

---

# CEG 3320 - Digital System Design


Instructor: Travis Doom, Ph.D.  
331 Russ Engineering Center  
775-5105  
travis.doom@wright.edu  
<http://www.wright.edu/~travis.doom>

Lecture slides created by T. Doom for Wright State University's course in Digital System Design. Some slides contain fair use images or material used with permission from textbooks and slides by R. Haggard, F. Vahid, Y. Patt, J. Wakerly, M. Mano, and other sources.



# Module V:

## Register Transfer Level Design Decomposition

A large, light gray, curved shape that starts from the left edge of the slide and curves downwards and to the right, ending near the bottom right corner. It has a smooth, organic, wave-like border.

Design Decomposition  
RTL Language  
Datapath construction  
Control unit design

# Register Transfer Level Design Decomposition



Dealing with complexity

# Dealing with Complexity

- Practical synchronous sequential circuits are too complex to design at the flip-flop level.
  - A simple 8-bit CPU capable of storing only four values (in four GPRs) has at least  $4 \times 8 = 32$  1-bit state devices!
  - If the CPU had only 8 inputs it would still require a  $2^{32}$  by  $2^8$  truth table to represent the device. Each of these billion or so entries would have to contain a 32-bit next state. Storing  $2^{40}$  32-bit entries would require 4 TB!
    - K (kilo)  $2^{10}$  1024
    - M (mega)  $2^{20}$  1048576
    - G (giga)  $2^{30}$   $1.07 \times 10^9$
    - T (tera)  $2^{40}$   $1.10 \times 10^{12}$
  - Simplification would require a 40-variable K-map!
- How do we manage complex design?



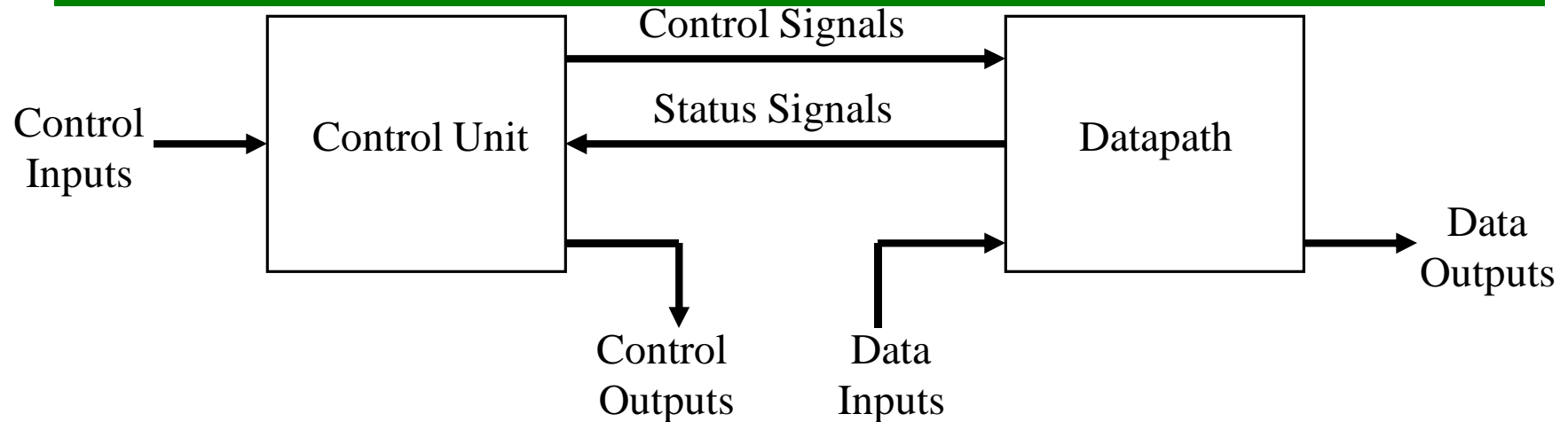
# Complex System Design

---

- Practical sequential designs, like combinational designs, require a hierarchical approach
  - Use well defined building blocks
    - Complex blocks made of simpler blocks (hierarchy)
    - Examples: Registers, counters
- Associate a high-level of behavior with those blocks (abstraction)
  - Design methodologies based on abstractions can more easily encompass complexity
  - Common function blocks: registers and counters
- ORDER a sequence of high-level behaviors that (when executed in the proper order) solves the overall problem
  - as per computer programming!



# Decomposing a Design: Control and Data



- Complex designs are generally broken down into high-level abstractions.
  - The *datapath* is home to one or more *datapath components* that provide higher-level functionality (viewable at the register transfer level)
  - The *control unit* controls the sequence in which the datapath functions are performed in order to perform the system task



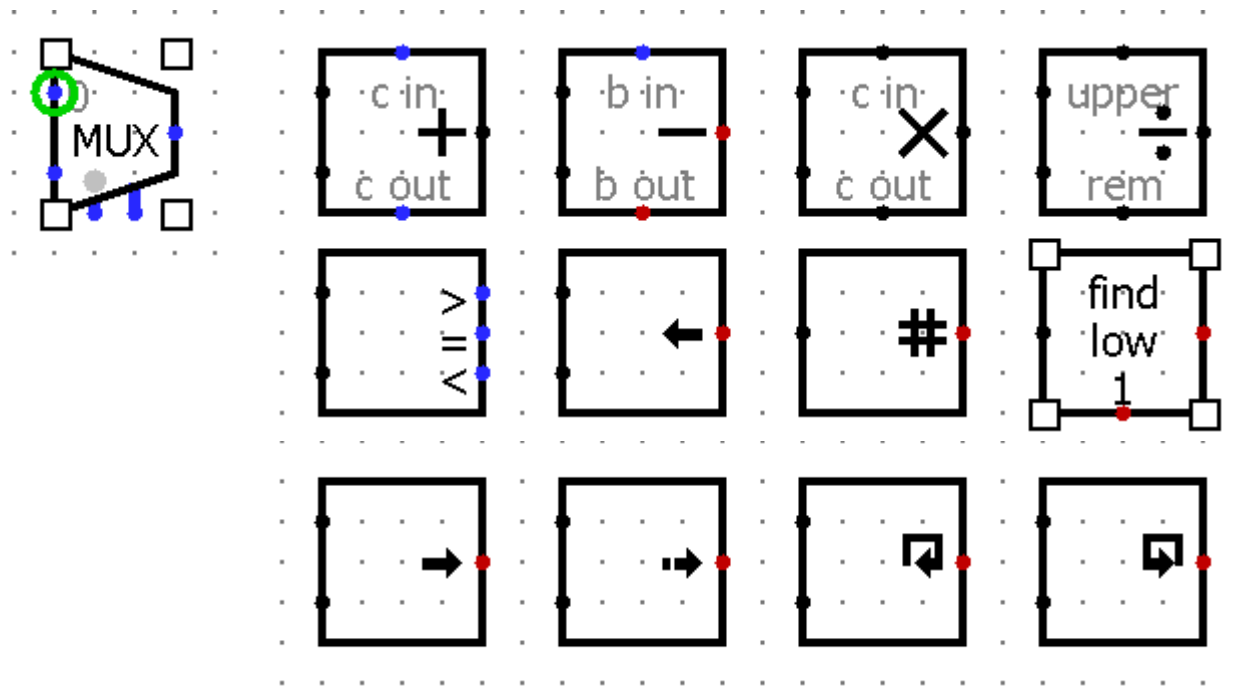
# Datapath components

---

- Gates and flip-flops are good building blocks for simple designs
- We need more sophisticated building blocks for complex systems
- Register-transfer level (RTL) components (aka Datapath components) include medium scale devices such as:
  - Registers
  - ALUs
  - Multiplexers
  - Comparators
  - etc



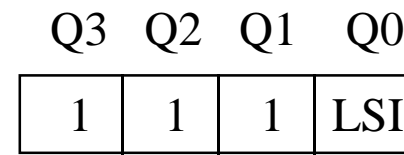
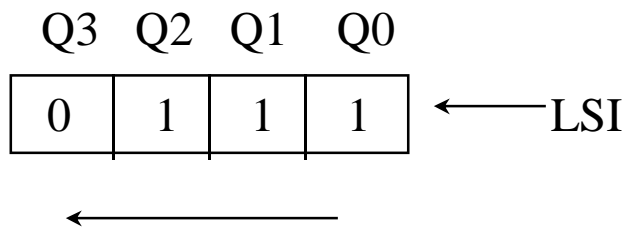
# Some RTL combinational devices



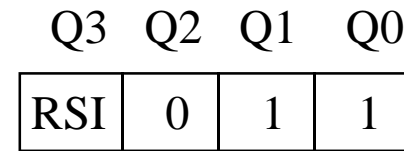
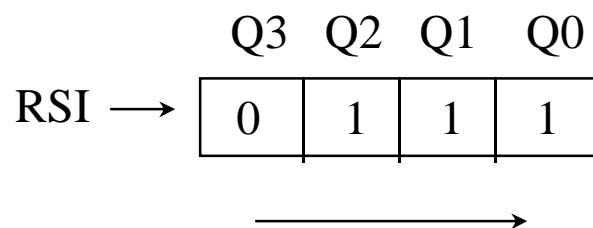


# Shifting

- Moving moves data “sideways” left/right
  - Shift Left (or Shift Down) is towards MSB



- Shift Right (or Shift Up) is towards LSB

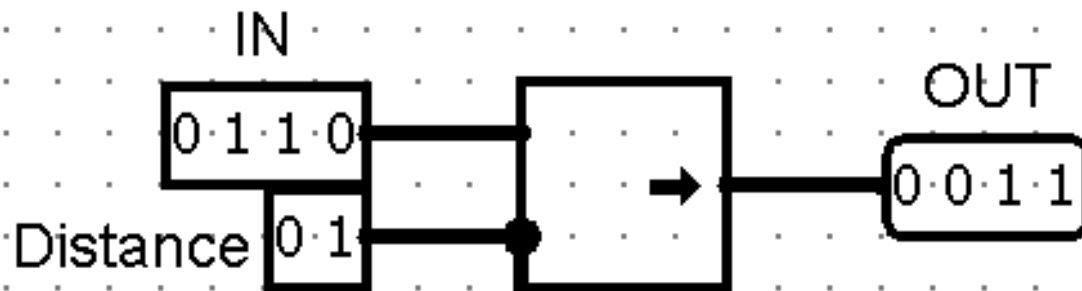
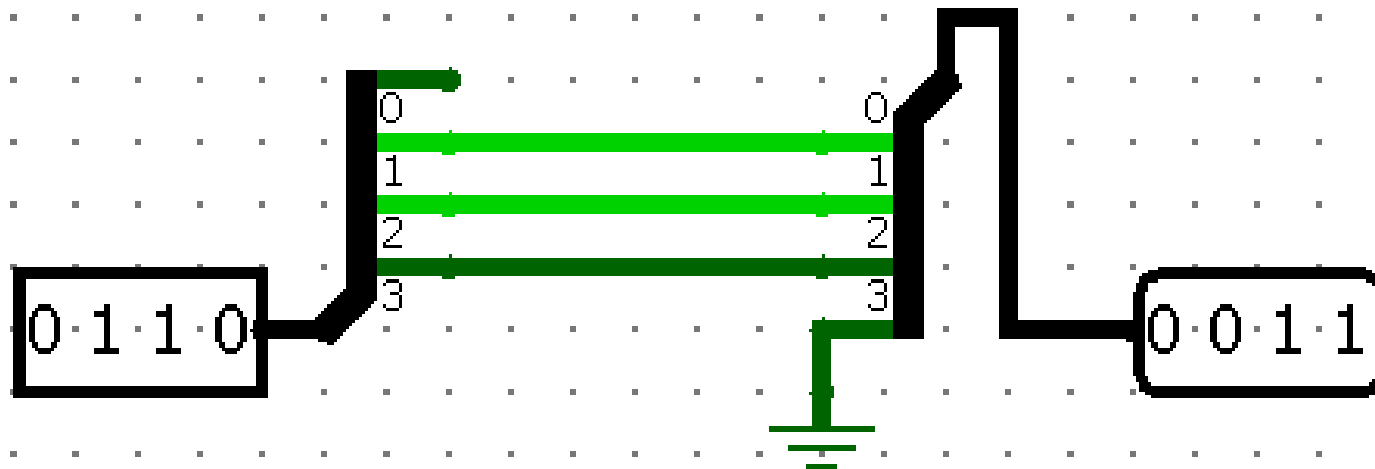


⇒ Often used to rearrange bits or Multiply/Divide by 2

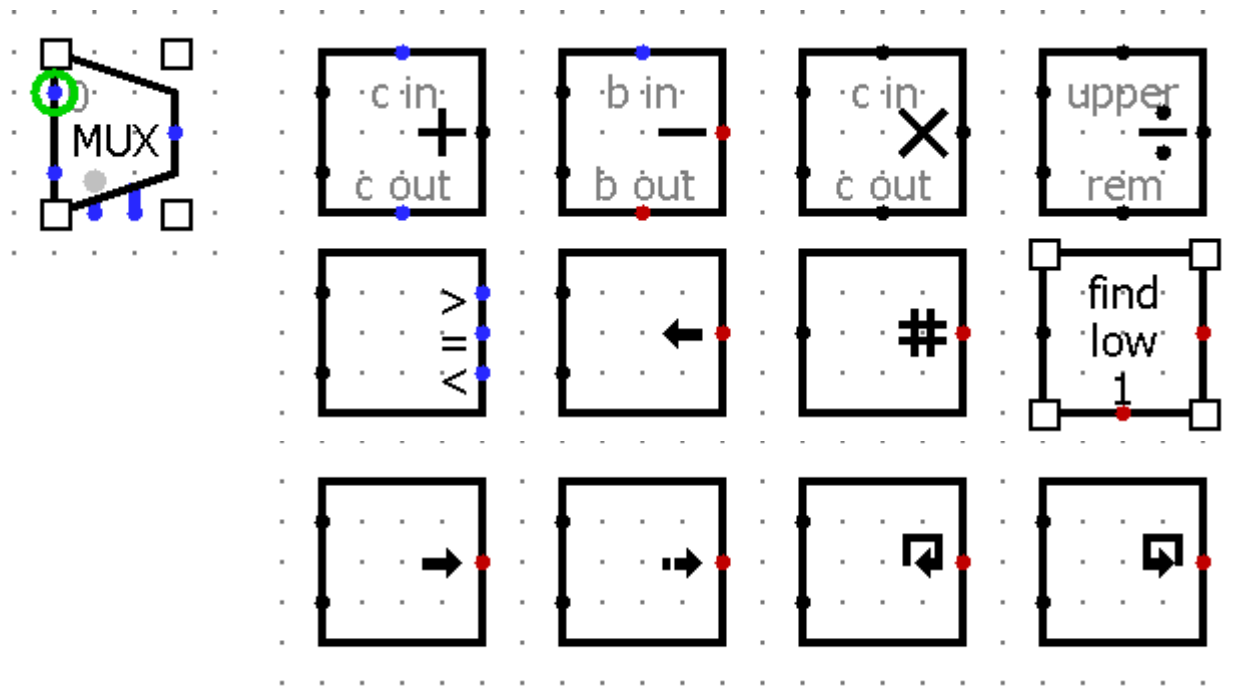


# Combinational Barrel Shifter

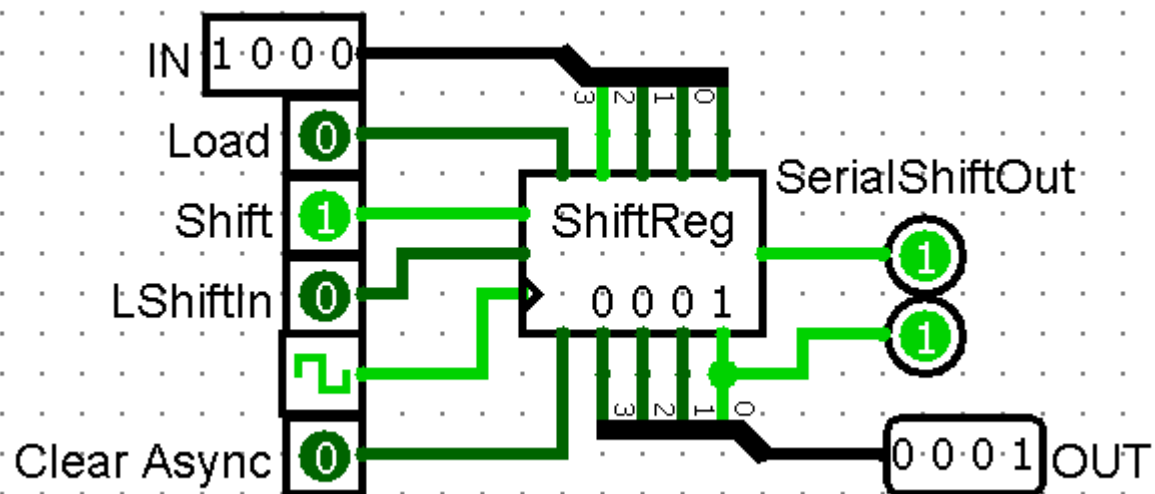
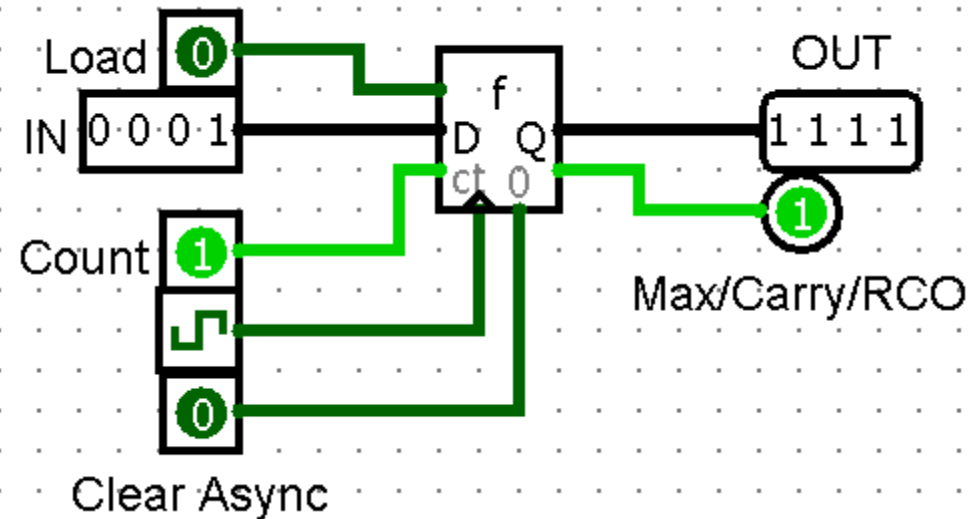
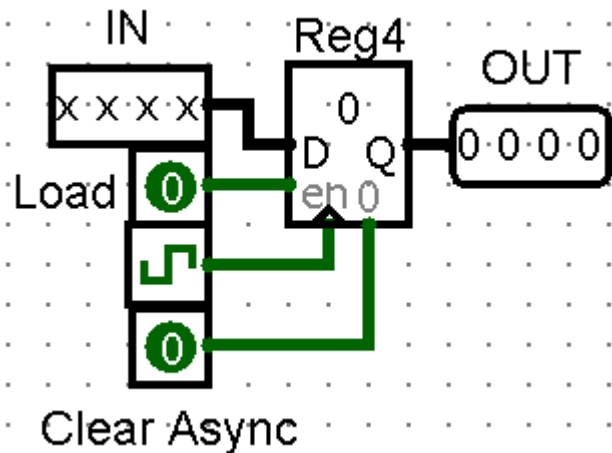
## Shift logical right



# Some RTL combinational devices

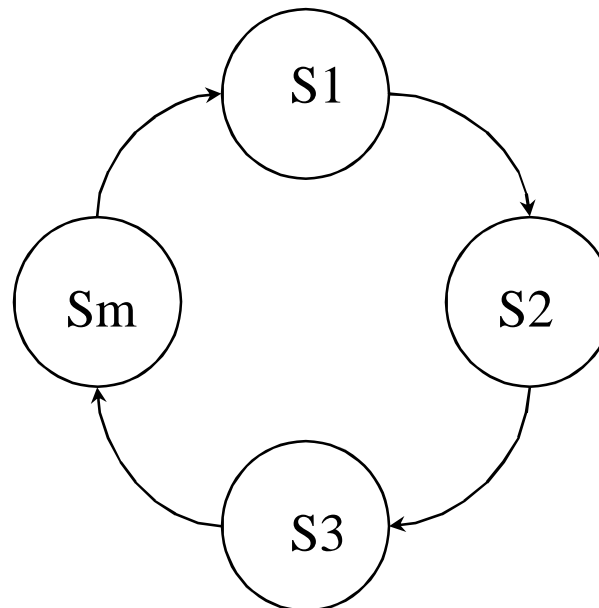


# Some sequential RTL devices



# Counting

- Counters are registers with extra functions
- Clocked sequential circuit with single-cycle state diagram
  - Modulo-m counter = divide-by-m counter
- Most Common:
  - n-bit binary counter, where  $m = 2^n$  ▲  
n flip-flops, counts 0 ...  $2^n - 1$
- Could implement with a register and an adder, but there are optimizations available if only ever adding +/- 1



000
001
010
011
100
101
110
111



# Counting in some common encodings

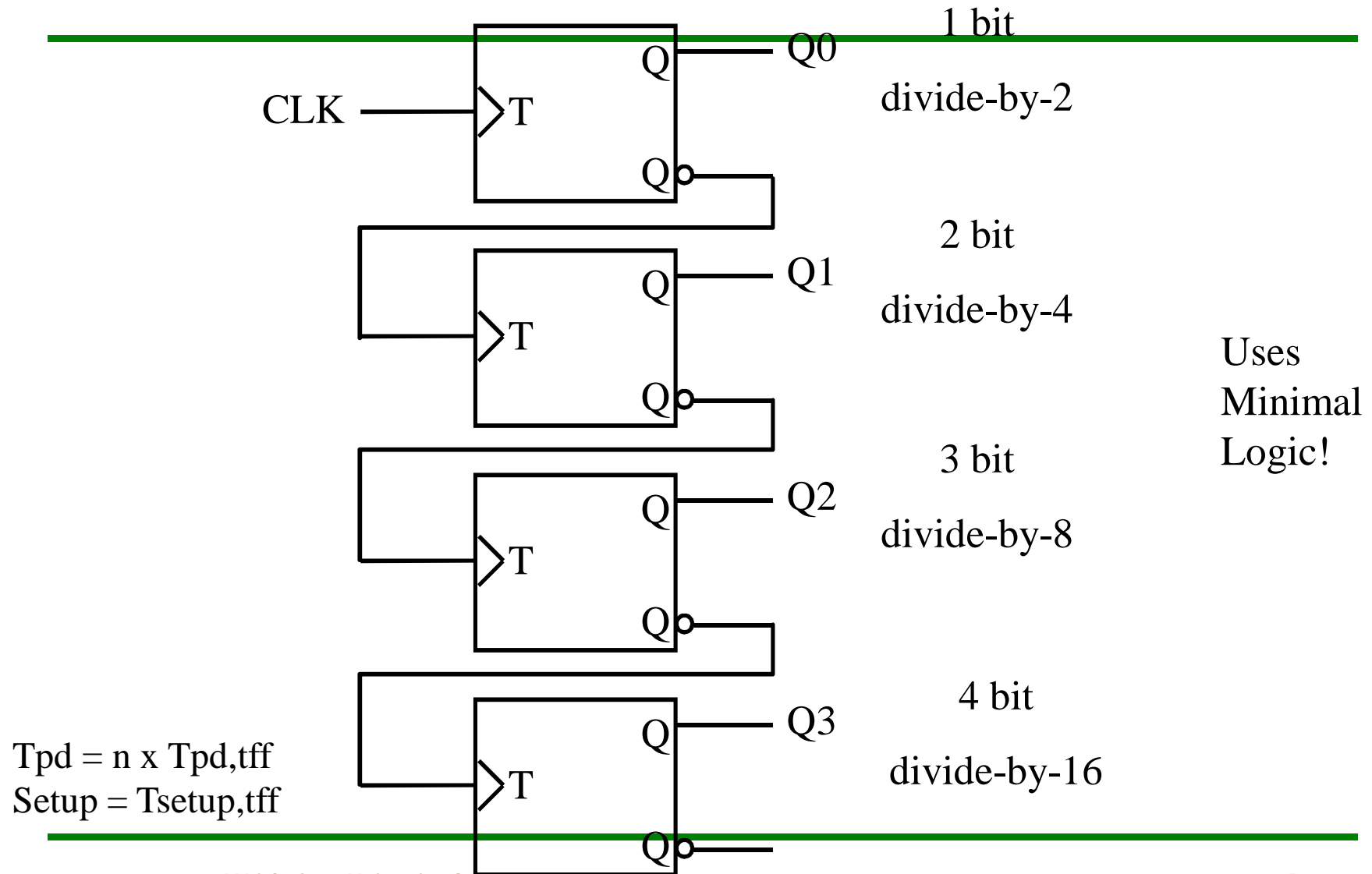
---

- Common output codes for mod-8 and decimal counters

State	Binary	BCD	Gray	Excess-3	Ring	Twisted-tail
0	000	0000	000	0011	00000001	0000
1	001	0001	001	0100	00000010	0001
2	010	0010	011	0101	00000100	0011
3	011	0011	010	0110	00001000	0111
4	100	0100	110	0111	00010000	1111
5	101	0101	111	1000	00100000	1110
6	110	0110	101	1001	01000000	1100
7	111	0111	100	1010	10000000	1000
8		1000		1011		
9		1001		1100		



# Asynchronous/Ripple Counter



# Synchronous Counters

---

- All clock inputs connected to common CLK signal
  - So all flip-flop outputs change simultaneously  $t_{cQ}$  after CLK↑
- Synchronous Counters are/have
  - Faster
  - More Complex Logic (more “expensive”)
  - Most Frequently Used Type of Counter
- Two types of synchronous counters
  - Serial
  - Parallel
- Easy to combine iteratively to build bigger counters
  - Combined counters have serial aspects. If the devices are parallel, then the overall device is “mixed mode”.



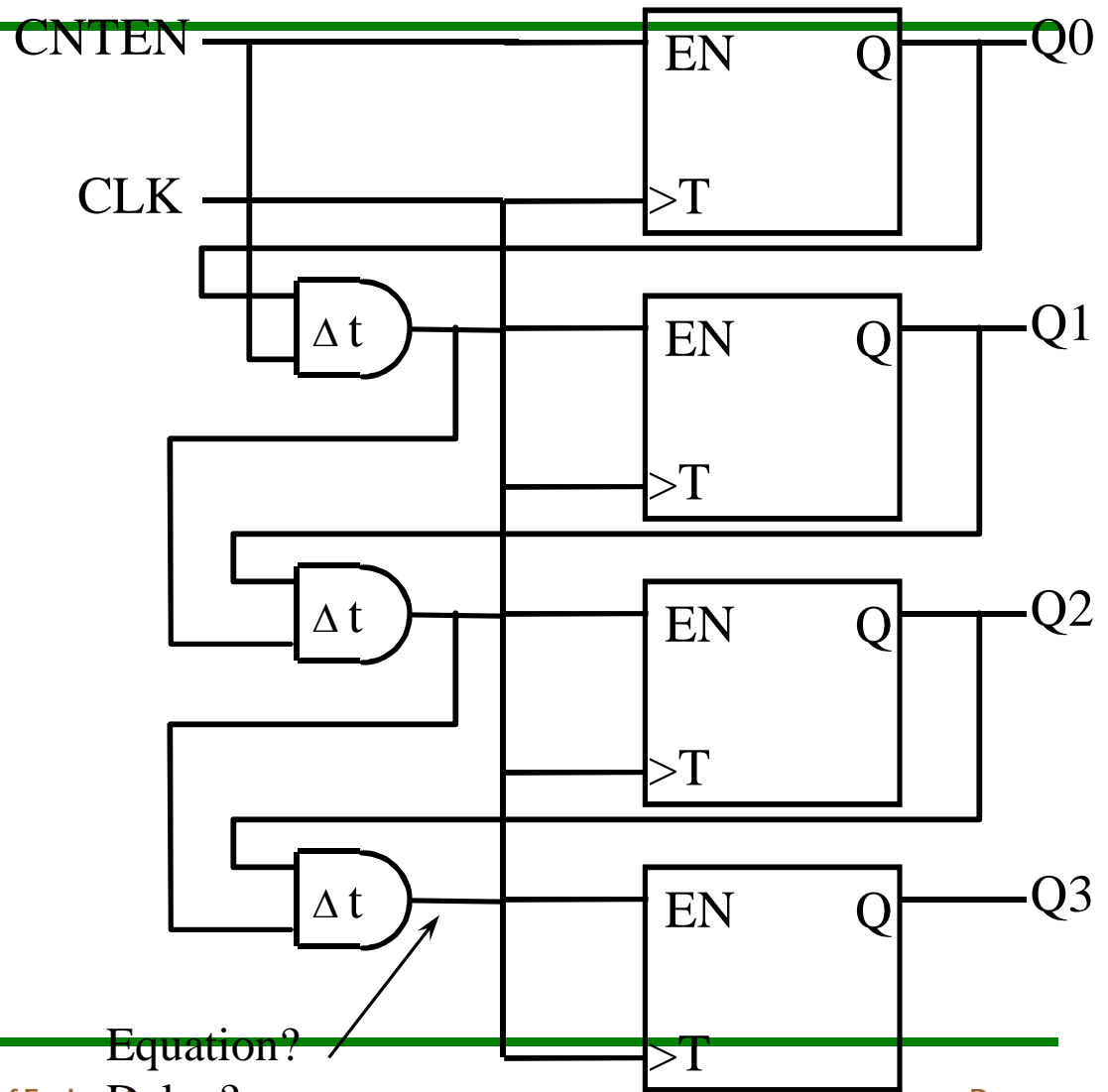


# Synchronous Serial Counter

- Flip-flops enabled when all lower flip-flops = 1.
- Enable propagates serially — limits speed
- Requires  $(n-1) \Delta t < T_{CLK}$
- All outputs change simultaneously  $t_{CQ}$  after CLK ↑

$$T_{pd} = T_{pd,tff}$$

$$T_{\text{setup}} = (n-1)\Delta t + T_{\text{setup,tff}}$$



# Equation?

# Delay?

# Synchronous Parallel Counter

- Single-level enable logic per flip-flop
- Fastest and most complex type of counter
- Requires  $\Delta t < T_{CLK}$
- All outputs change simultaneously  $t_{CQ}$  after CLK

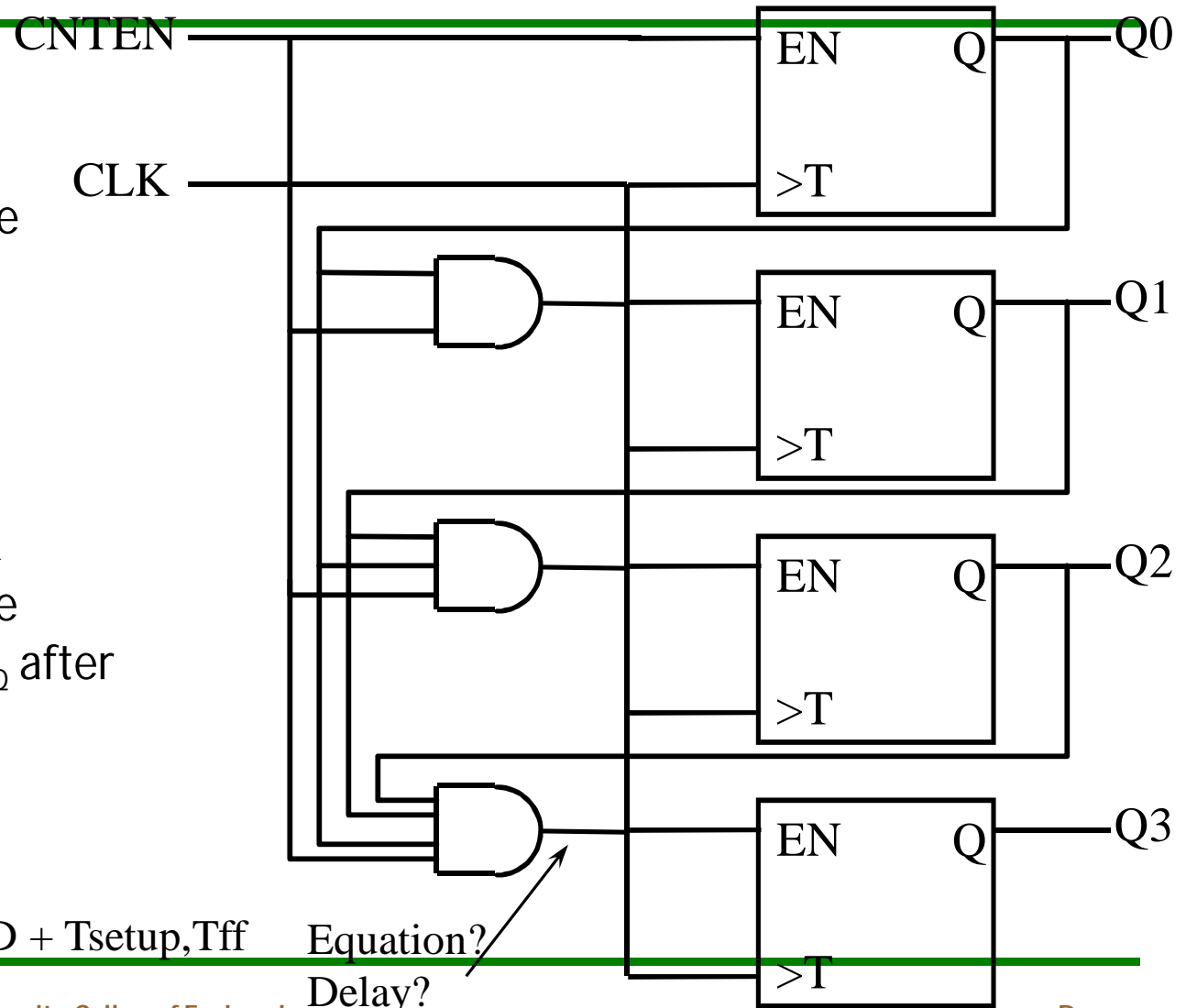


$T_{pd} = T_{pd,tff}$

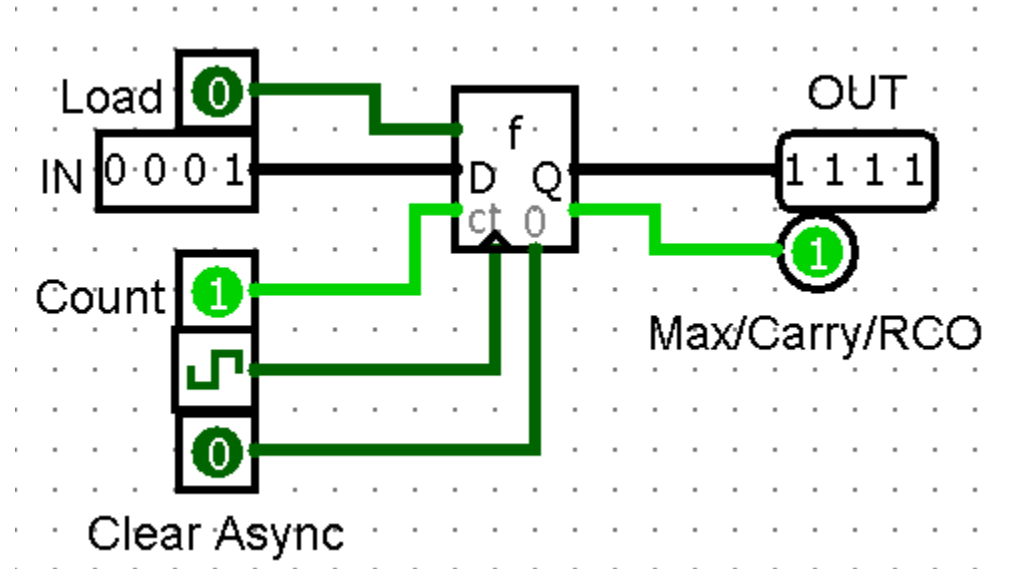
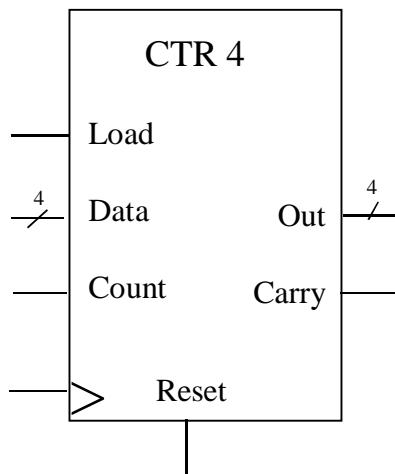
$T_{setup} = T_{pd,biggestAND} + T_{setup,Tff}$

Equation?

Delay?



# A Standard Counter (Logisim)



**Load:** when 1, load from D (if  $ct = 0$ ) or decrement (if  $ct = 1$ )

**Data:** value to load into counter

**Count:** when 1, counter increments (or decrements if  $load = 1$ )

**Reset:** when 1, resets to 0 asynchronously

**Output:** current value of the counter

**Carry:** is 1 when counter reaches maximum (minimum when decrementing)



# Using a counter

---



# Register Transfer Level Design Decomposition



Design Decomposition

# Design Decomposition

---

- A digital system is a sequential circuit with specified behavior.
  - A microprocessor is a digital system.
- Specifying large digital systems with state tables may be exceptionally difficult, due to the number of states involved.
  - As in computer programming, most digital systems are designed using a modular, hierarchical approach.
  - The system is partitioned into modular subsystems.
    - Each subsystem performs a well defined function with specified interface.
  - Interconnection the various subsystems though data and control signals results in a digital system.



# Design Decomposition

- 
- Most digital systems are partitioned into two top-level modules:
    - Data Unit (or Datapath): performs data-processing operations.
    - Control Unit: determines the sequence of these operations.
  - Datapaths are sequential systems.
    - the system state is defined by the contents of the registers.
    - the functionality is the set of defined operations that can be performed on the contents of the registers.
    - Elementary operations are usually, but not always, performed in parallel on a string of bits in one clock tick.
  - A microoperation is an elementary operation performed on data stored in the datapath. They fall into four general categories:
    - **Transfer** microoperations: transfer binary data from one register (or data input/memory) to another.
    - **Arithmetic** microoperations: perform arithmetic on data in registers.
    - **Logic** microoperations: perform bit manipulations on data in registers.
    - **Shift** microoperations: shift data in registers.
- 



# Register-Transfer Level Design

---

- An approach to specify, analyze, and design systems too complex to use the state-table based approaches commonly utilized in “simple” designs.
- The Register-Transfer Level (RTL) approach is characterized by:
  - A digital system is viewed as divided into a data subsystem and a control subsystem.
  - The state of the data subsystem consists of the contents of the registers.
  - The function of the system is performed as a sequence of register transfers.
  - A *register transfer* is a transformation performed on the datum while the datum is transferred from one register to another.
  - The sequence of register transfers is controlled by the control subsystem.
- The operation of the device can be designed as a sequence of register transfers can be designed using state diagrams, ASM charts, etc.
  - Each transfer must correspond to a sequence of microoperations.
  - The control unit implements the RTL design through microoperations.





# RTL Languages (1)

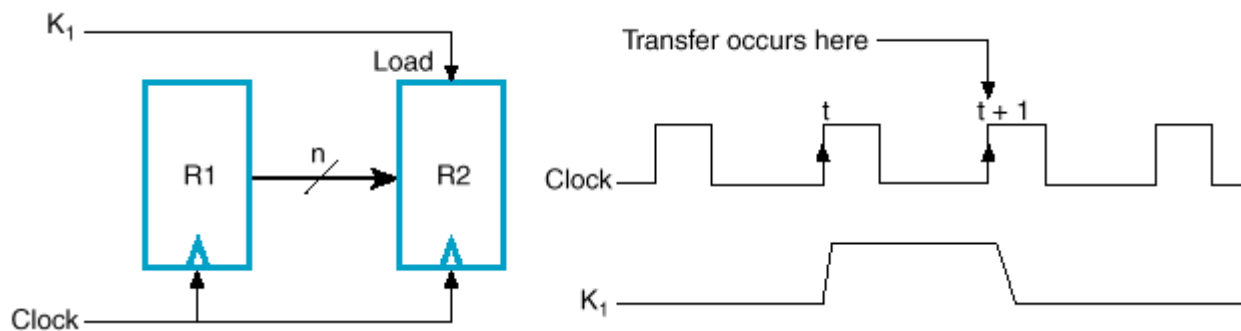
---

- The notation for register transfers are sufficiently complete to describe any digital system at the register-transfer level.
    - known as register-transfer languages.
  - Registers are denoted by uppercase letters (sometimes followed by numbers) that indicate the function of the register
    - e.g. R0, R1, AR, PC, MAR, et al.
    - The individual bits can be denoted using parenthesis and bit numbers or labels
      - e.g. R0(0), R0(7:0), PC(L), PC(H)
  - Data transfer is denoted in symbolic form by the means of the **replacement operator**  $\leftarrow$ .
    - e.g.  $R2 \leftarrow R1$
- 



# RTL Languages (2)

- Normally we want a given transfer to occur not for every clock pulse, but only for specific values of the control signals.
  - RTL conditional statements:
    - e.g. If ( $K_1 = 1$ ) Then ( $R_2 \leftarrow R_1$ )
  - Control function notation (Colon, :)
    - e.g.  $K_1: R_2 \leftarrow R_1$



- All RTL statements occur in response to a clock tick. A comma is used to separate two or more register transfers that are executed at the same time. A semi-colon is used for an instruction with different control
  - e.g. Brake:  $R_2 \leftarrow R_1, R_4 \leftarrow R_3$ ; not(Brake):  $R_1 \leftarrow R_2$



# RTL Languages (3)

- Register to Memory Transfers are denoted using square brackets surrounding the memory address.
  - e.g.  $DR \leftarrow M[AR]$  (Read operation)
  - e.g.  $M[AR] \leftarrow SR$  (Write operation)

Symbol	Description	Examples
Letters (and numerals)	Denotes a register	$AR, R2, DR, IR$
Parentheses	Denotes a part of a register	$R2(1), R2(7:0), AR(L)$
Arrow	Denotes transfer of data	$R1 \leftarrow R2$
Comma	Separates simultaneous transfers	$R1 \leftarrow R2, R2 \leftarrow R1$
Square brackets	Specifies an address for memory	$DR \leftarrow M[AR]$



# RTL Languages (4)

## Examples of Arithmetic Microoperations

Symbolic designation	Description
$R0 \leftarrow R1 + R2$	Contents of $R1$ plus $R2$ transferred to $R0$
$R2 \leftarrow \overline{R2}$	Complement of the contents of $R2$ (1's complement)
$R2 \leftarrow \overline{R2} + 1$	2's complement of the contents of $R2$
$R0 \leftarrow R1 + \overline{R2} + 1$	$R1$ plus 2's complement of $R2$ transferred to $R0$ (subtraction)
$R1 \leftarrow R1 + 1$	Increment the contents of $R1$ (count up)
$R1 \leftarrow R1 - 1$	Decrement the contents of $R1$ (count down)

Symbolic designation	Description
$R0 \leftarrow \overline{R1}$	Logical bitwise NOT (1's complement)
$R0 \leftarrow R1 \wedge R2$	Logical bitwise AND (clears bits)
$R0 \leftarrow R1 \vee R2$	Logical bitwise OR (sets bits)
$R0 \leftarrow R1 \oplus R2$	Logical bitwise XOR (complements bits)

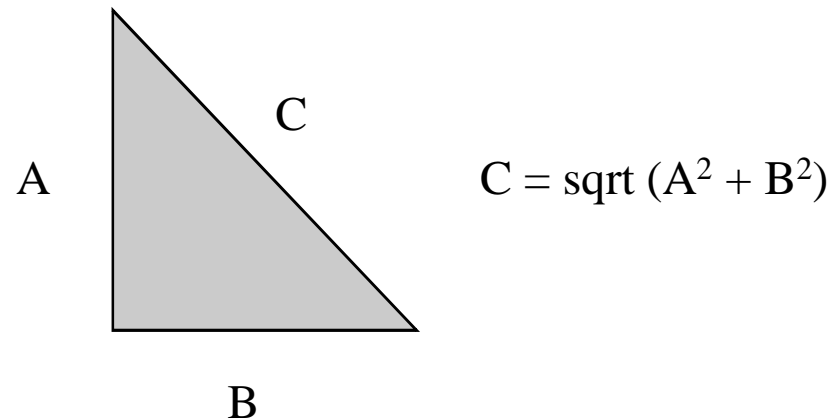
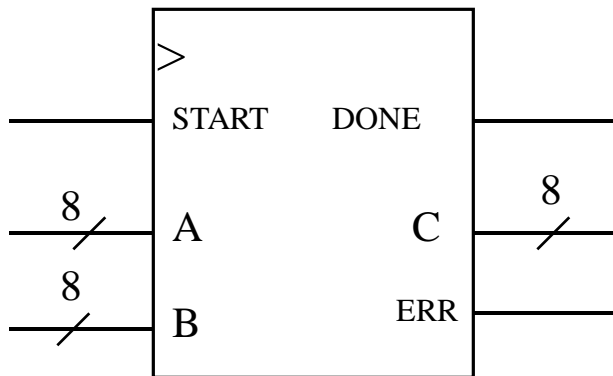
## Examples of Logic Microoperations





# Designing a datapath

- Design a device with two 8-bit inputs A and B, one 1-bit input START, one 8-bit output C, and one 1-bit output DONE. The device begins idle (with output DONE = 0). When START is asserted (for one clock tick) the unsigned binary inputs A and B are and held constant until the device asserts "DONE". The device must calculate the approximate length of the hypotenuse of a right triangle with sides A and B. When the final answer is available on output C, the device will assert DONE for one clock tick. If the answer cannot be computed, assert ERR.









# Designing a datapath

---

- Euler formula for Square Root Approximation:
  - Let  $x = \max(|a|, |b|)$
  - Let  $y = \min(|a|, |b|)$
  - $\text{Sqrt}(a^2 + b^2) \approx \max(x, (0.875x + 0.5y))$
- What sort of functions do you need to process the data?



# Designing a datapath

- SRA Circuit Model      Let  $x = \max(a, b)$  and  $y = \min(a, b)$ 
  - $\text{Sqrt}(a^2 + b^2) \sim \max(x, (0.875x + 0.5y))$

## Control unit

...

$R_x \leftarrow \max(A, B)$

$R_y \leftarrow \min(A, B)$

$R_y \leftarrow R_y \gg 1$  # shift right 1;  $\cdot 0.5$

$R_t \leftarrow R_x \gg 3$  # shift right 3 ( $1/8^{\text{th}}$ )

$R_t \leftarrow R_x - R_t$  #  $x - 1/8x = x \cdot 0.875$

$R_t \leftarrow R_y + R_t$

$R_c \leftarrow \max(R_x, R_t)$

$C \leftarrow R_c$

...

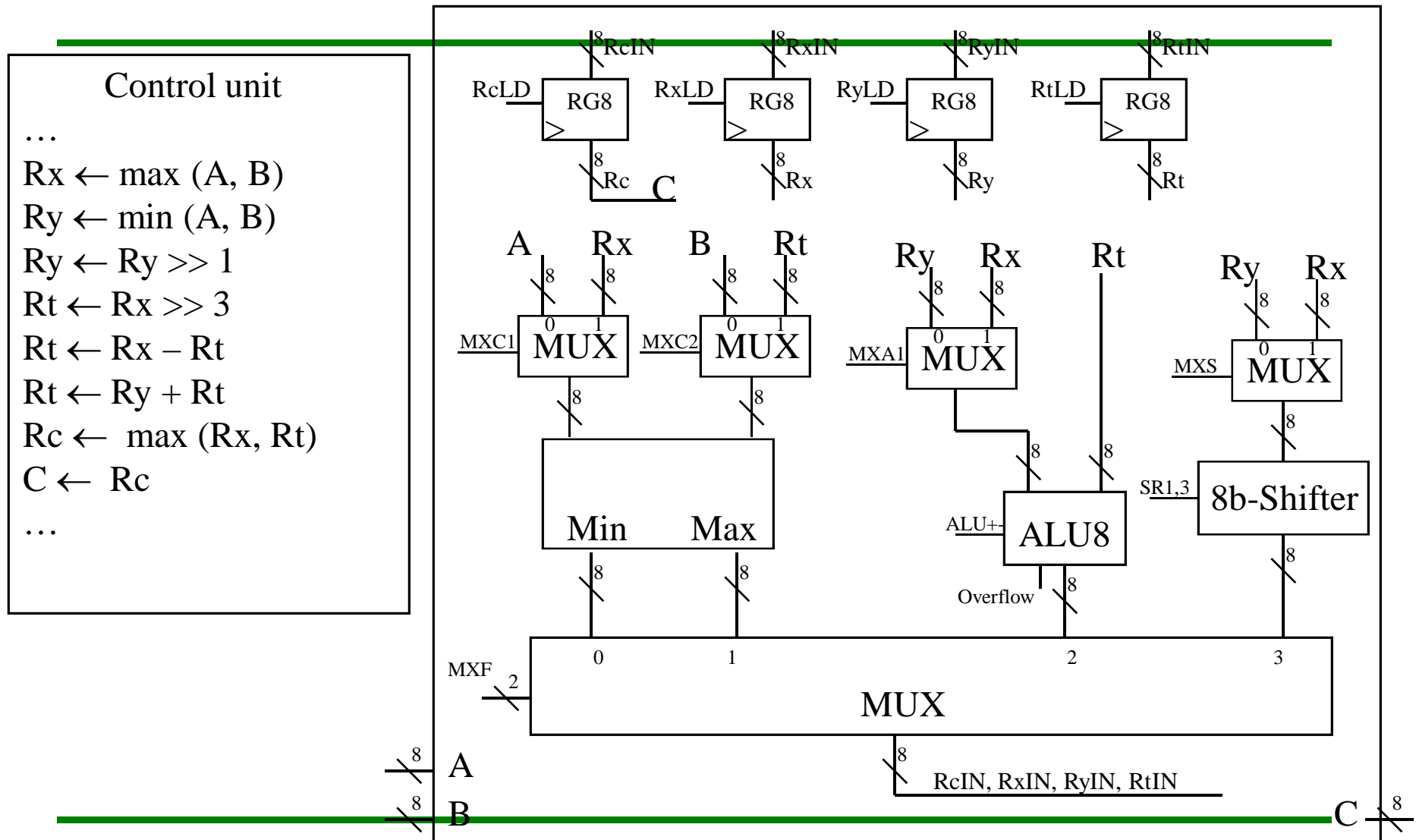
## Datapath

Registers: c, x, y, t

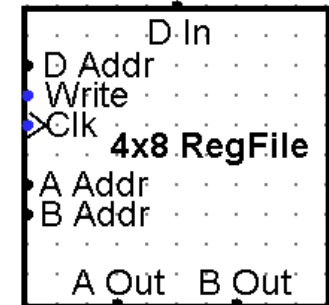
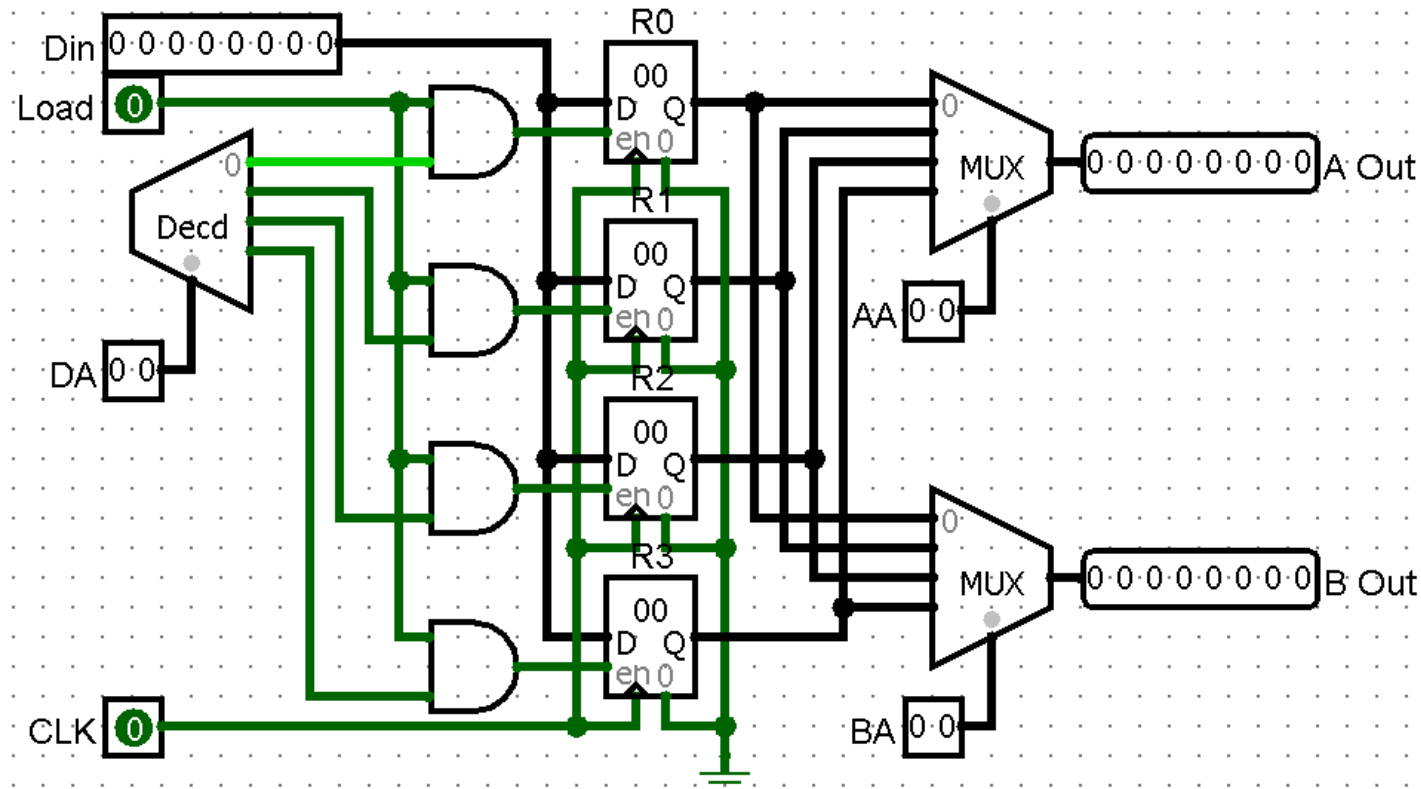
Functions: min, max, +, -, shift



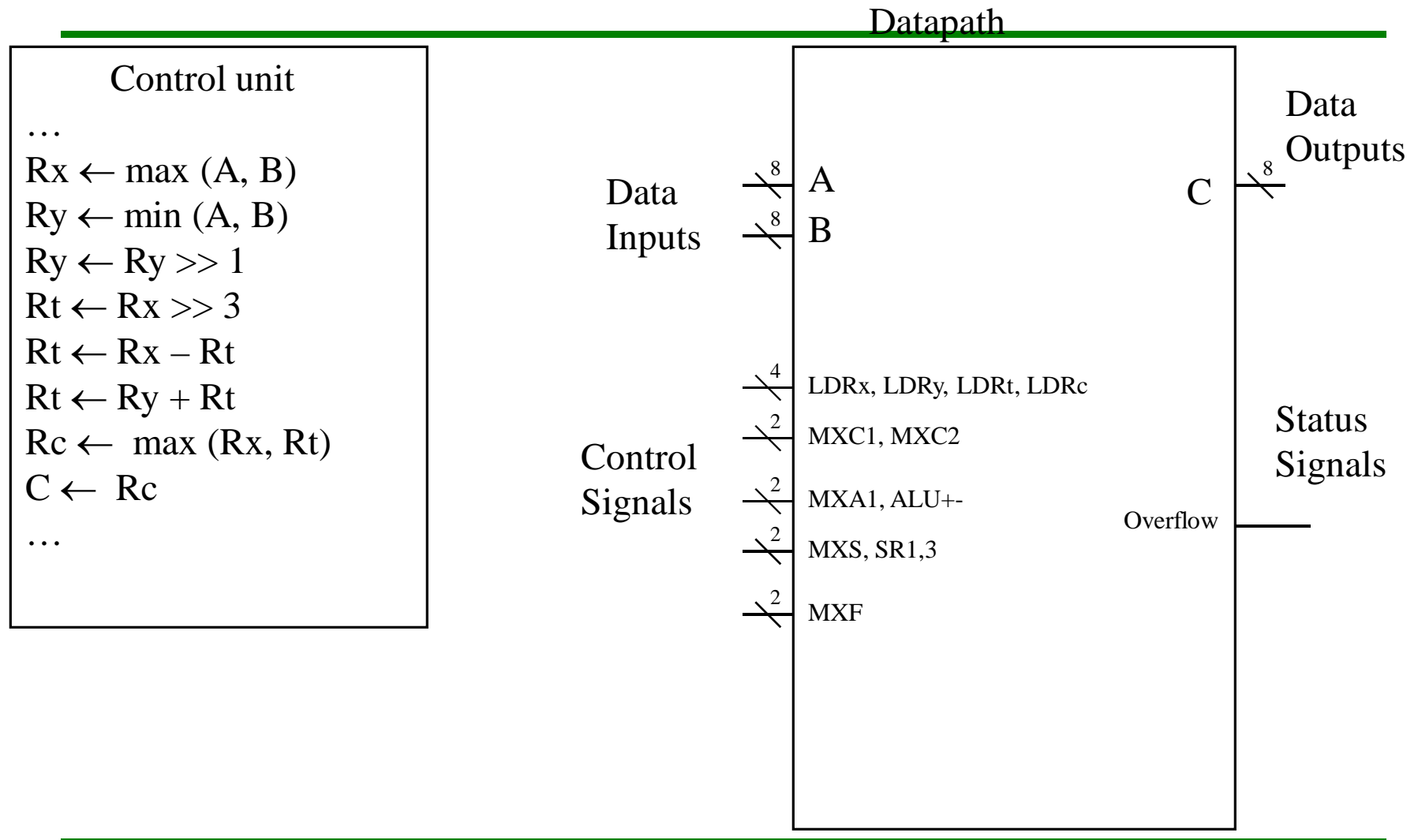
# Designing a datapath: Can we do better?



# Register File



# Designing a datapath

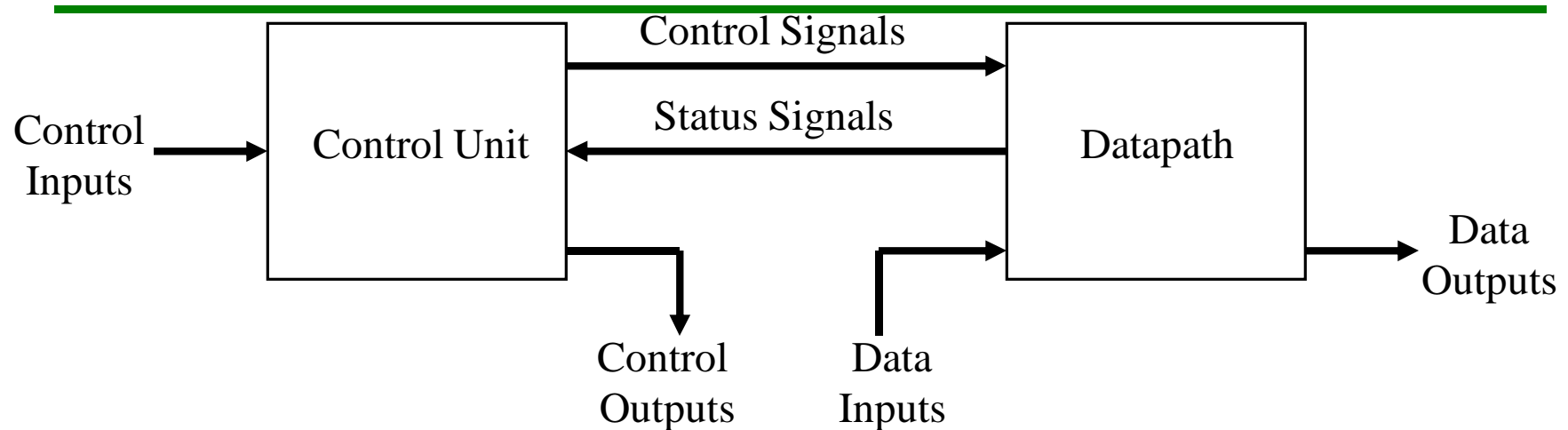


# Register Transfer Level Design Decomposition



Control unit design

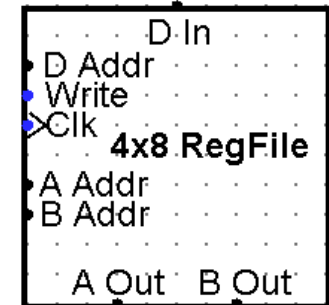
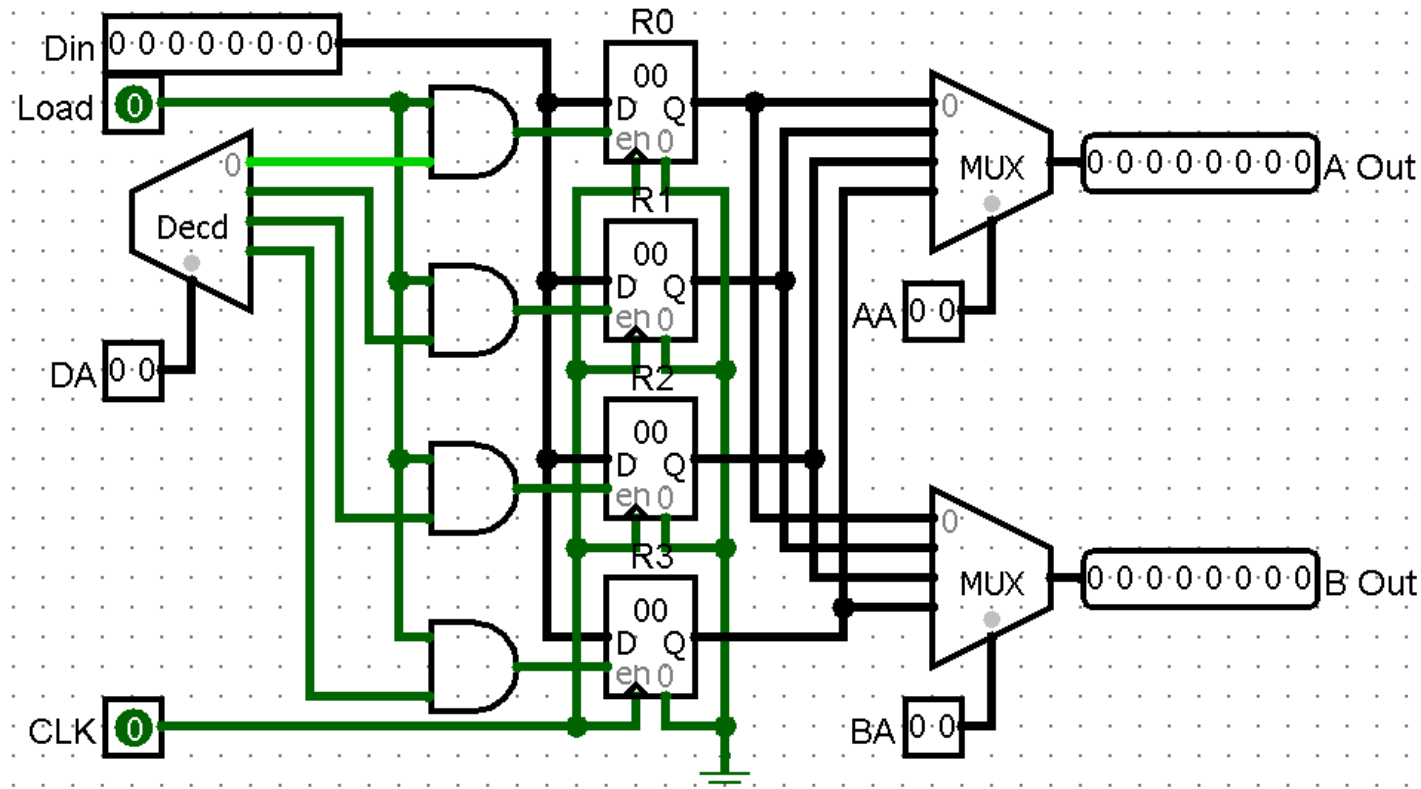
# Interaction between Data and Control Units



- Control Signals - signals that activate data-processing functions.
  - To activate a sequence of such operations, the control unit sends the proper sequence of control signals to the datapath.
- Status Signals - signals that describe aspects of the state of the datapath.
  - The control unit uses these signals in determining the specific sequence of operations to be performed.
- Other Signals - allow the control unit and datapath to interact with other parts of the system, such as memory and input-output logic.



# Register File





# The Control Unit

---

- The control unit generates the signals for sequencing the operations in the datapath
  - A sequential circuit with states that *dictate the control signals* for the system
  - Using status conditions and control inputs, the sequential control unit *determines the next state* in which additional microoperations are activated.
- Hardwired Control
  - The control unit is implemented to provide a particular digital function
- Microprogrammed Control
  - sequencer

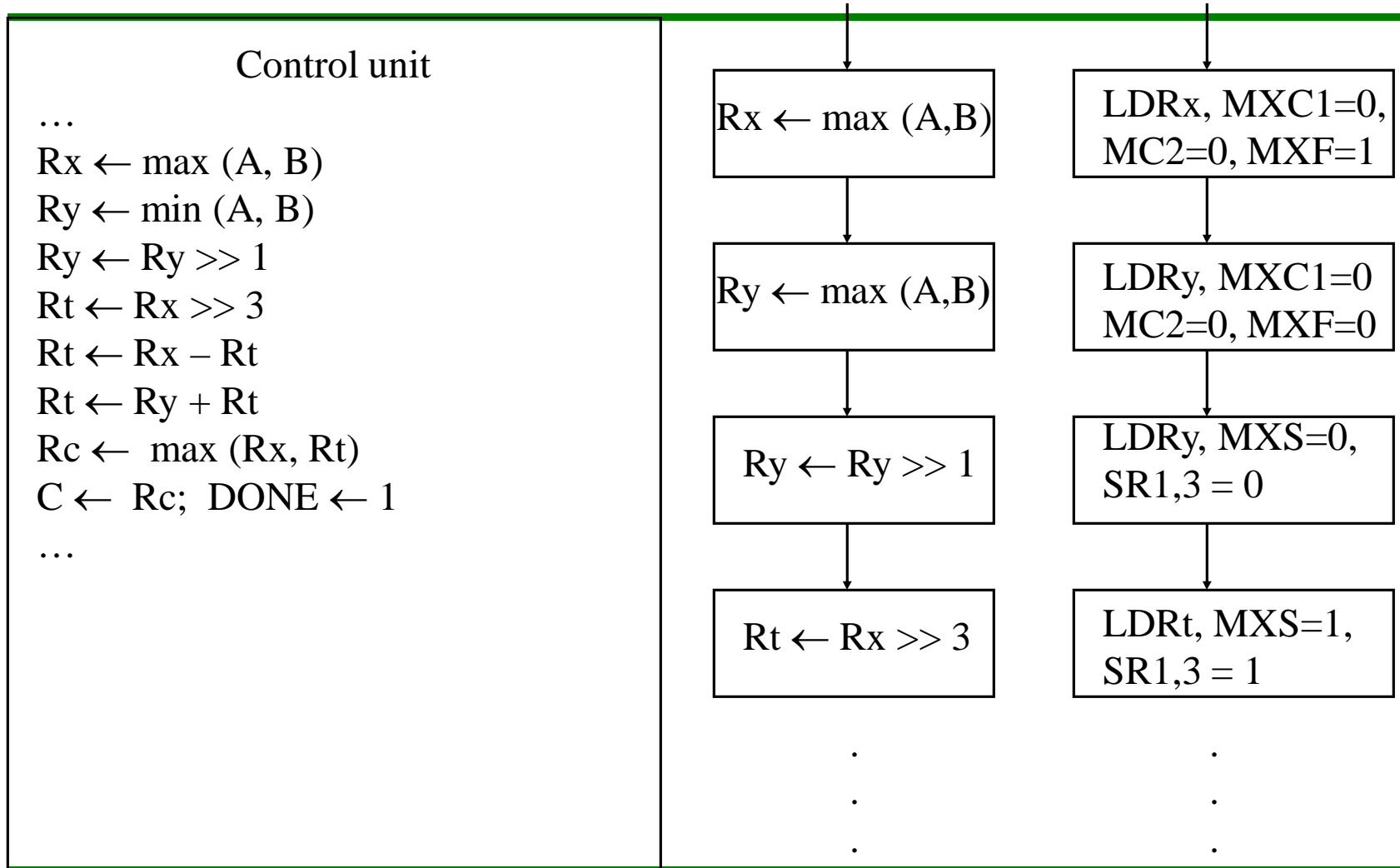


# Control Unit Design

Control unit						LDx ...
	Next State					
	StartOverflow =					
	00	01	10	11	RTL	
PS	00	01	10	11	RTL	
IDLE	IDLE	IDLE	S1	S1		
S1	S2	S2	S2	S2	$R_x \leftarrow \max(A, B)$	
S2	S3	S3	S3	S3	$R_y \leftarrow \min(A, B)$	
S3	S4	S4	S4	S4	$R_y \leftarrow R_y \gg 1$	
S4	S5	S5	S5	S5	$R_t \leftarrow R_x \gg 3$	
S5	S6	S6	S6	S6	$R_t \leftarrow R_x - R_t$	
S6	S7	ERR	S7	ERR	$R_t \leftarrow R_y + R_t$	
S7	S8	S8	S8	s8	$R_c \leftarrow \max(R_x, R_t)$	
S8	IDLE	IDLE	IDLE	IDLE	$C \leftarrow R_c; \text{Done} \leftarrow 1$	
ERR	IDLE	IDLE	IDLE	IDLE	$\text{ERR} \leftarrow 1$	Done



# ASM for Control



# The Control Unit

- The control unit generates the signals for sequencing the operations in the datapath
  - A sequential circuit with states that *dictate the control signals* for the system
  - Using status conditions and control inputs, the sequential control unit *determines the next state* in which additional microoperations are activated.
- Hardwired Control
  - The control unit is implemented to provide a particular digital function
- Microprogrammed Control
  - The control unit's binary control values are stores as words in a microprogrammed control (usually ROM).
  - Each word in the control contains a microinstruction
  - A sequence of microinstructions constitutes a microprogram
  - Firmware!





# ROM and PLDs

Fabrication

Programmable Logic Devices

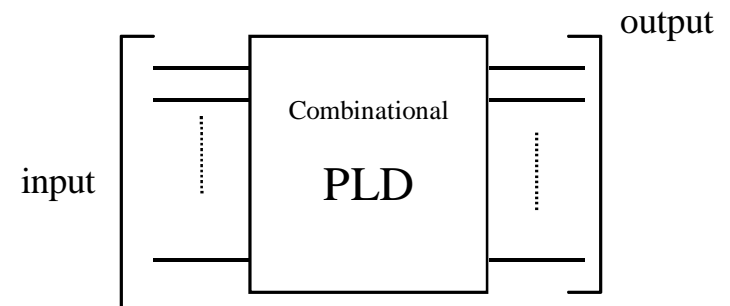
Read Only Memory

Iterative combinational implementation

# Programmable Logic Definitions

---

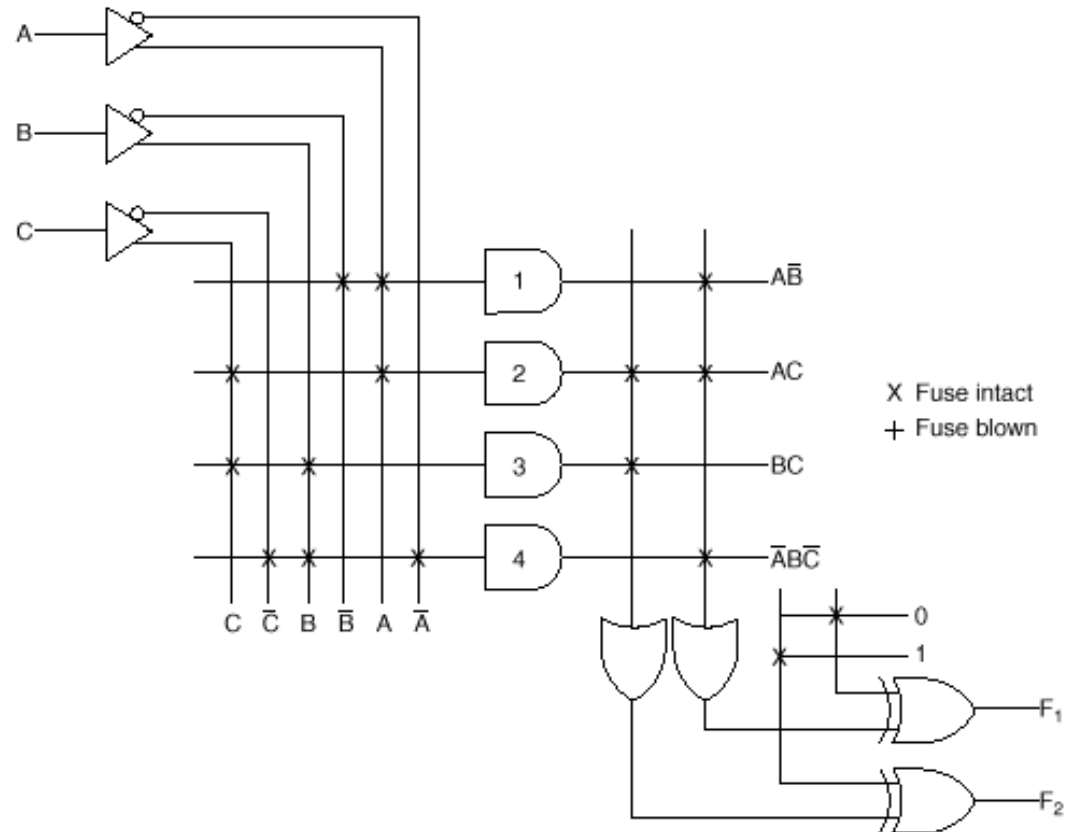
- Digital integrated circuit (MSI, LSI, VLSI)
  - manufactured as a standard off-the-shelf component
  - containing “regular” array of logic gates and flip-flops
  - whose logic functions are determined by the application design engineer and implemented locally
- Many types of programmable logic
  - sometimes generically called PLDs (Programmable Logic Devices)
  - PAL or PLD
  - PROM
  - FPGA
- How are ICs fabricated?



# Programmable Logic Definitions

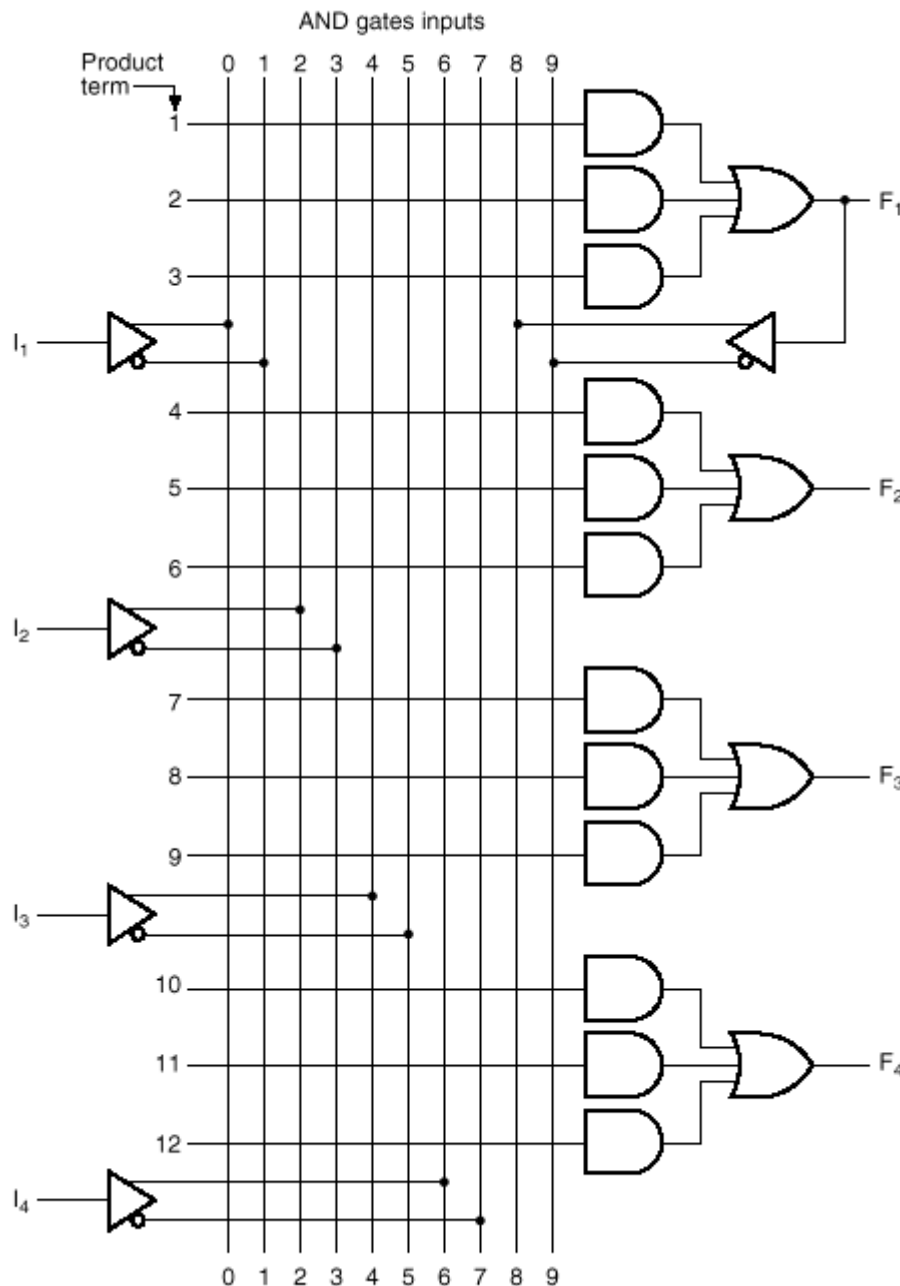
- PLA

- Programmable Logic Array
- first PLDs
- simple programmable AND/OR array
- programmed by “blowing fuses” by hand or by mask





# Programmable Logic



- PAL

- Programmable Array Logic
- similar to a PLA, but the OR-array is fixed
- most commonly used PLD
- may include input/output flip-flops

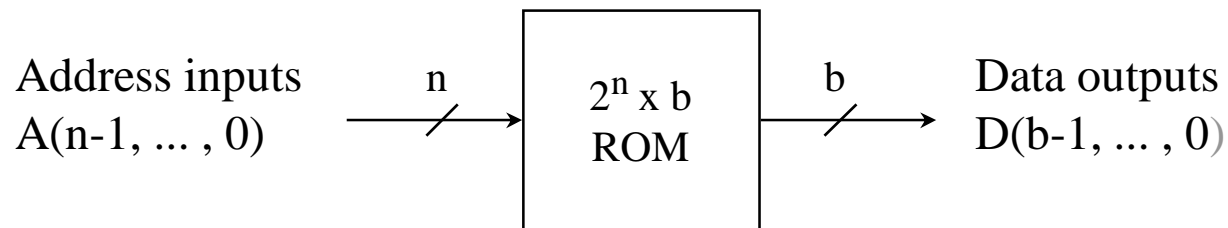
- GAL

- Generic Array Logic
- can be configured to emulate the AND/OR, flip-flop, and output structure of a variety of combinational and sequential PAL devices

# Read-Only Memory (ROM)

---

- A combinational circuit with  $n$  inputs and  $b$  outputs:



- Programmable — values determined by user
- Nonvolatile — contents retained without power
- Uniform (Random) Access — delay is uniform for all addresses



# Read-Only Memory (ROM)

---

- Multiple “views”:
  - ROM stores  $2^n$  words of  $b$  bits each, or
  - ROM implements  $b$  functions in  $n$  variables, or
  - ROM stores an  $n$ -input,  $b$ -output truth table

Example:

$n = 2$		$b = 4$			
A1	A0	D3	D2	D1	D0
0	0	0	1	0	1
0	1	1	1	1	1
1	0	0	0	0	1
1	1	1	0	0	0

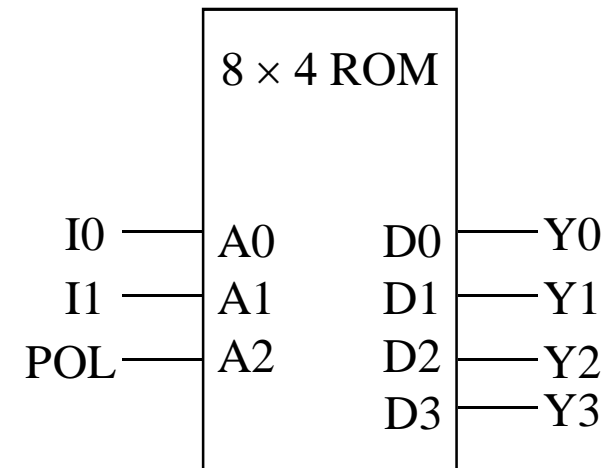
Address	Data
x0	x5
x1	xF
x2	x1
x3	x8



# Using ROMs for Combinational Logic

A 3-input, 4-output combinational logic function:

Inputs			Outputs			
A2	A1	A0	D3	D2	D1	D0
0	0	0	1	1	1	0
0	0	1	1	1	0	1
0	1	0	1	0	1	1
0	1	1	0	1	1	1
1	0	0	0	0	0	1
1	0	1	0	0	1	0
1	1	0	0	1	0	0
1	1	1	1	0	0	0



Function: 2-to-4 Decoder with Polarity Control

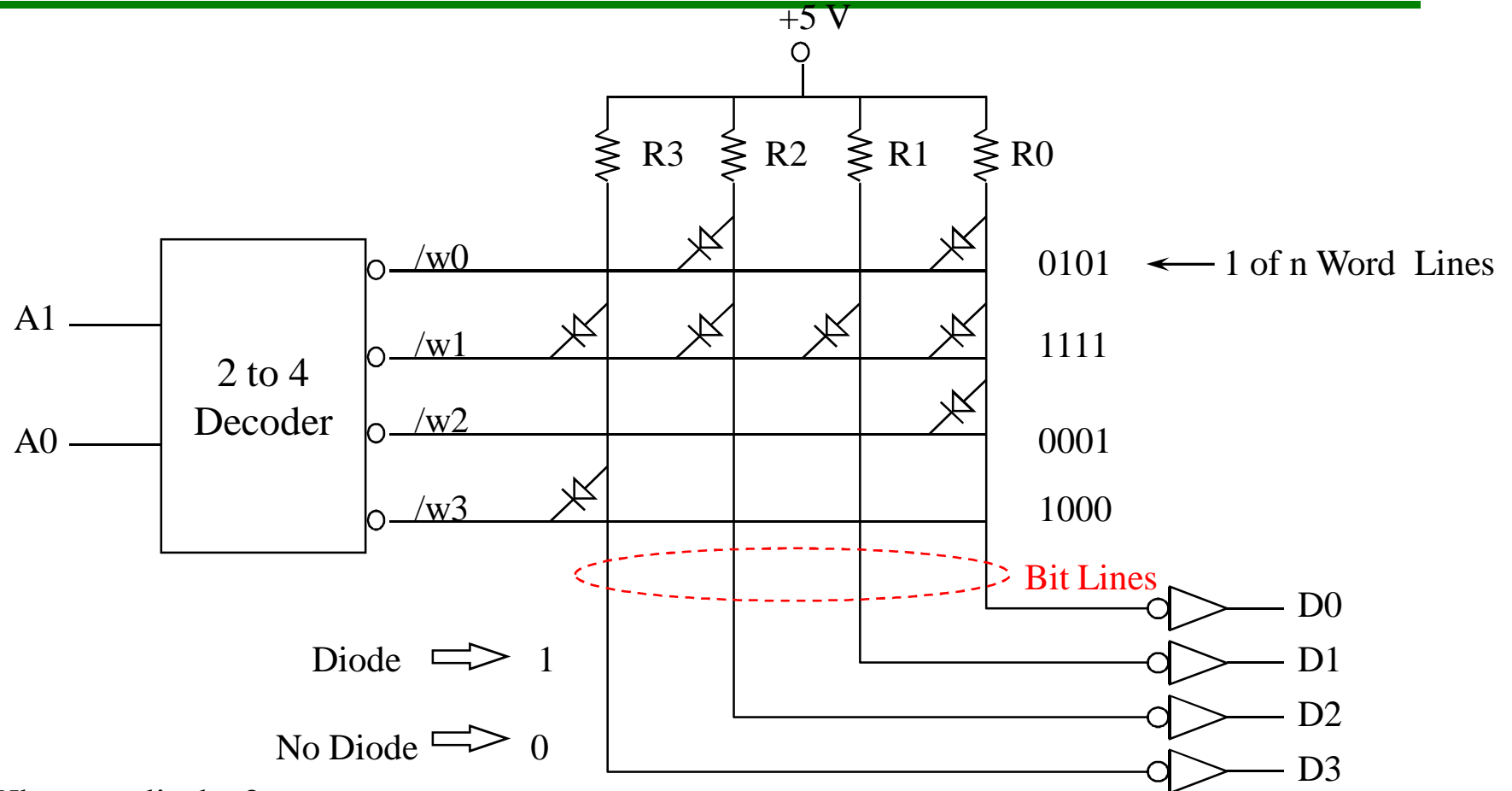
A2 = Polarity (0 = active Low, 1 = active High)

A1, A0 = I1, I0 (2-bit input)

D3...D0 = Y3...Y0 (4-bit decoded output)



# Internal Structure of 4×4 Diode ROM



Why use diodes?

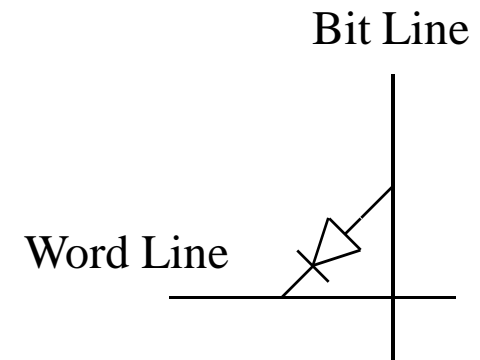
Why not replace them with wires?



# Types Of ROMs (1)

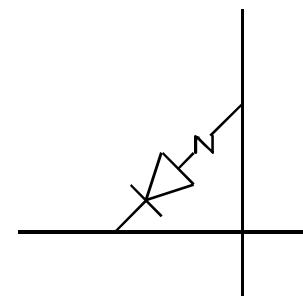
- Mask ROM

- Connections made by the semiconductor vendor
- Expensive setup cost
- Several weeks for delivery
- High volume only
- Bipolar or MOS technology



- PROM

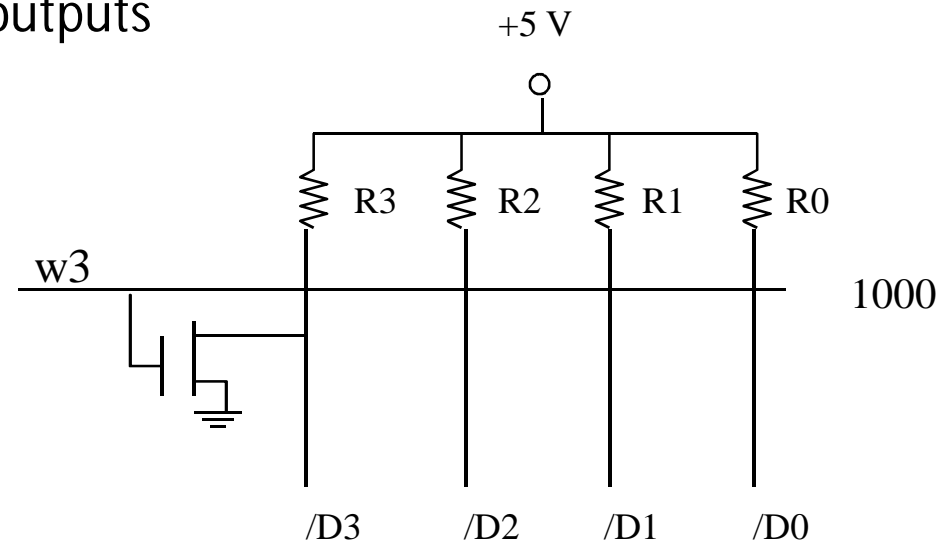
- Programmable ROM
- Connections made by equipment manufacturer
- Vaporize (blow) fusible links with PROM programmer using high voltage/current pulses
- Bipolar technology
- One-time programmable



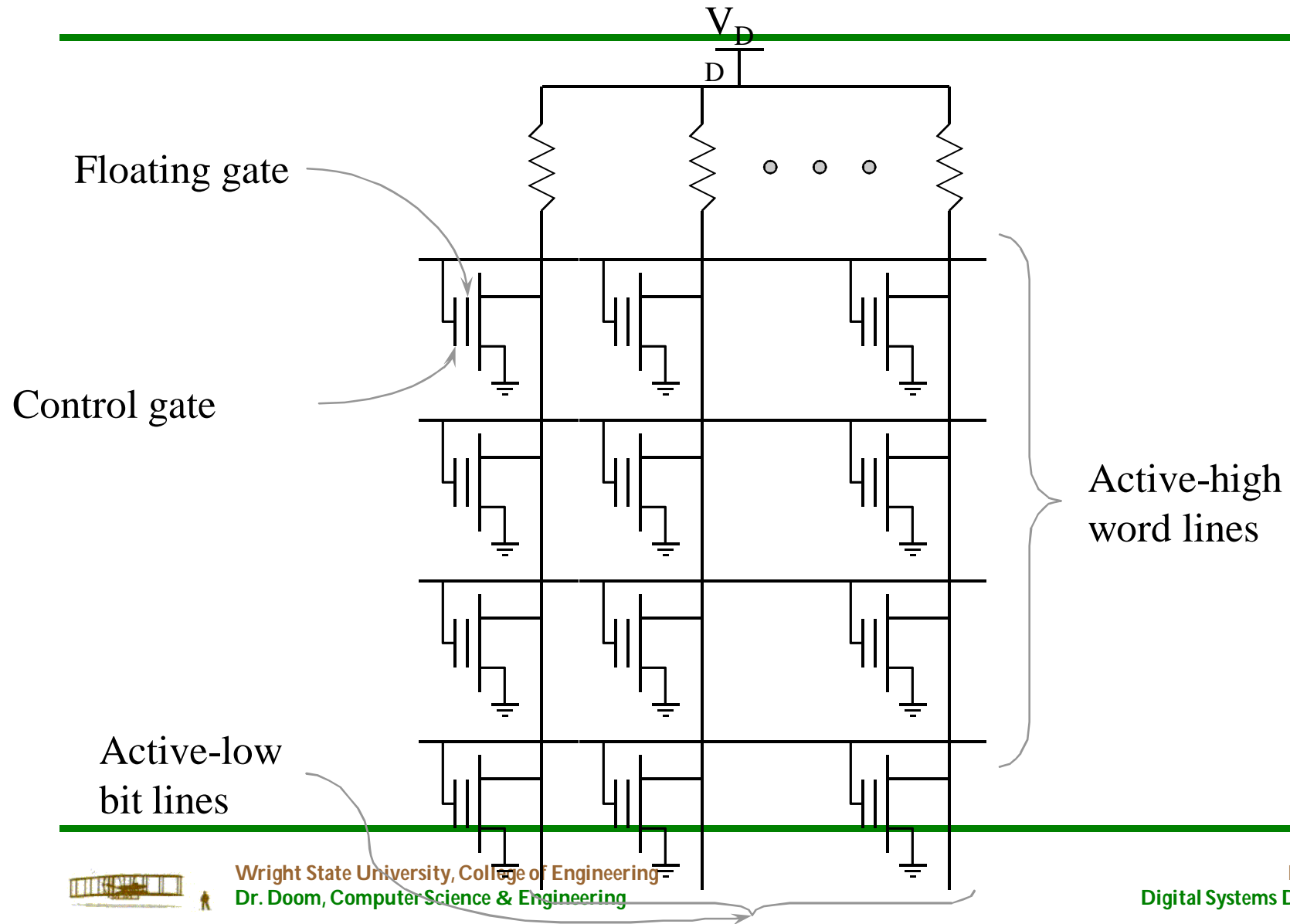
# Internal Structure of Transistor ROM

- Replace diodes with MOS transistors
- Change decoder to active-high outputs

Transistor  $\Rightarrow$  1  
No transistor  $\Rightarrow$  0



# EPROM and EEPROM Structure



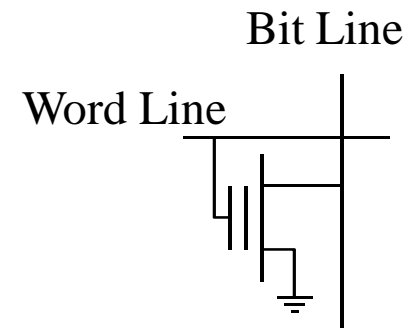




# Types of ROMs (2)

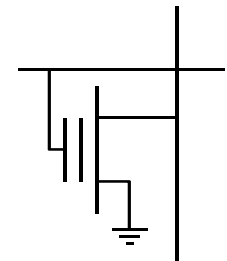
- EPROM

- Erasable Programmable ROM
- Charge trapped on extra “floating gate
- Exposure to UV light removes charge
  - 10-20 minutes
  - Quartz Lid = expensive package
- Limited number of erasures (10-100)



- EEPROM (E<sup>2</sup>ROM)

- Electrically Erasable ROM
- Floating gates charged/discharged electrically
- Not RAM! (relatively slow charge/discharge)
- limited number of charge/discharge cycles
  - (initially 10,000s; now millions)



# Types of ROMs (3)

- Flash Memory
  - Electronically erasable in blocks
  - 100,000 erase cycles
  - Simpler and denser than EEPROM
  - Often used for firmware

Type	Technology	Read Cycle	Write Cycle	Comments
Mask ROM	NMOS, CMOS	20-200 ns	4 weeks	Write once; low power
Mask ROM	Bipolar	<100 ns	4 weeks	Write once; high power; low density
PROM	Bipolar	<100 ns	5 minutes	Write once; high power; no mask charge
EPROM	NMOS, CMOS	25-200 ns	5 minutes	Reusable; low power; no mask charge
EEPROM	NMOS	50-200 ns	10 $\mu$ s/byte	10,000 writes/location limit
FLASH	CMOS	25-200 ns	10 $\mu$ s/block	100,000 – 1,000,000s erase cycles

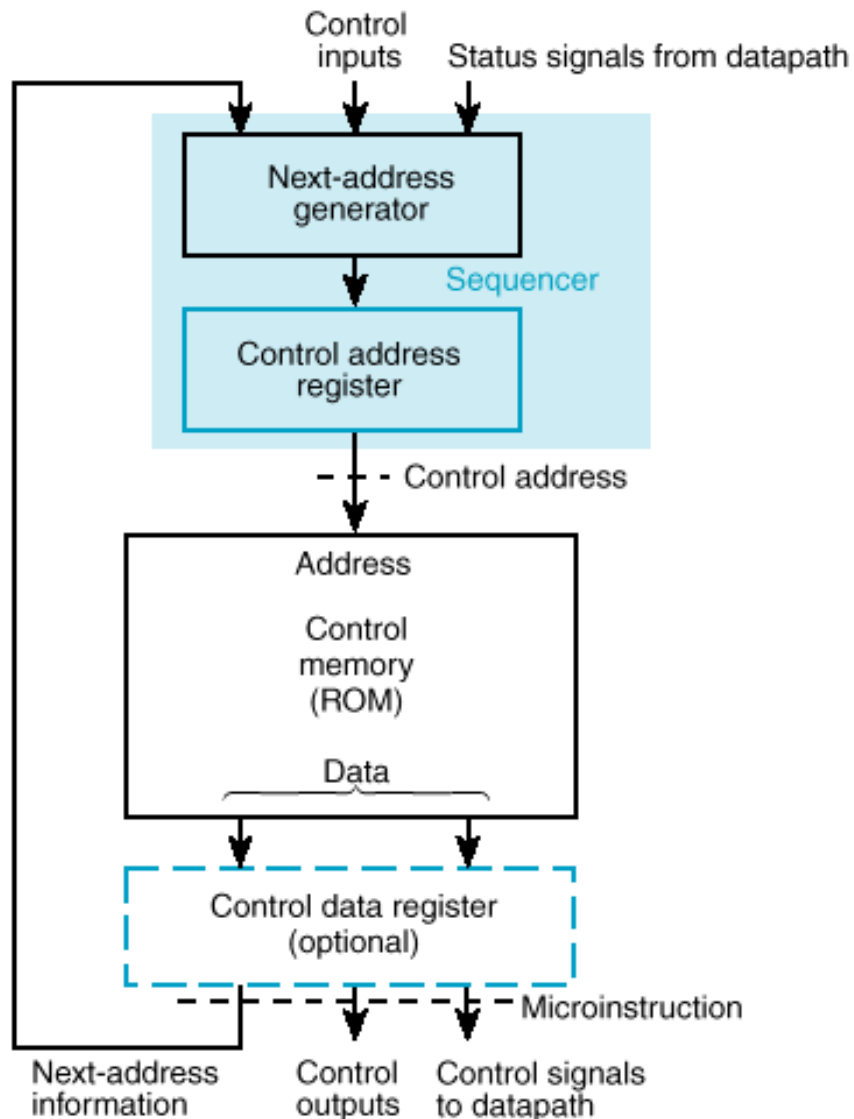


# The Control Unit

- 
- The control unit generates the signals for sequencing the operations in the datapath
    - A sequential circuit with states that *dictate the control signals* for the system
    - Using status conditions and control inputs, the sequential control unit *determines the next state* in which additional microoperations are activated.
  - Hardwired Control
    - The control unit is implemented to provide a particular digital function
  - Microprogrammed Control
    - The control unit's binary control values are stores as words in a microprogrammed control (usually ROM).
    - Each word in the control contains a microinstruction
    - A sequence of microinstructions constitutes a microprogram
    - Firmware!
- 



# Diagram of a Microprogrammed Control Unit

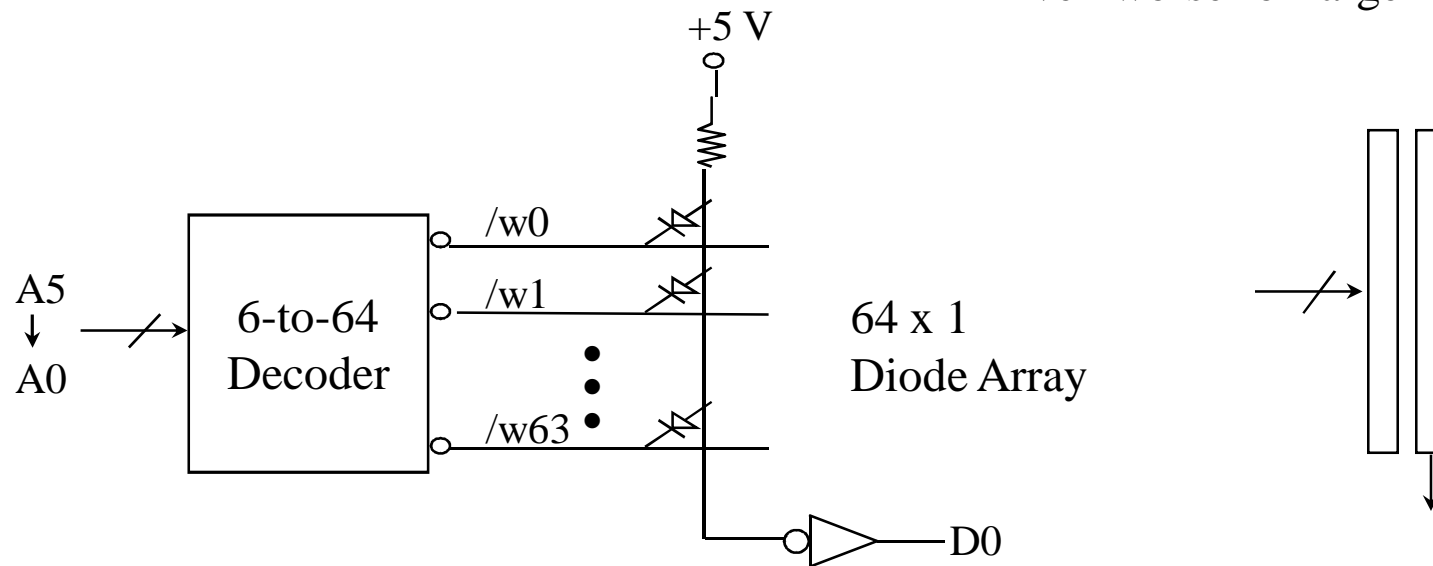


- The values of the control signals (and outputs) are determined by the contents of the Control Memory (a.k.a. the Control Store)
- A portion of the contents of the Control Memory is used (along with the next set of inputs).
  - The “next-address” field maintains internal state

# Consider a 64 x 1 ROM

This Decoder needs 64 6-input gates!

Very tall, narrow chip (BAD)  
Even worse for larger chips!

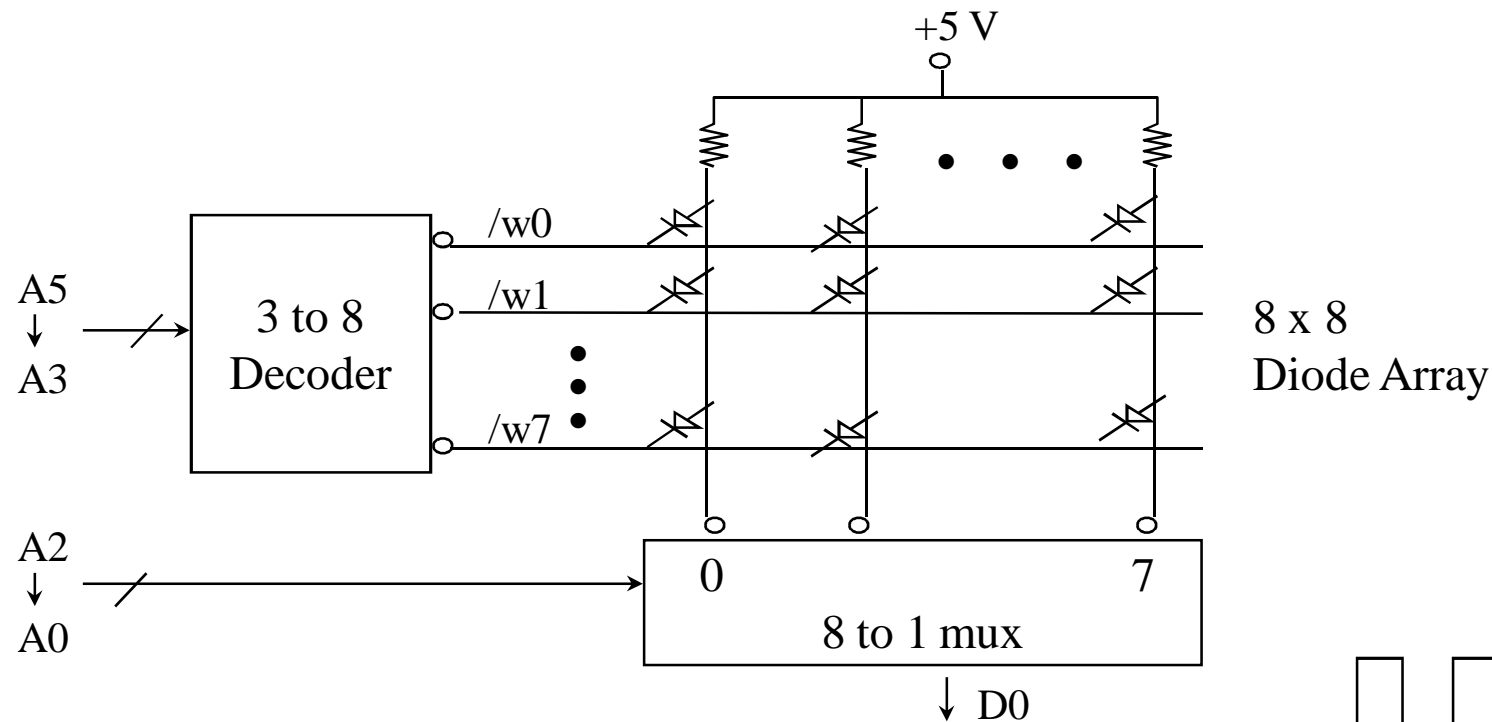


How can we make it more square?

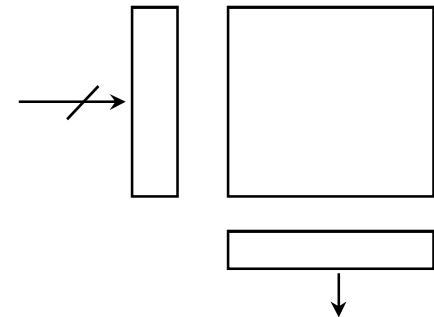




# 64 x 1 ROM with 2-Dimensional Decoding

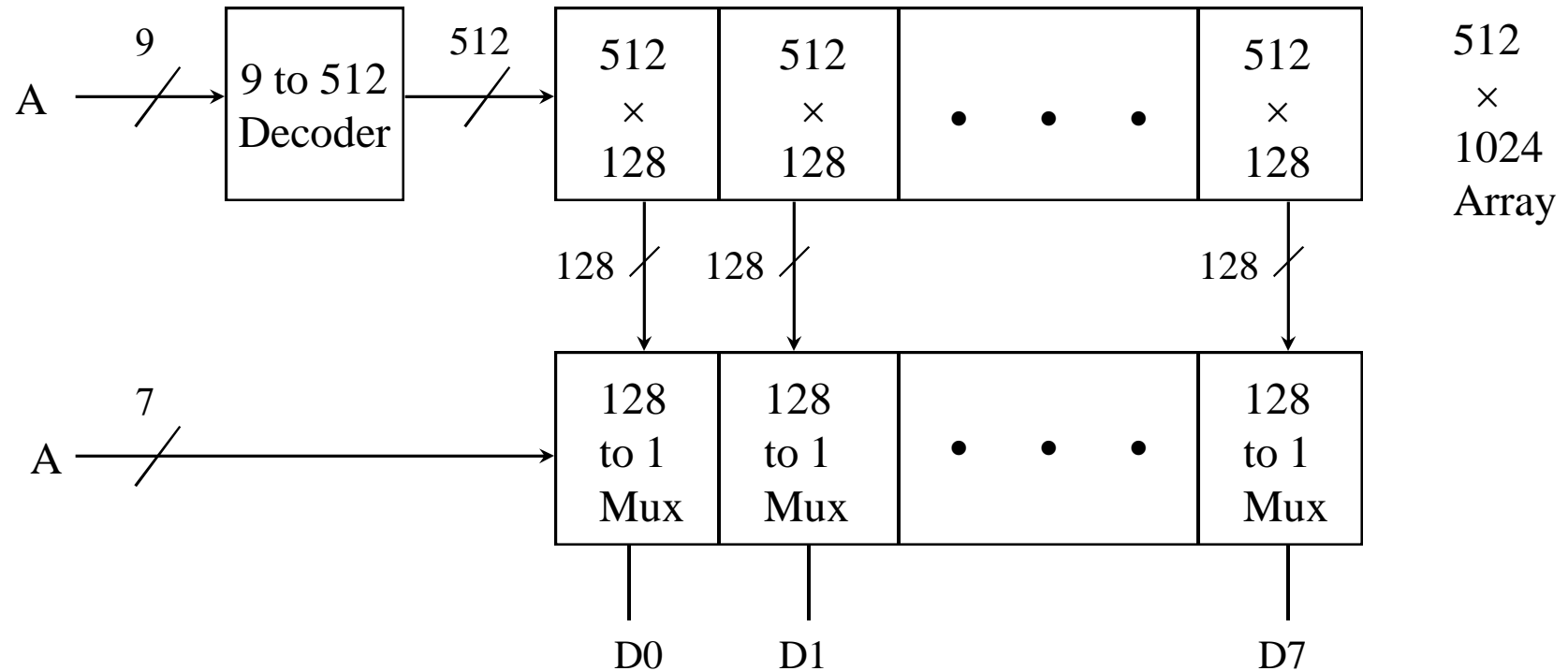


Decoder and mux = 8 3-input gates + 8 4-input gates





# 64K x 8 ROM with 2-D Decoding



Is this a square chip?

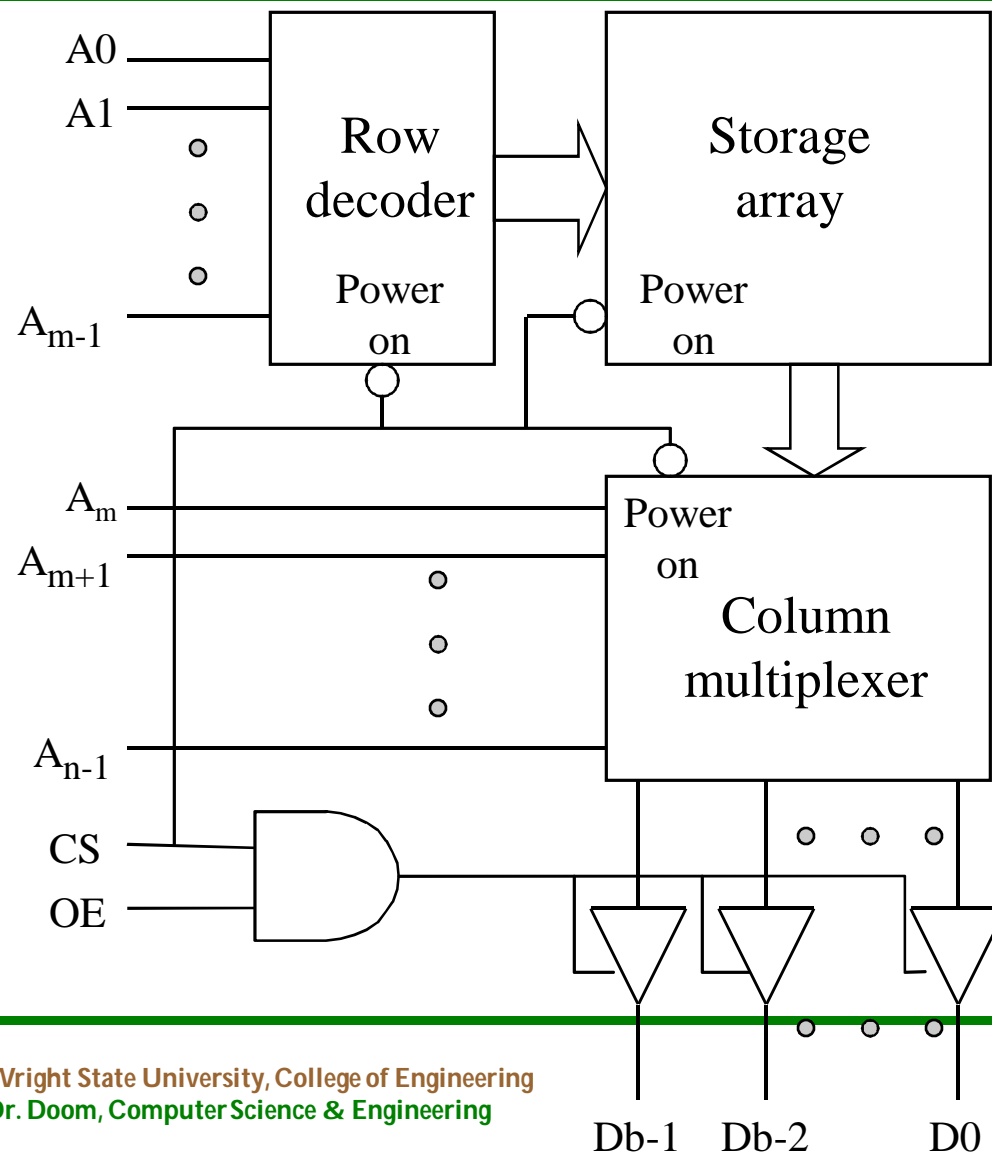
2-D Decoding / Coincident selection

$$64k \times 8 = 2^{16} * 2^3 = 2^{19}$$

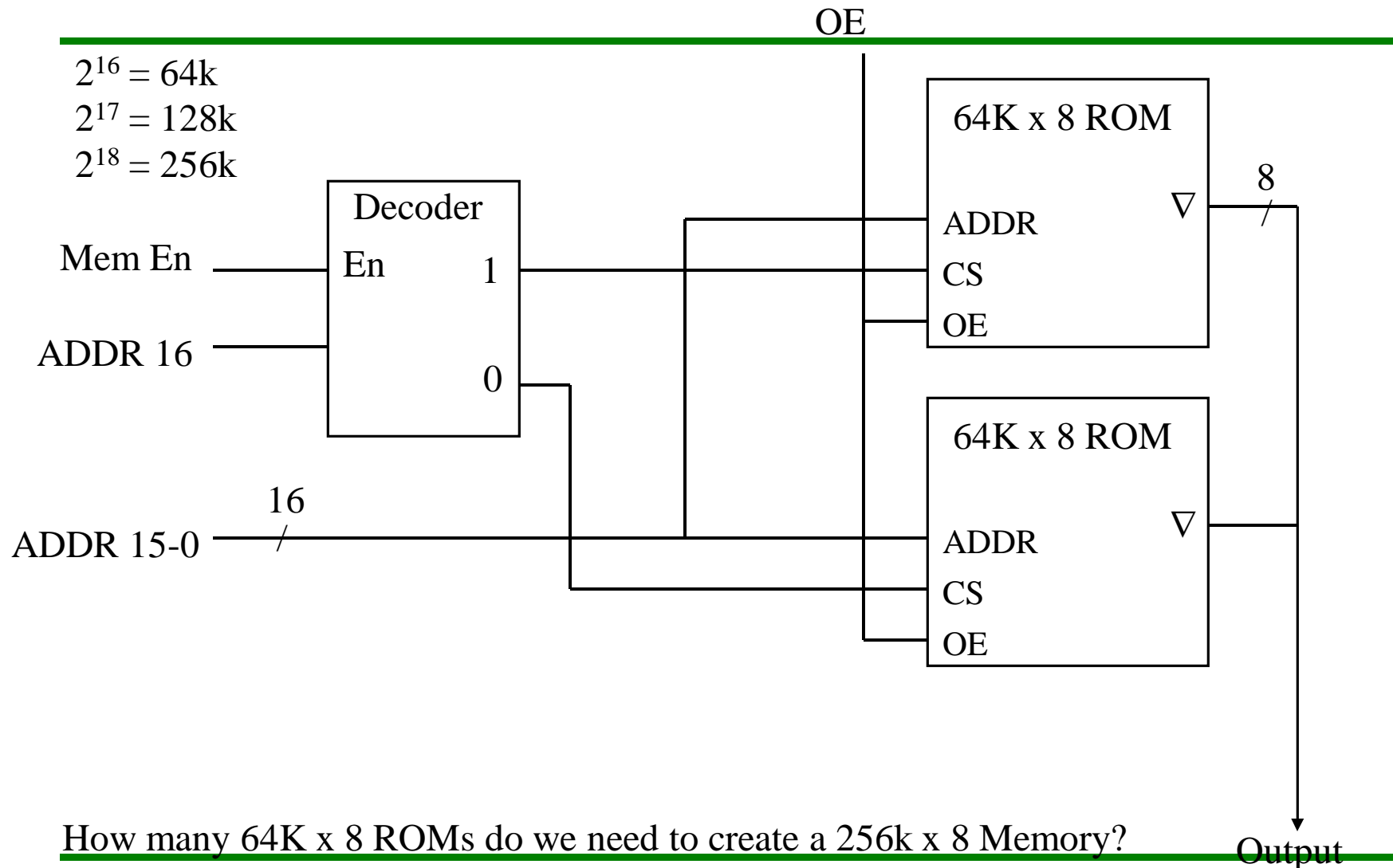
$$\text{square root } (2^{19}) \approx 2^9$$



# Internal $2^n \times b$ ROM Control Structure



# Composite 128K x 8 Memory



# Composite 64K x 16 Memory

