**0**	**0**	**0**	**1**	**1**	**0**
C	p2	p1	p0	r2	r1	r0
0	0	0	0	0	1	1
	1	**1**	**1**	Add and then shift right		
0	**1**	**1**	**1**	**0**	**1**	**1**
C	p2	p1	p0	r2	r1	r0
0	0	1	1	1	0	1
	1	**1**	**1**	Add and then shift right		
1	**0**	**1**	**0**	**1**	**0**	**1**
C	p2	p1	p0	r2	r1	r0
0	1	0	1	0	1	0

Idea in Practice

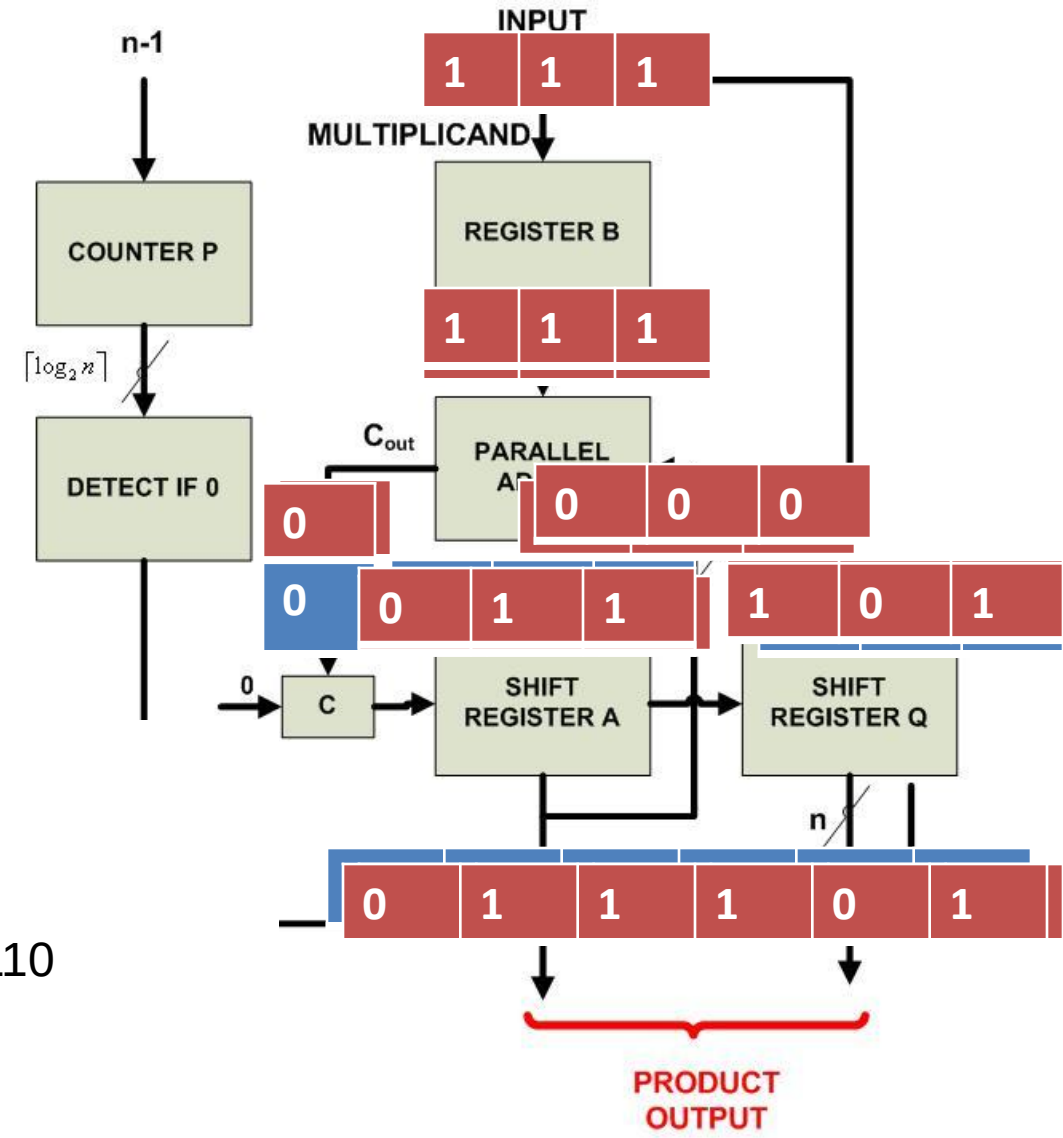
- 1. Load multiplier into Q register. As bits shift into Q from left, examine successive bits of multiplier at Q[0] (rightmost bit)
- 2. Set C <- 0 so that 0 shifts into leftmost bit of A register when no addition
- 3. Use counter P to count down from n-1 to 0
- 4. Define Z=1 when current value of P is 0…0

Data Path Components for Binary Multiplier



Binary Multiplier: Data Path

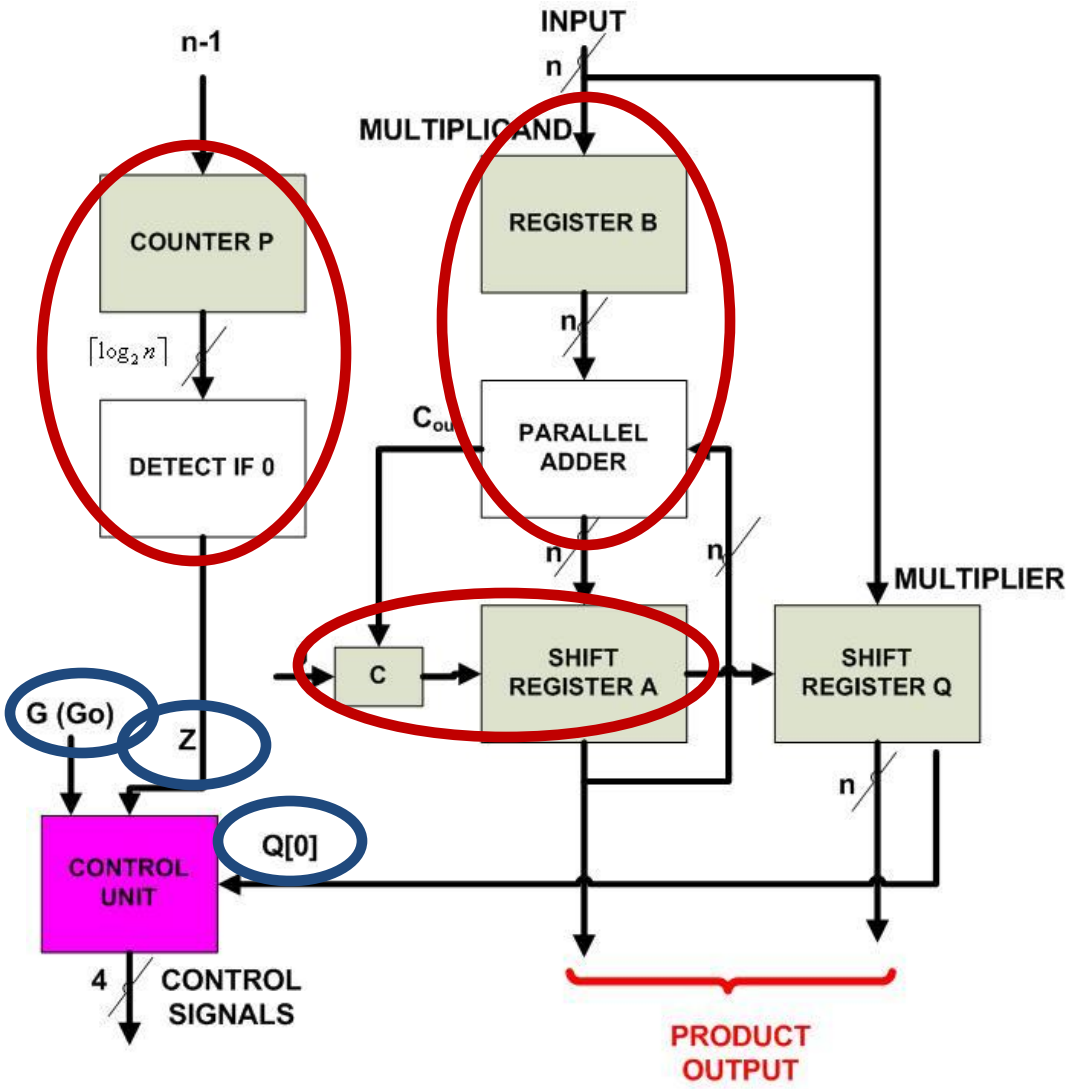
- Block Diagram of Data Path Includes
  - Register File
  - Function Unit
  - Buses



Example: Step 1 of executing
111 x 110 (n=3)
Multiplicand=111; Multiplier = 110
Add and then shift right

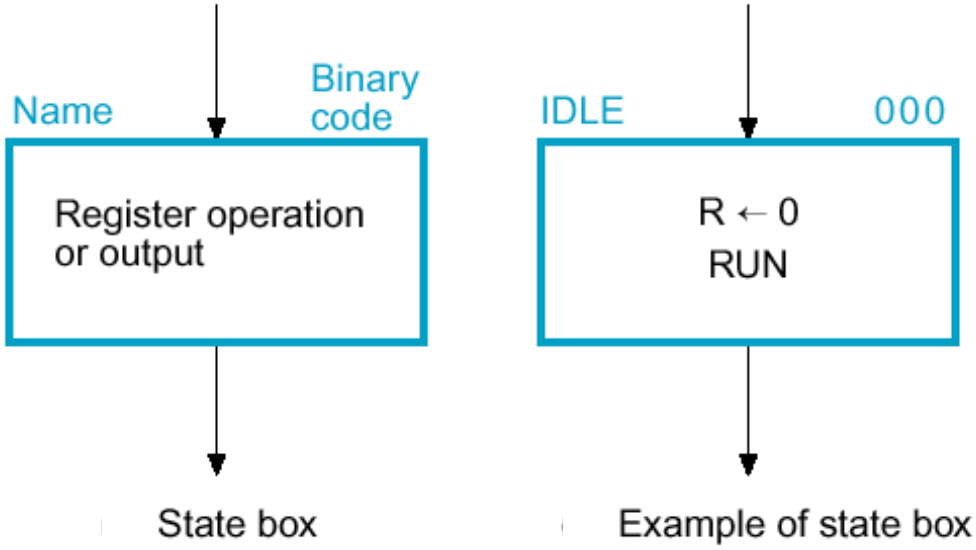
Binary Multiplier: Data Path & Control Unit

- Block Diagram of Control Unit Includes
  - Status Bits (Z =1 if P=0)
  - Go signal (Go=1 to start multiplication)
  - Q[0]
    - Q[0]=1 ~ add B register & updated A & C registers;
    - Q[0]=0 ~ A & C registers do not change
-



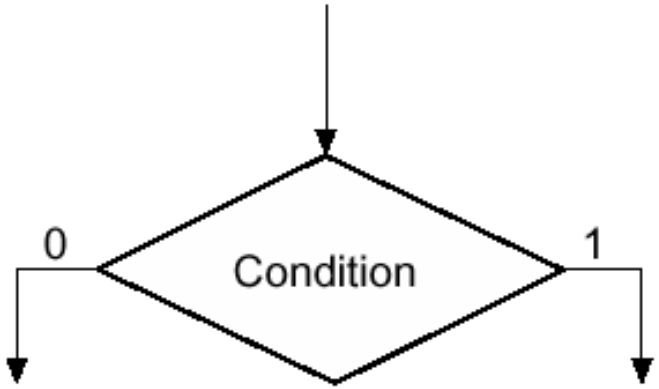
Algorithmic State Machines

- **Algorithmic State Machine (ASM) Chart**
  - a sequence of events
  - the **timing relationship** between the states
  - Three basic elements
    - state box, decision box, conditional output box
  - **State box**



Algorithmic State Machines (2)

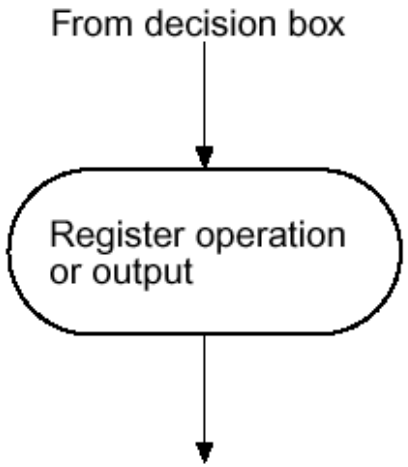
- **Decision box**
  - Condition; single variable or Boolean expression



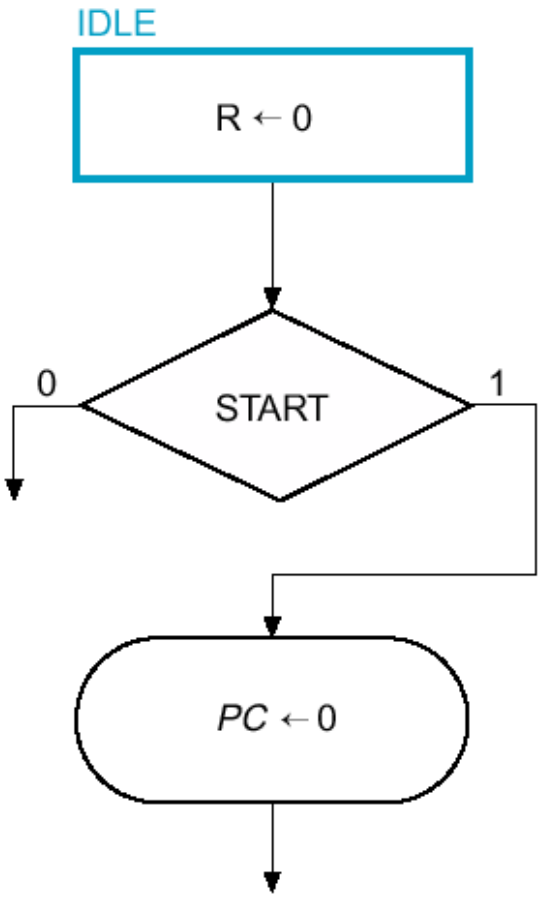
Decision box

Algorithmic State Machines (3)

- **Conditional output box**



Conditional output box

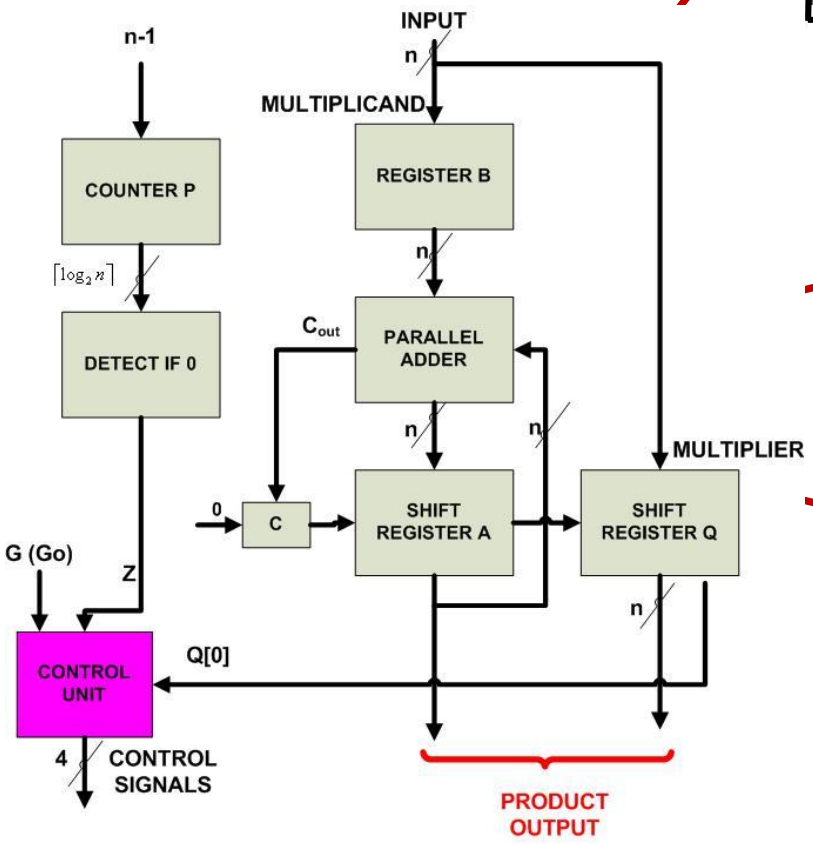


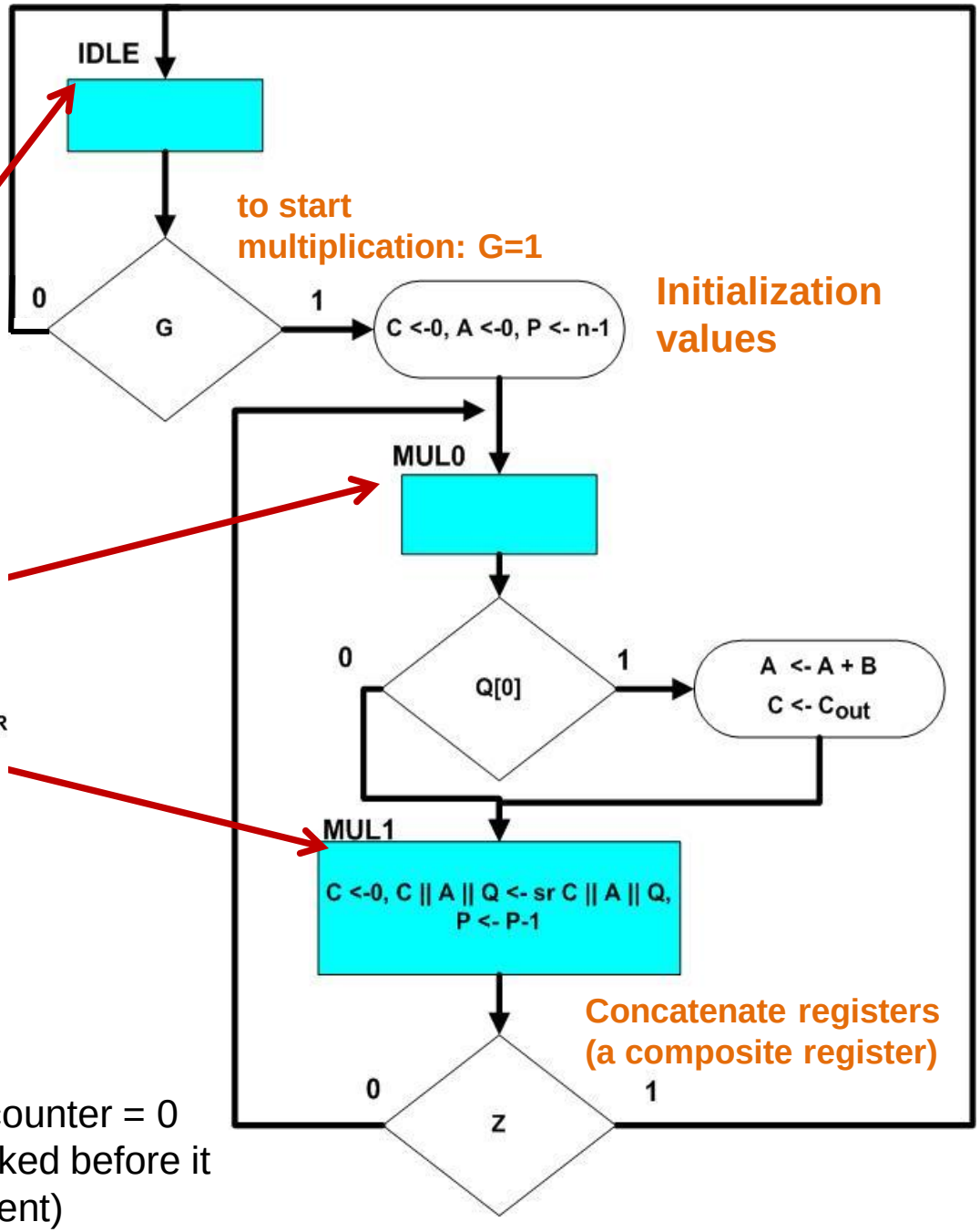
Example of decision and condition output box

Binary Multiplier Described by ASM Chart

- Assume:
  - Multiplicand is in register B and
  - Multiplier is in register Q
  - Loading of B and Q is not handled by the control unit explicitly
  - Result will be a concatenation of registers C, A and Q

ASM Chart for Binary Multiplier





Z=1 iff P counter = 0
(P is checked before it is decrement)

Hardwire Control Design

Two parts of the design:

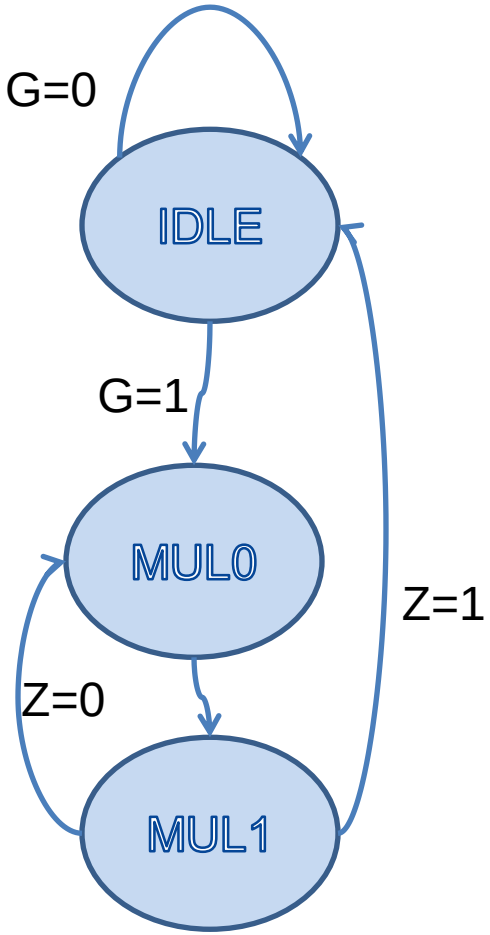
- **Control of micro-operations**
  - Table of control signals defined in terms of states and inputs
- **Sequencing of the control unit and micro-operations**
  - Table of transition of states

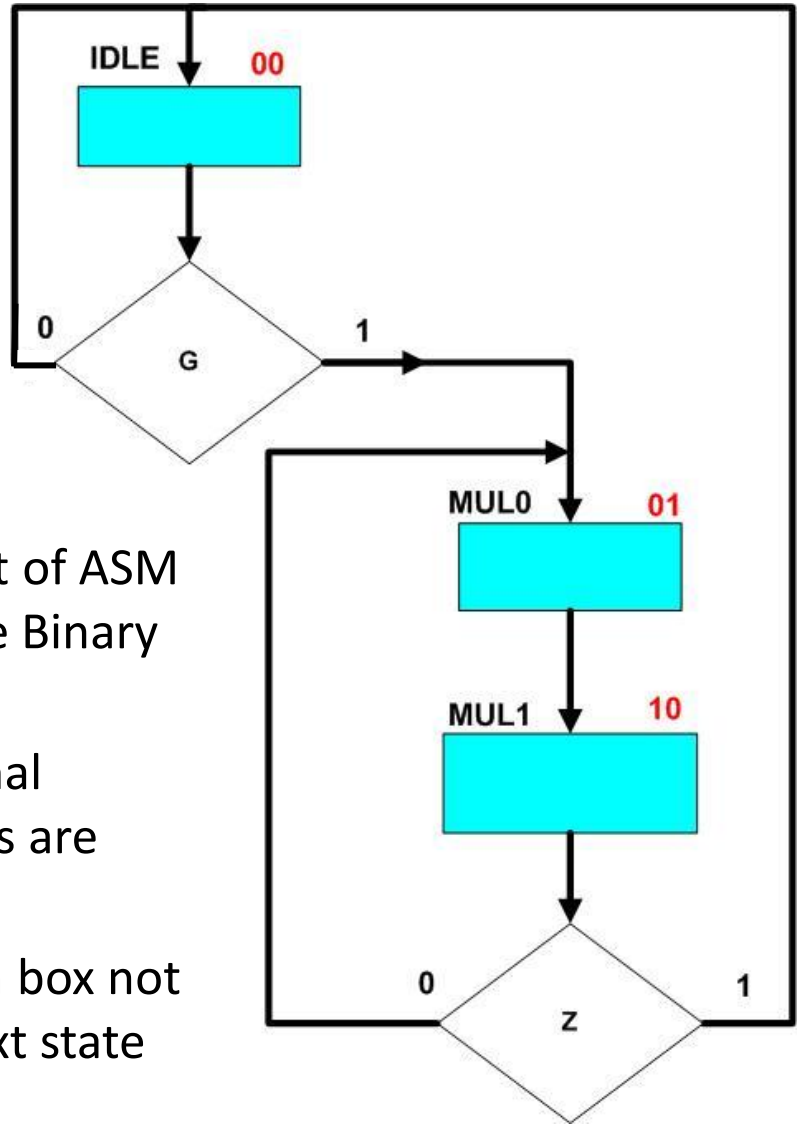
- The logic for these two parts can be shared

Control of Micro-operations for Binary Multiplier

	Micro-operations	Control Signal Name	Boolean Expression
Register A	A <-0	Initialize	IDLE * G
	A <- A + B	Load_AC	MUL0 * Q[0]
	Shift right (C||A||Q)	Shift_dec	MUL1
Register B	B <- IN	Load_B	LoadB
Register C	C <- 0	Clear_C	MUL1 + IDLE * G
	C <- C_{OUT}	Load_AC	----
Register Q	Q <- IN	Load_Q	LoadQ
	Shift right (C||A||Q)	Shift_dec	----
Register P	P <- n-1	Initialize	----
	P <- P -1	Shift _dec	----

Sequencing the Control Unit and Micro-operations





Sequencing Part of ASM Chart for the Binary Multiplier:

- All conditional output boxes are removed
- Any decision box not affecting next state is removed

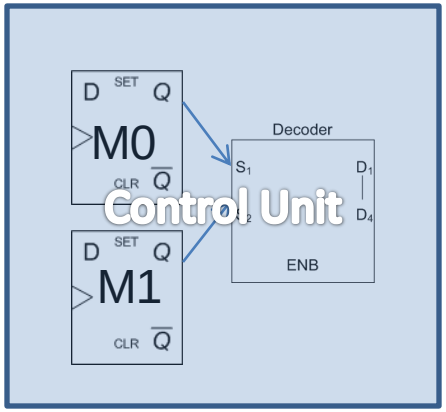
Hardwire Control: Implementation

1. **Sequence register and decoder**
   - **Use sequence register and decoder to generate a sequence of control signals from a control unit**
2. **One flip-flop per state**
   - **Use flip-flops in such a way that at any time only one flip-flop contains 1 and the rest contain 0**

Hardwire Control: Sequence Register and Decoder Method

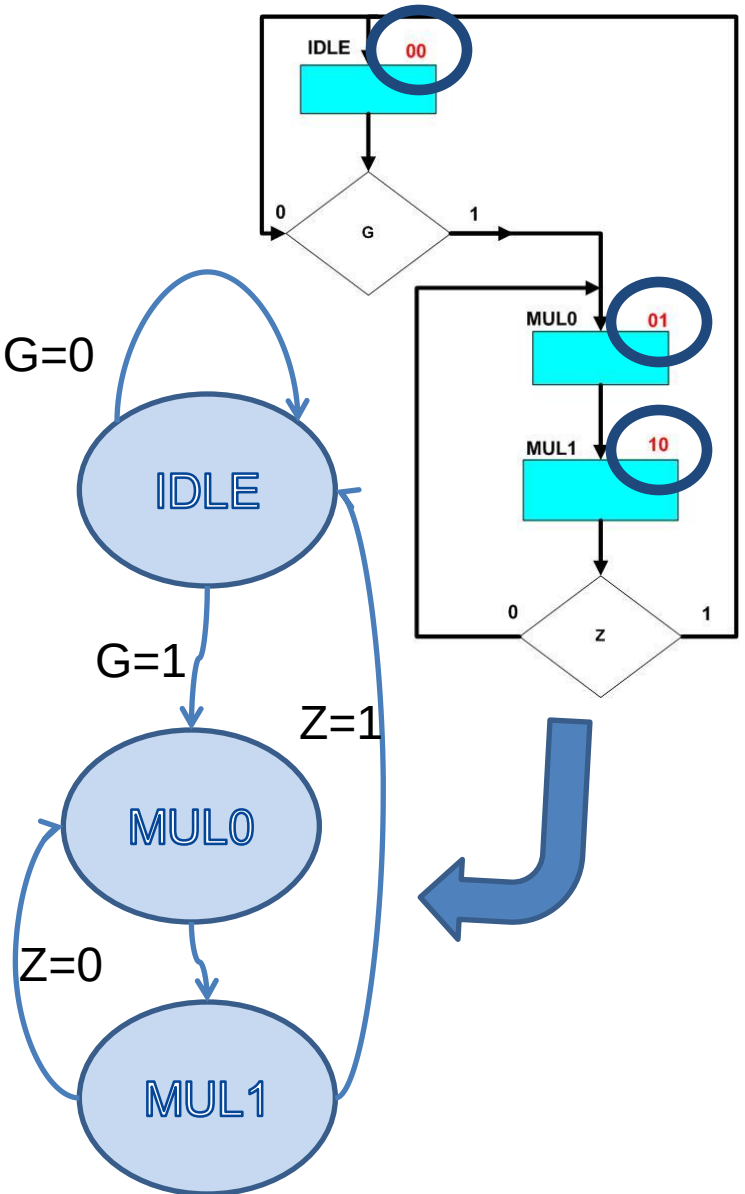
Method1: using sequence register and decoder:

1. Assign binary state to each ASM state
2. Keep binary state in sequence register
3. Use decoder to create a signal for each state
4. Design combinational logic to generate control signal and next state



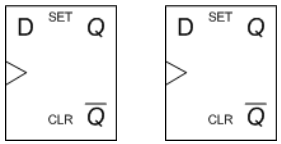
Method 1: From ASM to Control Signals

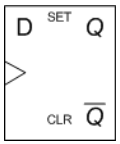
- Starting Point:
  - Two inputs: G, Z (plus Q[0])
  - Three states: IDLE, MUL0, MUL1 (treated as Boolean variables)
- Needed:
  - 2 Flip-Flops for the sequence register
  - 2-to-4 line decoder (three states => only 3 out 4 decoder outputs will be used)
  - State Table

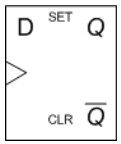


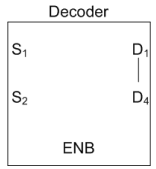
State Table for Sequence Register and Decoder

- Derive relationships between inputs & current states to generate signals to get to next states









	Present States		Inputs		Next State		Decoder Outputs		
Name	M1	M0	G	Z	M1+	M0+	IDLE	MUL0	MUL1
IDLE	0	0	**0**	X	0	0	1	0	0
	0	0	**1**	X	0	1	1	0	0
MUL0	**0**	**1**	X	X	1	0	0	**1**	**0**
MUL1	**1**	**0**	X	**0**	0	1	0	**0**	**1**
	1	**0**	X	**1**	0	0	0	**0**	**1**
------	1	1	X	X	X	X	X	X	X

IDLE=1 ~ I am not doing multiplication

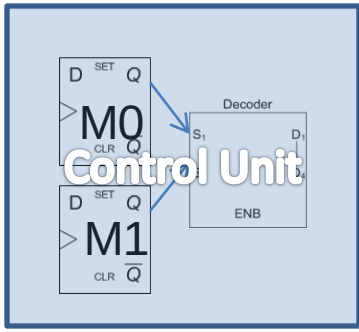
Boolean Relationships

- Boolean Expressions for State Transitions

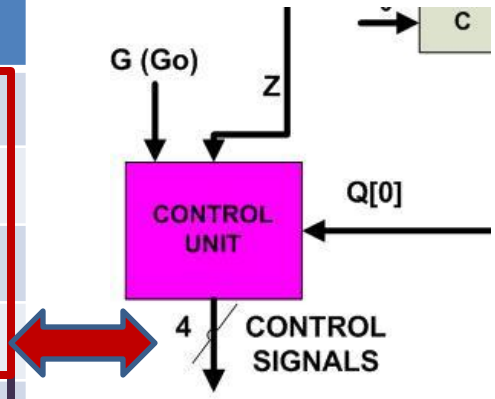
$$M0(t+1) = IDLE \cdot G + MUL1 \cdot \bar{Z}$$

$$M1(t+1) = MUL0$$

- Boolean Expressions for Control Signals



Control Signal Name	Boolean Expression
Initialize	IDLE * G
Load_AC	MUL0 * Q[0]
Shift_dec	MUL1
Clear_C	MUL1 + IDLE * G
Load_B	LoadB
Load_Q	LoadQ



Method1: Control Unit for Binary Multiplier