Registers and Register Transfers

CO 2206 Computer Organization

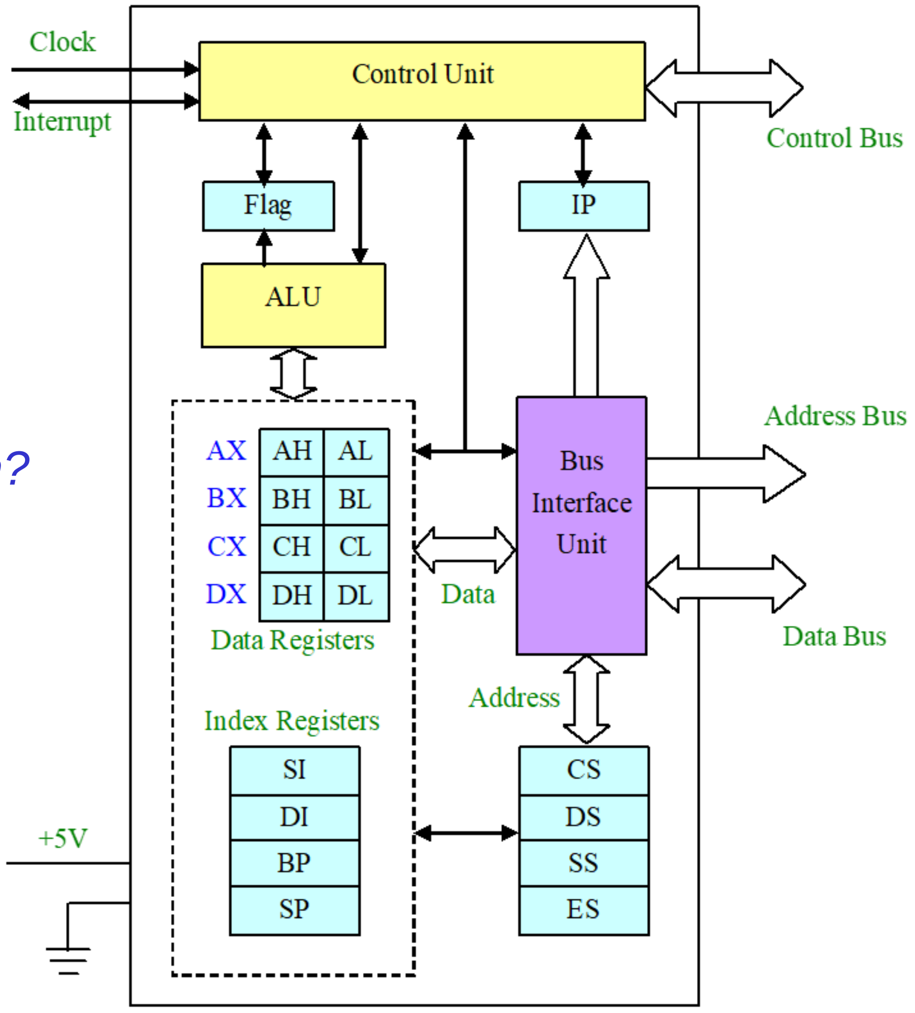
Topics

- Registers
  - Register with Parallel Load
- Register Transfer Operations
  - Microoperations
- Register Transfer Language
- Shift Registers
- Counters
- Register Cell Design
- Register Transfer Structures
- Serial Operations

CPU Recall: 8086 Architecture

Intel 8086

Internal Block Diagram



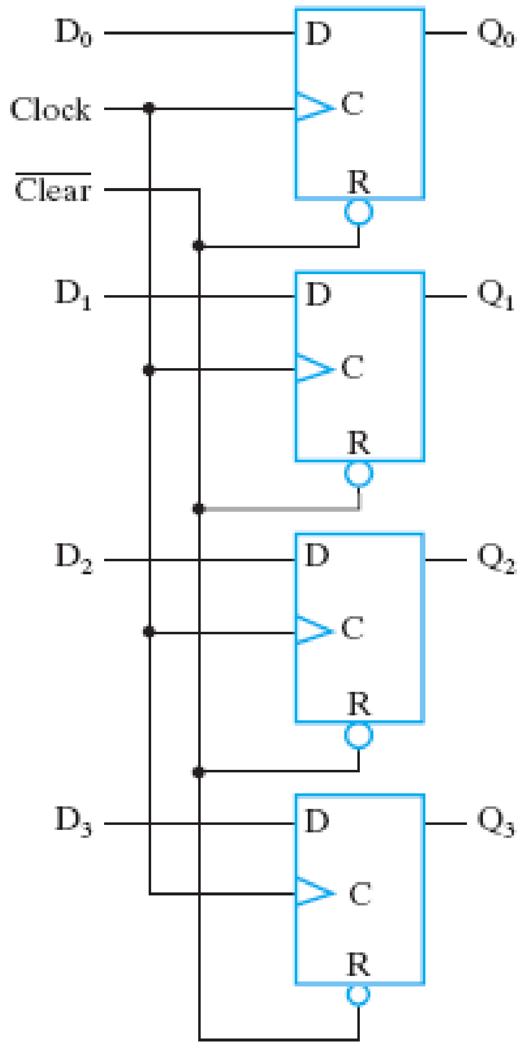
Identify Control and Datapath?

Registers

- Each flip-flop is capable of storing one bit of information
- A register includes a set of flip-flops, with gates that implement their state transitions
- An *n*-bit register has a group of *n* flip-flops capable of storing *n* bits
- Registers are useful for storing and manipulating information
- Simplest possible register consists of only flip-flops without any external gates

4-bit Register: an Example

- Common *Clock* input triggers all flip-flops on the rising edge of each pulse
  - four input values transferred into the register
- 4 outputs sampled to obtain the info stored in the register
- (Clear)' used to clear all register to 0's
  - 0 must be applied to flip-flops

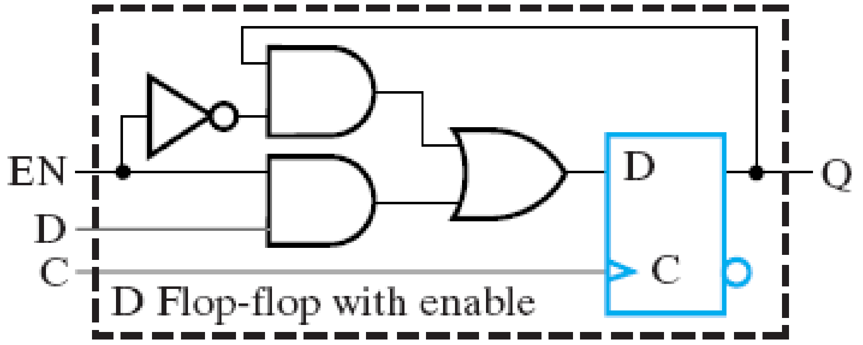


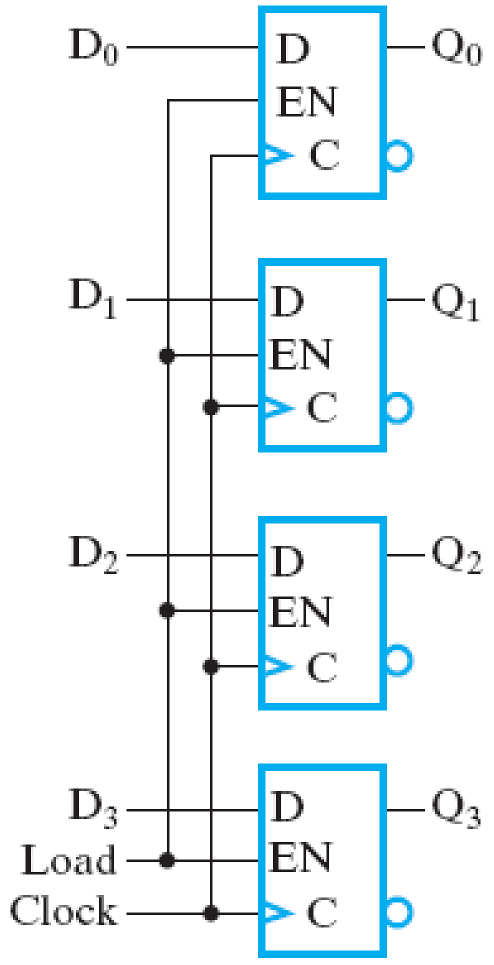
Register with Parallel Load - 1

- *Loading* – transfer of info into register
- *Parallel-loading*
  - all bits loaded simultaneously with a single clock pulse
  - Master-clock generator supplies a continuous train of clock pulses to all registers in the system
  - A separate control signal (*Load*) decides what specific clock pulses will have an effect on particular register

Register with Parallel Load - 2

- When Load = 1,
  - input transferred into register on next clk
- When Load = 0,
  - inputs inhibited
  - D reloaded with present value
  - Feedback necessary for D





Register Transfer Operations

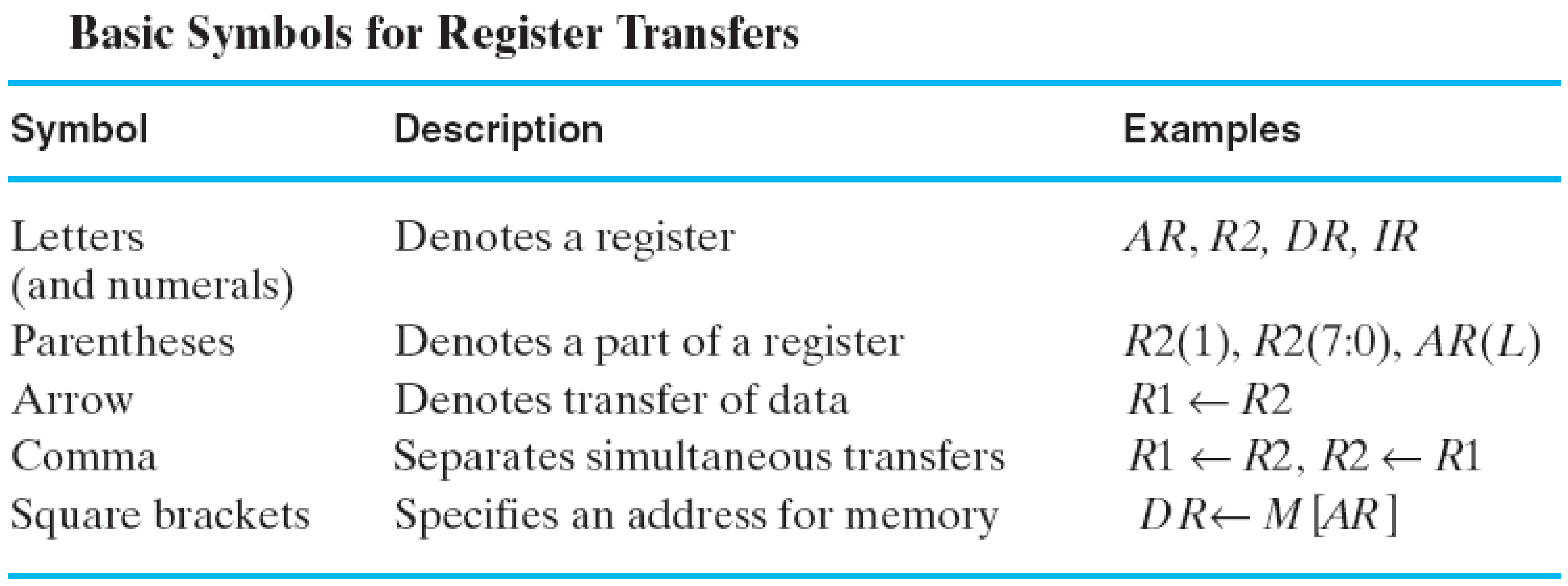
- The movement of data stored in registers and processing performed on the data are referred as *register transfer operations*
- Register transfer operations are specified by 3 basic components:
  - The set of registers in the system
  - The operations performed on the data stored
  - The control that supervises the sequence of operations
- A register has the capability to perform one or more elementary operations such as load, count, add, subtract and shift

Microoperation

- An elementary operation performed on data stored in registers is called a *microoperation*. Four common microop:
  - – Transfer: data between registers
  - – Arithmetic: arithmetic on data in registers
  - – Logic: bit manipulation on data in registers
  - – Shift: shift data (bits) within registers

Register Transfer Language

- *Register transfer language* (RTL) is used to represent registers and specify the operations on their contents



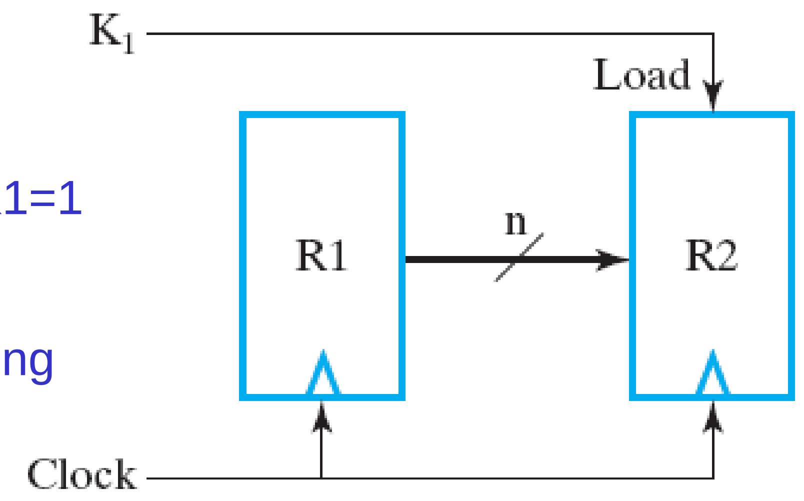
Basic Symbols for Register Transfers

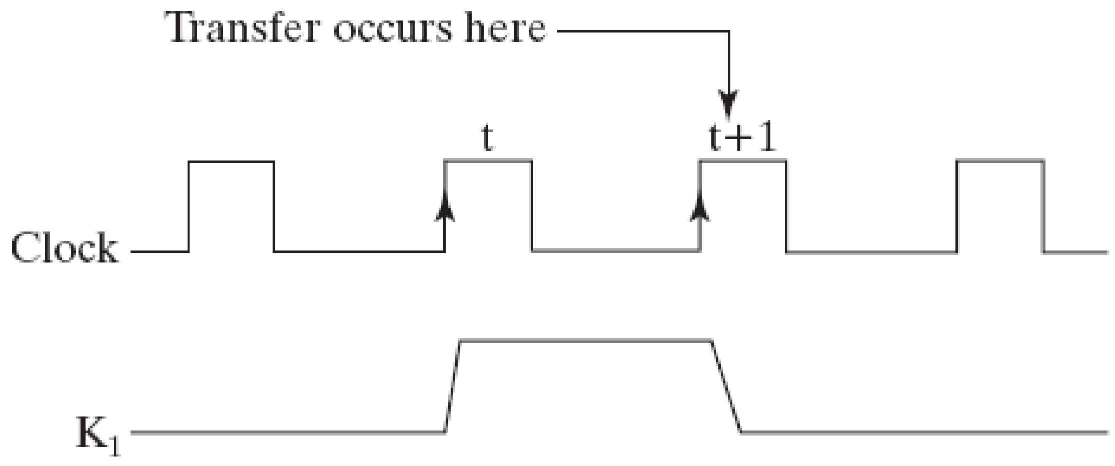
Symbol	Description	Examples
Letters (and numerals)	Denotes a register	AR, $R2$, DR, IR
Parentheses	Denotes a part of a register	$R2(1)$, $R2(7{:}0)$, $AR(L)$
Arrow	Denotes transfer of data	$R1 \leftarrow R2$
Comma	Separates simultaneous transfers	$R1 \leftarrow R2$, $R2 \leftarrow R1$
Square brackets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

RTL: Example

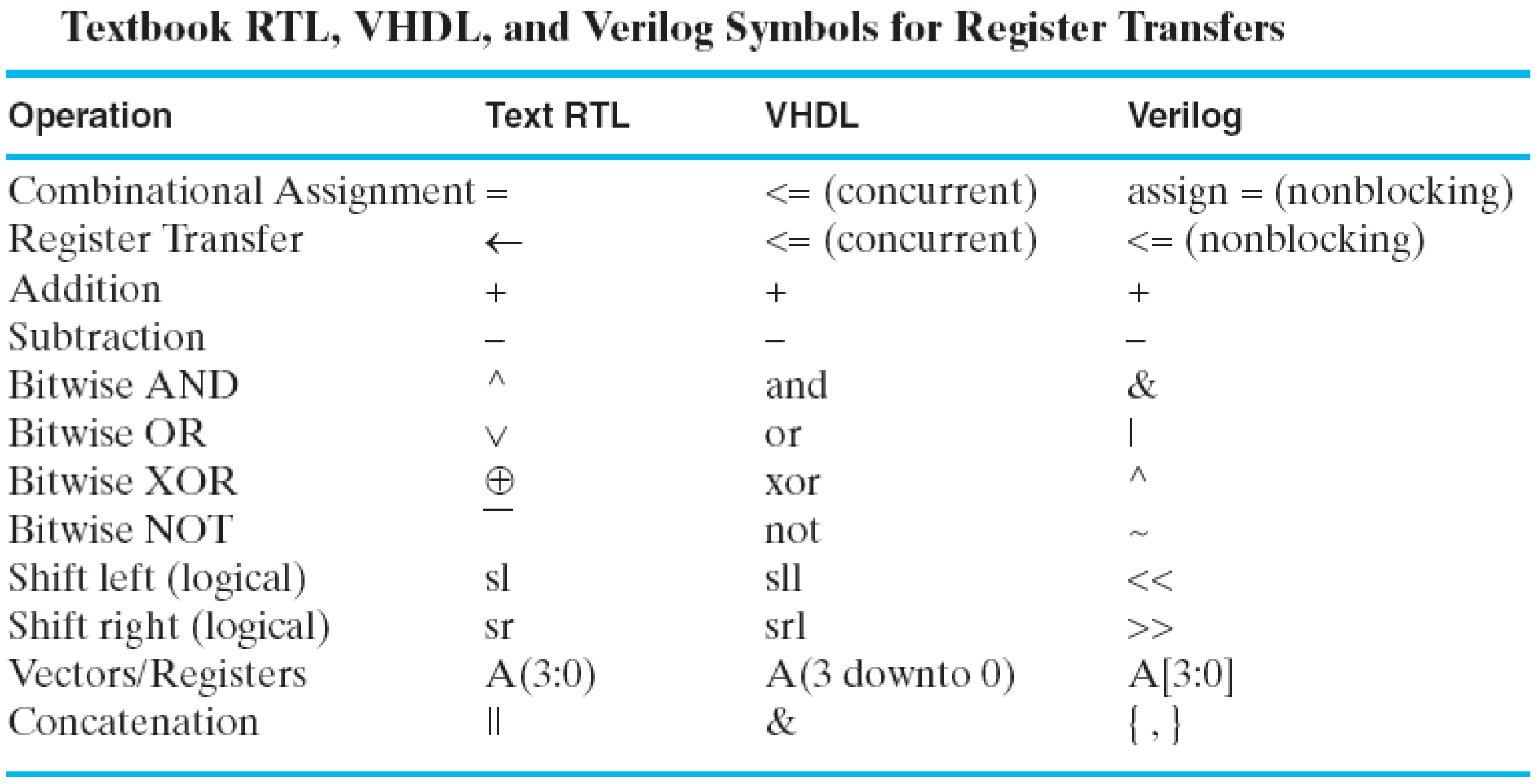
Operation: transfer from R1 to R2 when K1=1
RTL: K1:(R2 ← R1)

Note operation only executed upon triggering





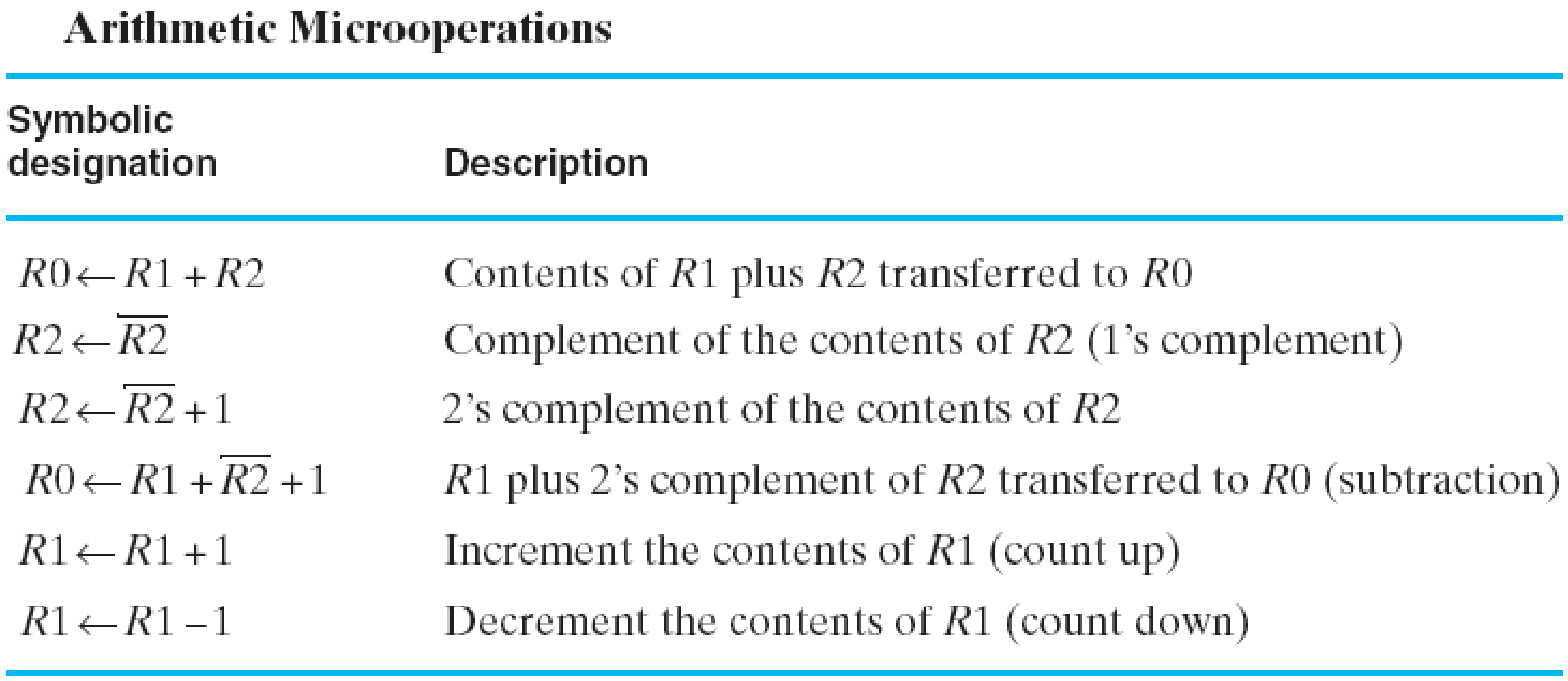
RTL Operators



Textbook RTL, VHDL, and Verilog Symbols for Register Transfers

Operation	Text RTL	VHDL	Verilog
Combinational Assignment	=	<= (concurrent)	assign = (nonblocking)
Register Transfer	$\leftarrow$	<= (concurrent)	<= (nonblocking)
Addition	+	+	+
Subtraction	–	–	–
Bitwise AND	$\wedge$	and	&
Bitwise OR	$\vee$	or	|
Bitwise XOR	$\oplus$	xor	^
Bitwise NOT	$\overline{}$	not	~
Shift left (logical)	sl	sll	<<
Shift right (logical)	sr	srl	>>
Vectors/Registers	A(3:0)	A(3 downto 0)	A[3:0]
Concatenation	||	&	{ , }

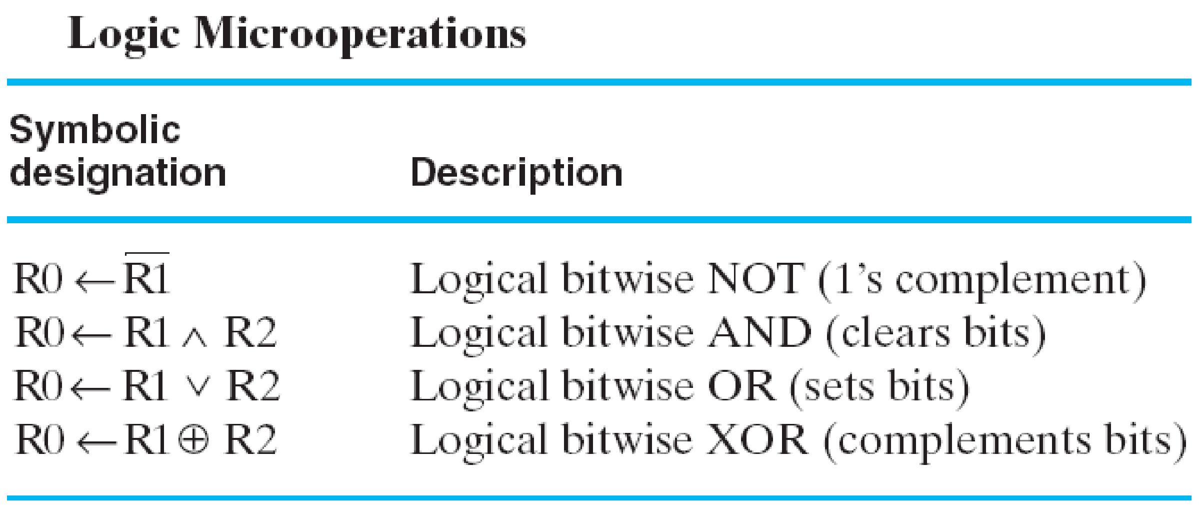
RTL: Examples 1



Arithmetic Microoperations

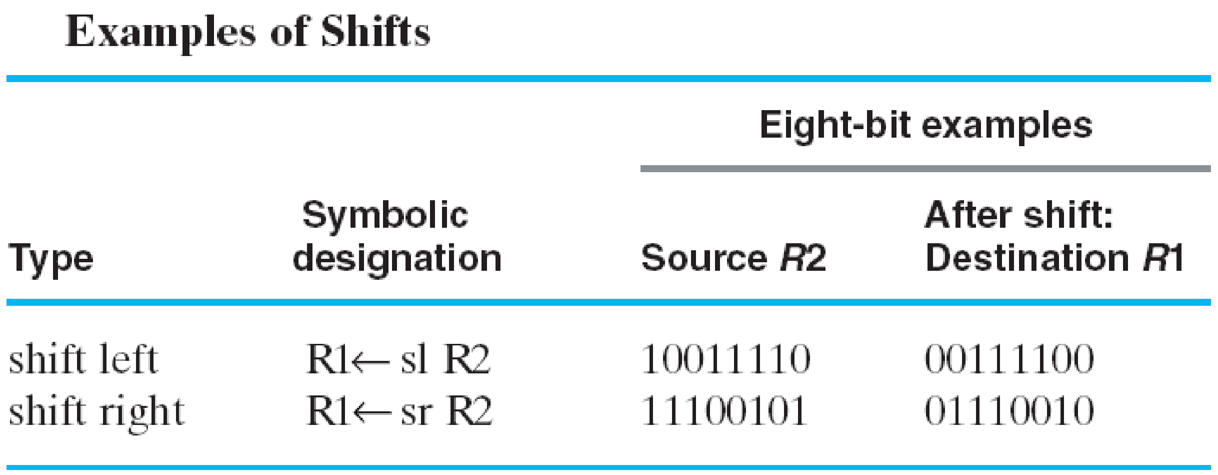
Symbolic designation	Description
$R0 \leftarrow R1 + R2$	Contents of $R1$ plus $R2$ transferred to $R0$
$R2 \leftarrow \overline{R2}$	Complement of the contents of $R2$ (1's complement)
$R2 \leftarrow \overline{R2} + 1$	2's complement of the contents of $R2$
$R0 \leftarrow R1 + \overline{R2} + 1$	$R1$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

RTL: Examples 2



Logic Microoperations

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)

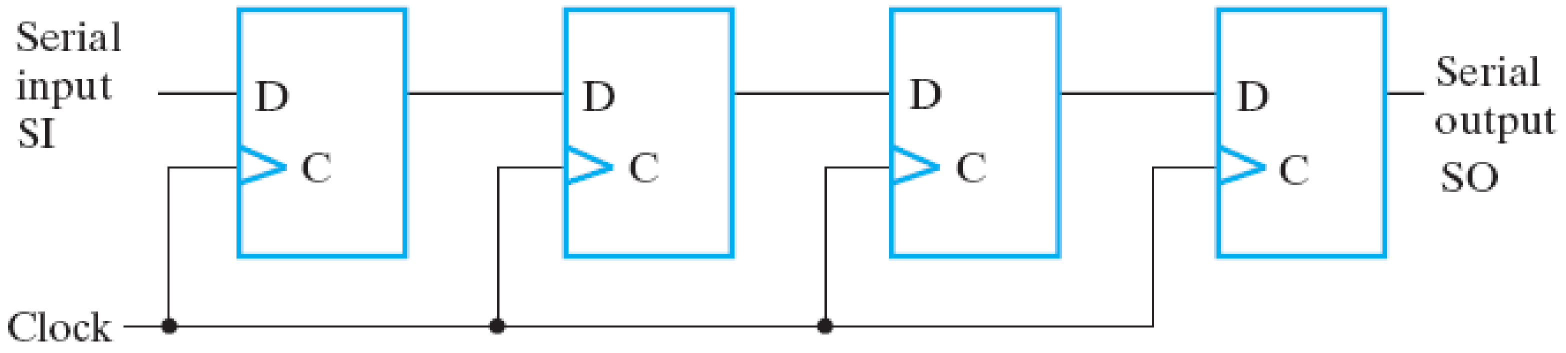


Examples of Shifts

		Eight-bit examples	
Type	Symbolic designation	Source *R*2	After shift: Destination *R*1
shift left	$R1 \leftarrow \text{sl } R2$	10011110	00111100
shift right	$R1 \leftarrow \text{sr } R2$	11100101	01110010

Shift Registers - 1

- *Shift register*
  - a register capable of shifting its stored bits either to the right and/or left.
  - consists of chain of flip-flops connected in cascade, with output of one connected to input of next



Shift Registers - 2

- All flip-flops receive a common clk pulse
- Each clk shifts the contents of the register one bit position to the right
- SI determines what goes into leftmost flip-flop during shift
- SO taken from output of rightmost flip-flop prior to application of pulse

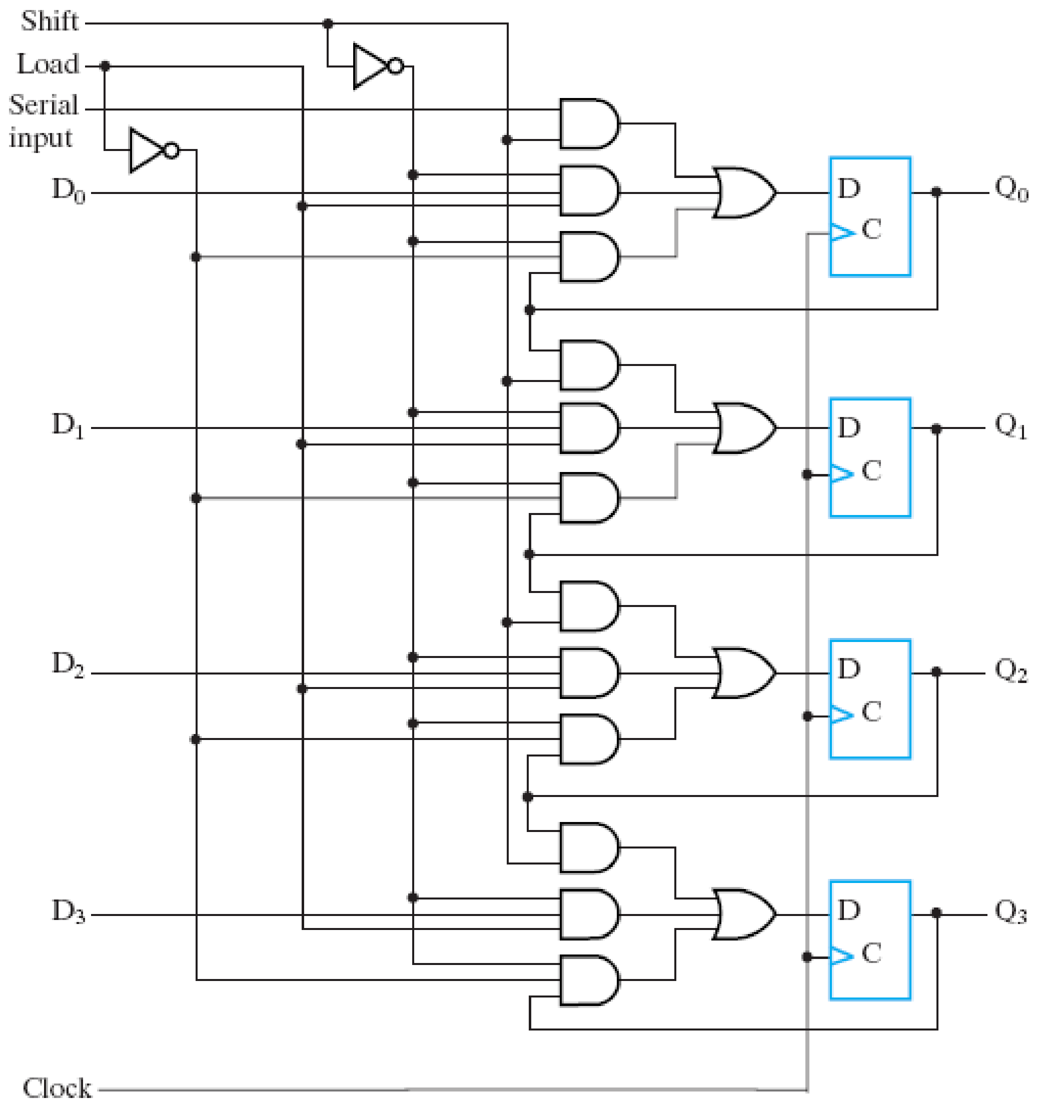
Shift Register with Parallel Load - 1

- A shift register with accessible flip-flop outputs and parallel load can be used for converting incoming parallel data to outgoing serial data and vice-versa
- 2 control inputs – *shift*, *load*

Shift	**Load**	**Operation**
0	0	No change
0	1	Load parallel data
1	X	Shift down from Q_0 to Q_3

Shift Register with Parallel Load - 2

- Parallel load allows for an initial bit pattern to be loaded, which can then be shifted around the registers

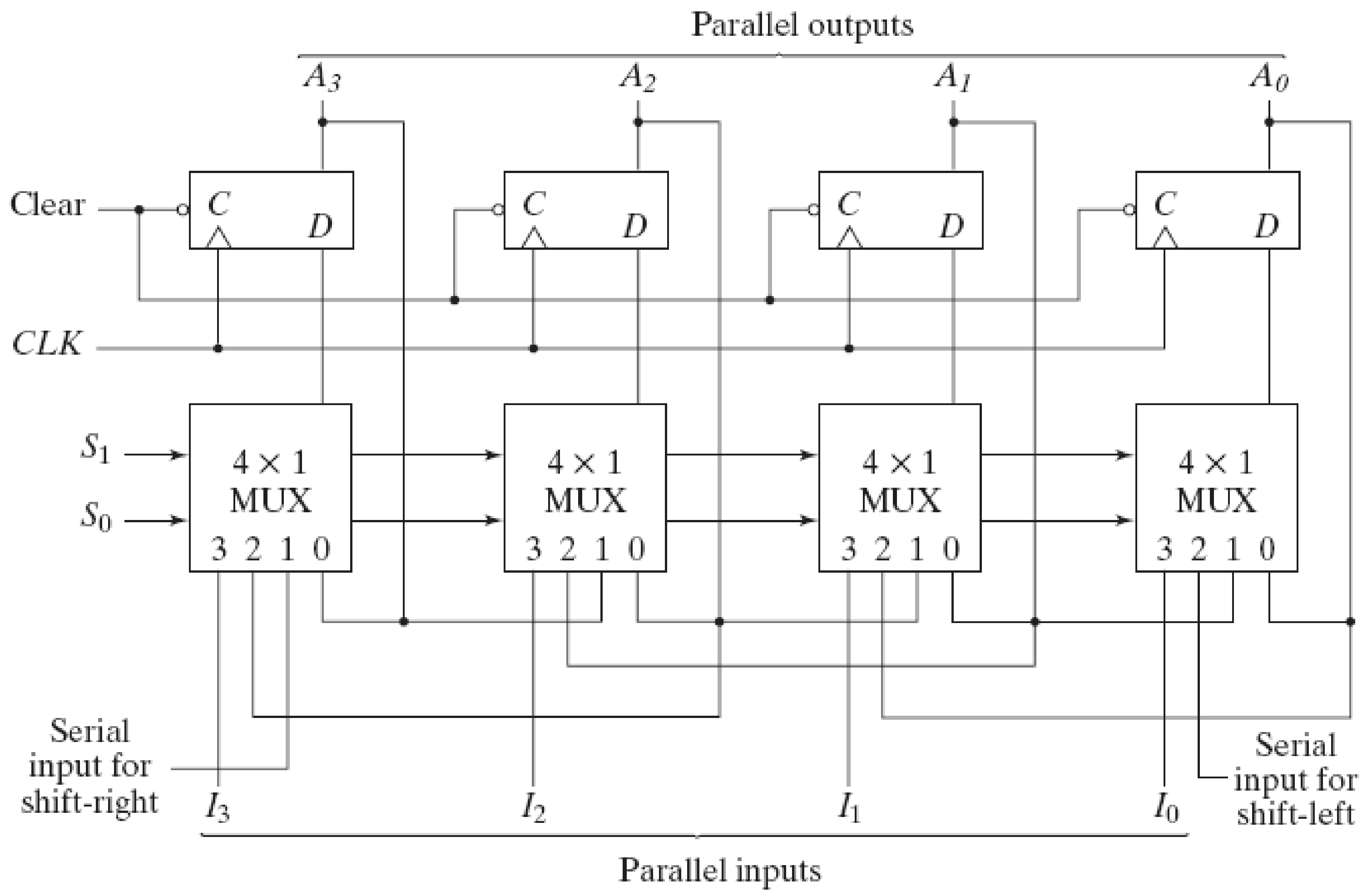


4-bit Universal Shift Register - 1

- 4 Mux have 2 common selection s0, s1
  - When s1s0 = 00,
    - no change of states
  - When s1s0 = 01,
    - shift right
  - When s1s0 = 10,
    - shift left
  - When s1s0 = 11,
    - parallel load

*the **multiplexers** act as switches to reconfigure the interconnections to realise different operations*

4-bit Universal Shift Register - 2



Serial – Parallel Conversion

- Shift registers can be used for converting serial data to parallel data and vice versa
  - If we've all the flip-flop outputs of a shift register, then info entered serially by shifting can be taken out in parallel
  - If parallel-load capability is added, data entered in parallel can be taken out in serial fashion by shifting data stored in register

Counters

- Counter
  - goes through prescribed sequence of states upon application of input pulses
  - useful for counting the number of occurrences of an event and generating timing sequence to control the operations in a digital system
- Binary counter
  - Follows binary number sequence
  - n-bit binary counter consists of n flip-flops and can count in binary from 0 to 2^n-1.
- Counter may also follow any other sequence of states

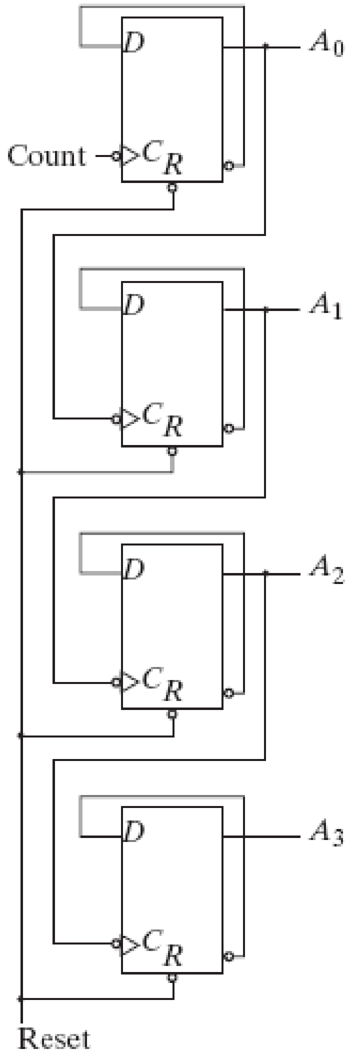
Ripple Counter (Asynchronous)

- A binary ripple counter
  - consists of a series connection of complementing flip-flops
  - the flip-flop output transition serves as a source for triggering other flip-flops
  - the flip-flop holding the least significant bit receives the incoming clock pulse
- Ripple counter is *asynchronous sequential circuit* and cannot be described by Boolean equations developed for describing clocked sequential circuit

Example: 4-bit Ripple Counter

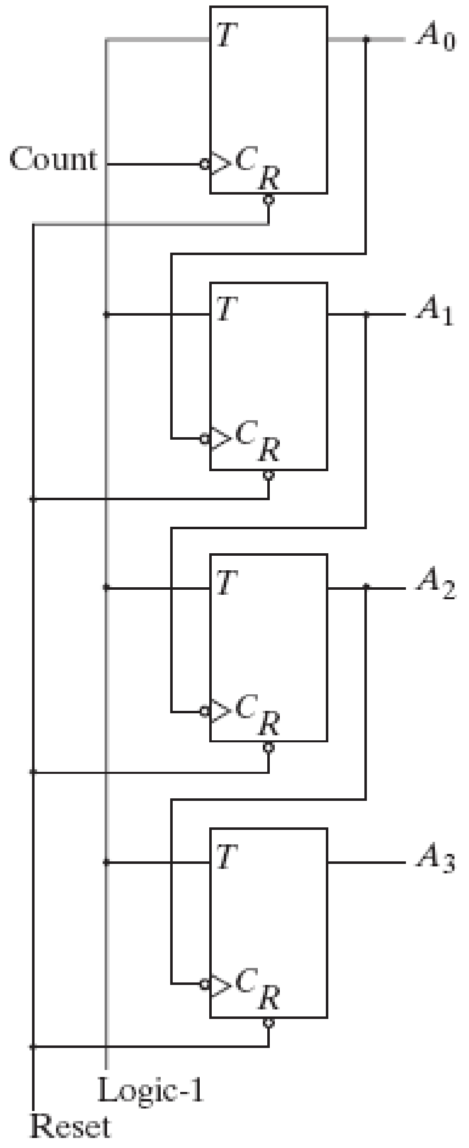
Upward Counting Sequence

Q_3	Q_2	Q_1	Q_0
0	0	0	0
0	0	0	1
0	0	1	0
0	0	1	1
0	1	0	0
0	1	0	1
0	1	1	0
0	1	1	1
1	0	0	0
1	0	0	1
1	0	1	0
1	0	1	1
1	1	0	0
1	1	0	1
1	1	1	0
1	1	1	1



Ripple Counter: Pros and Cons

- Advantage of ripple counter – simple
- Disadvantage
  - Delay dependence and unreliable operation
  - Large ripple counter can be slow due to the length of time required for the ripple to finish

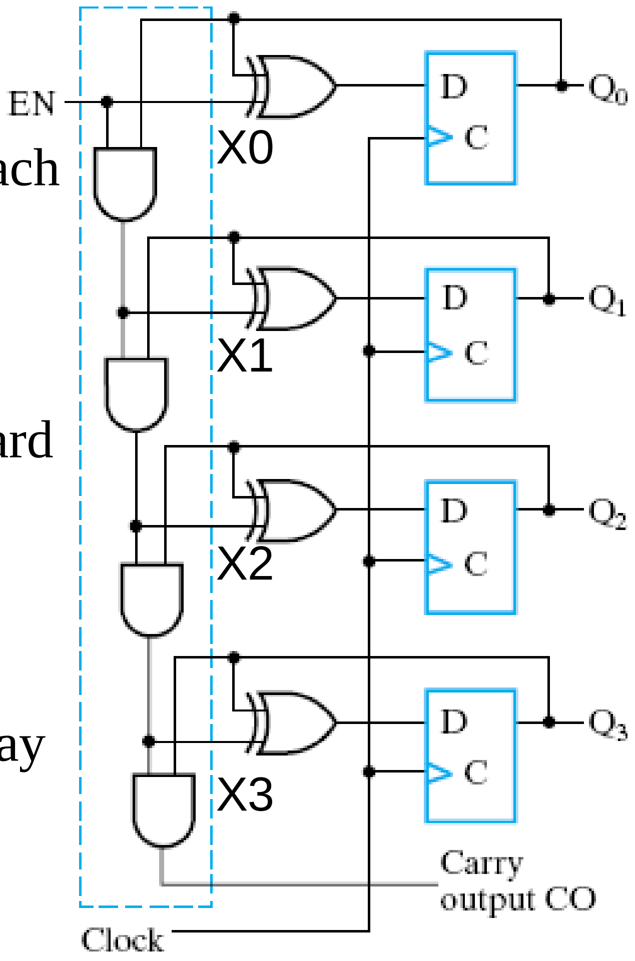


Synchronous Binary Counters

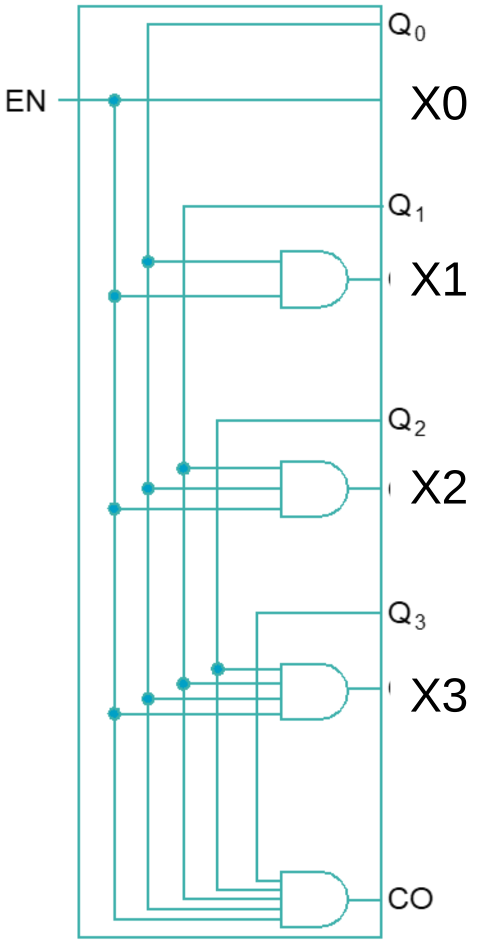
- In contrast to ripple counter,
  - Synchronous counter have the clock applied to the C inputs of all flip-flops
  - The common clock pulse triggers all flip-flops simultaneously rather than one at a time

4-bit Synchronous Counter

- Logic
  - XOR complements each bit
  - AND chain causes complement
    of a bit if all bits toward LSB
    from it equal 1
- Parallel gating
  - fixed 1 AND gate delay



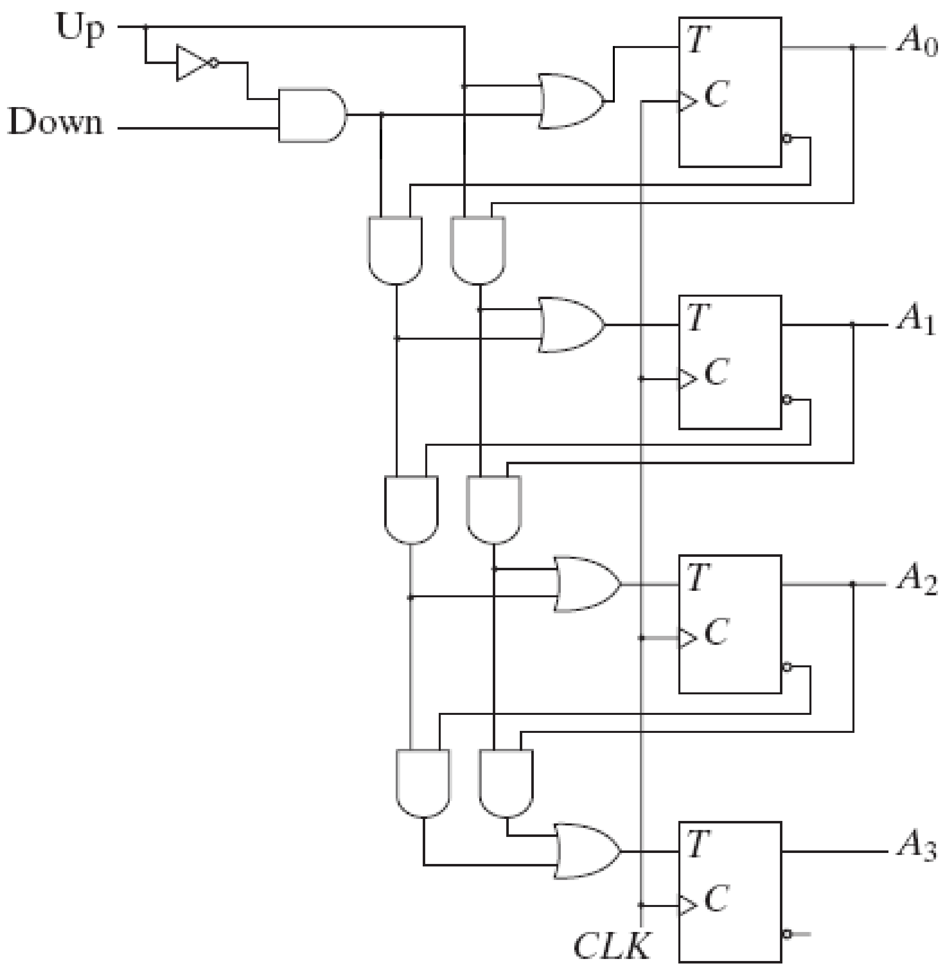
Serial Gating



Logic Diagram-Parallel Gating

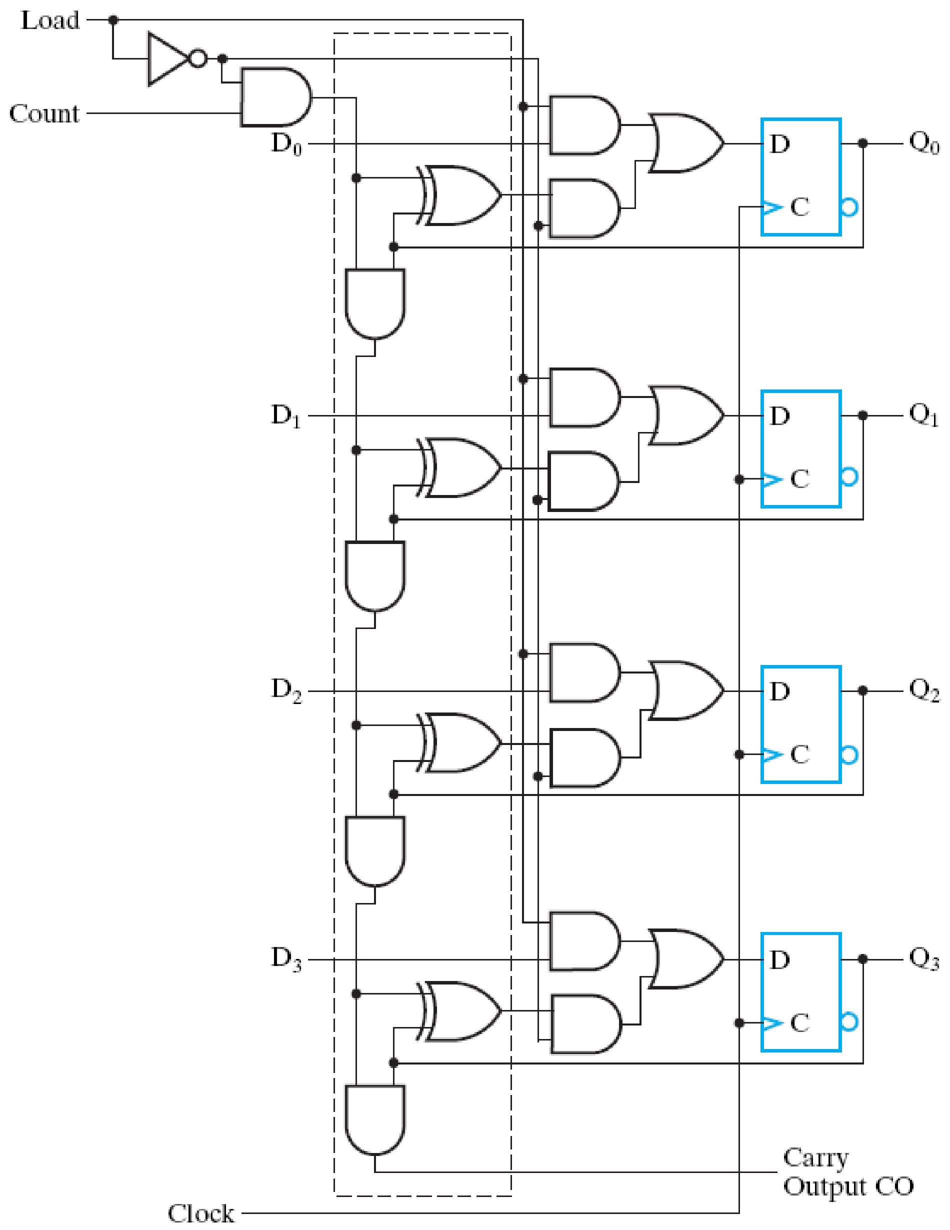
Up-down Parallel Binary Counter

- Up = 1, count up
- Up = 0, Down = 1, count down
- Up = Down = 0, no change in state
- Count-up binary counter, flip-flop at a position complemented if all lower-order bits are 1
- Count-down binary counter, flip-flop at a position complemented if all lower-order bits are 0



Binary Counter with Parallel Load - 1

- Counter often require a parallel-load capability for transferring an initial number prior to the count operation



Binary Counter with Parallel Load - 2

- Load = 1, disable count, D_0 to D_3 transfer into flip-flops
- Load = 0, enable count, operates as a counter
- Load = Count = 0, no change in states
- Carry-out = 1, if all flip-flops = 1 while count input is enabled
  - useful for expanding to > 4 bits

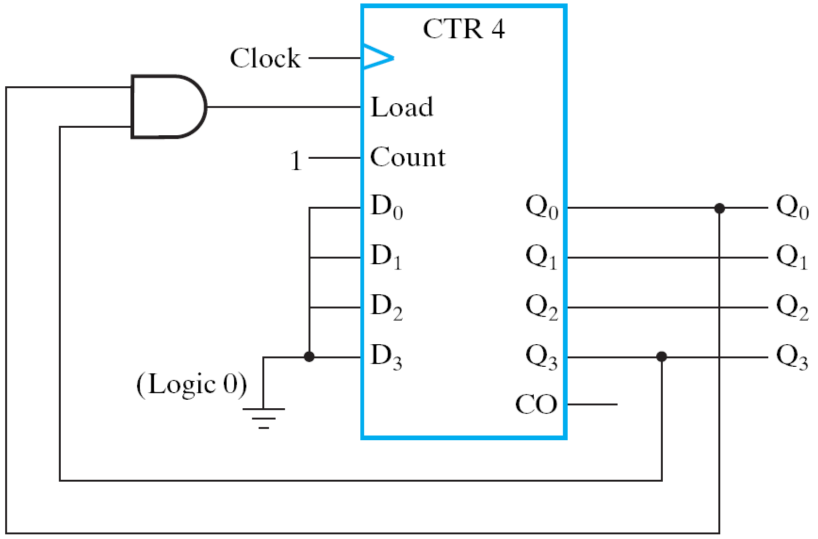
BCD Counter

- Y = 1 when present state is 1001
  - Y can enable the count of next decade when it switches from 1001 to 0000

Present State				Next State				Output
Q_8	Q_4	Q_2	Q_1	$D_8 = Q_8(t+1)$	$D_4 = Q_4(t+1)$	$D_2 = Q_2(t+1)$	$D_1 = Q_1(t+1)$	Y
0	0	0	0	0	0	0	1	0
0	0	0	1	0	0	1	0	0
0	0	1	0	0	0	1	1	0
0	0	1	1	0	1	0	0	0
0	1	0	0	0	1	0	1	0
0	1	0	1	0	1	1	0	0
0	1	1	0	0	1	1	1	0
0	1	1	1	1	0	0	0	0
1	0	0	0	1	0	0	1	0
1	0	0	1	0	0	0	0	1

BCD Counter: Using Binary Counter

- Reset to 0000 (i.e. make D=0000) when Q=1001 (10d)
  - using combinational logic, in this case an AND gate



BCD Counter: Using Sequential Logic

- Input equations
  - $D_1 = Q_1'$
  - $D_2 = Q_2 \oplus Q_1 Q_8'$
  - $D_4 = Q_4 \oplus Q_1 Q_2$
  - $D_8 = Q_8 \oplus (Q_1 Q_8 + Q_1 Q_2 Q_4)$
  - $Y = Q_1 Q_8$
- Unused case are used as don't care conditions

circuit can be build from four D flip-flops and some logic gates

Modulo-N Counter

- A *modulo-n* counter is a counter that goes through a repeated sequence of N states
  - E.g.
    - 4-bit binary counter is mod-16 counter
    - BCD counter is mod-10 counter
- Sequence may follow the binary count or any other arbitrary sequence
- The design follows the procedure for the design of synchronous sequential circuit

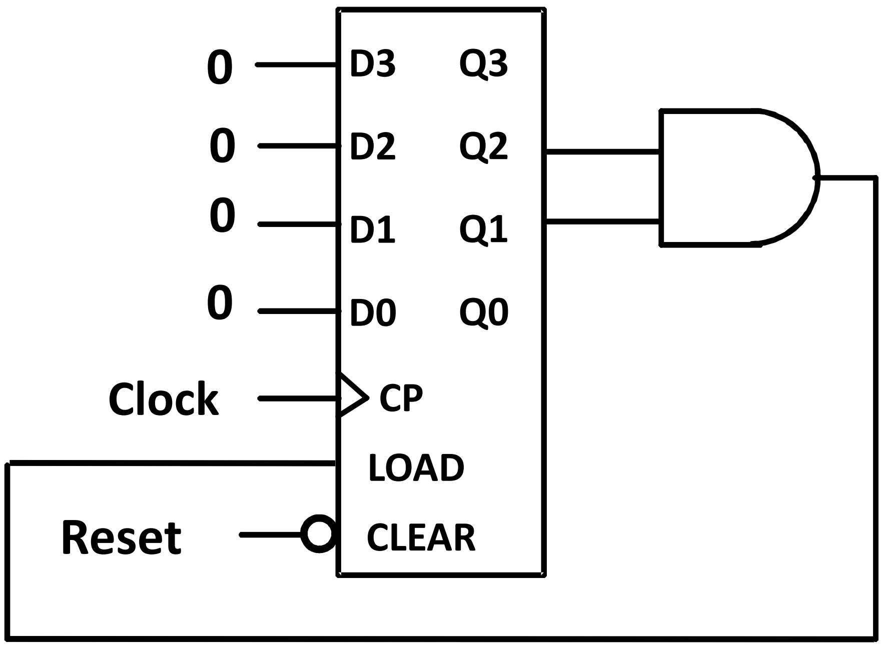
Counting Modulo N

- The following techniques use an *n*-bit binary counter with asynchronous or synchronous clear and/or parallel load:
  - Detect a *terminal count* of N in a Modulo-N count sequence to asynchronously Clear the count to 0 or asynchronously Load in value 0 (These lead to counts which are present for only a very short time and can fail to work for some timing conditions!)
  - Detect a terminal count of N - 1 in a Modulo-N count sequence to Clear the count synchronously to 0
  - Detect a terminal count of N - 1 in a Modulo-N count sequence to synchronously Load in value 0
- Alternatively, custom design a modulo N counter as done for BCD

Counting Modulo 7: 0 to 6

Synchronously Load on Terminal Count of 6

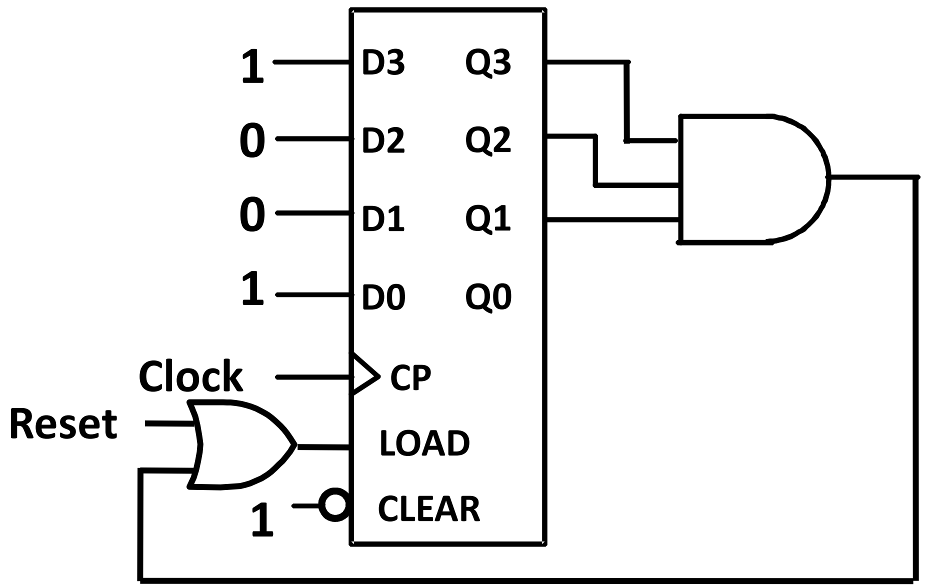
- A synchronous 4-bit binary counter with a synchronous load and an asynchronous clear is used to make a Modulo 7 counter
- Use the Load feature to detect the count "6" and load in "zero". This gives a count of 0, 1, 2, 3, 4, 5, 6, 0, 1, 2, 3, 4, 5, 6, 0, ...
- Using don't cares for states above 0110, detection of 6 can be done with Load = Q4 Q2



Counting Modulo 6: 9 to 14

Synchronously Preset 9 on Reset and Load 9 on Terminal Count 14

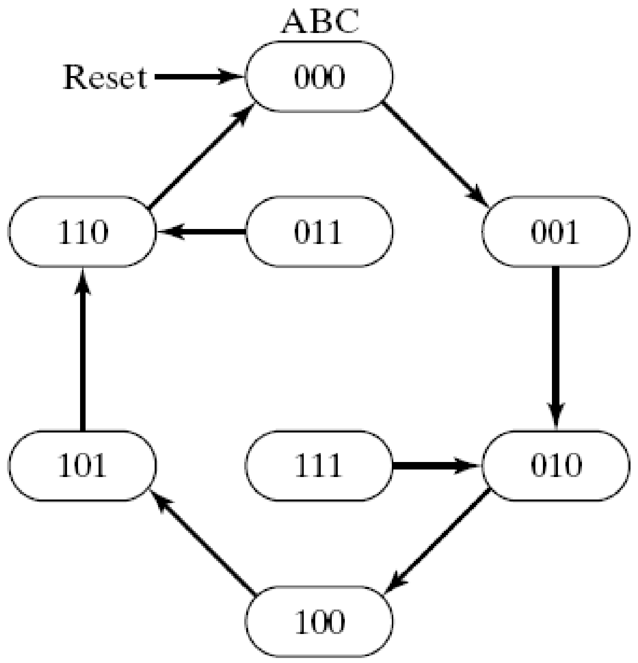
- A synchronous, 4-bit binary counter with a synchronous Load is to be used to make a Modulo 6 counter.
- Use the Load feature to preset the count to 9 on Reset and detection of count 14.



- This gives a count of 9, 10, 11, 12, 13, 14, 9, 10, 11, 12, 13, 14, 9, …
- If the terminal count is 15 detection is usually built in as Carry Out (CO)

Arbitrary Count Sequence - 1

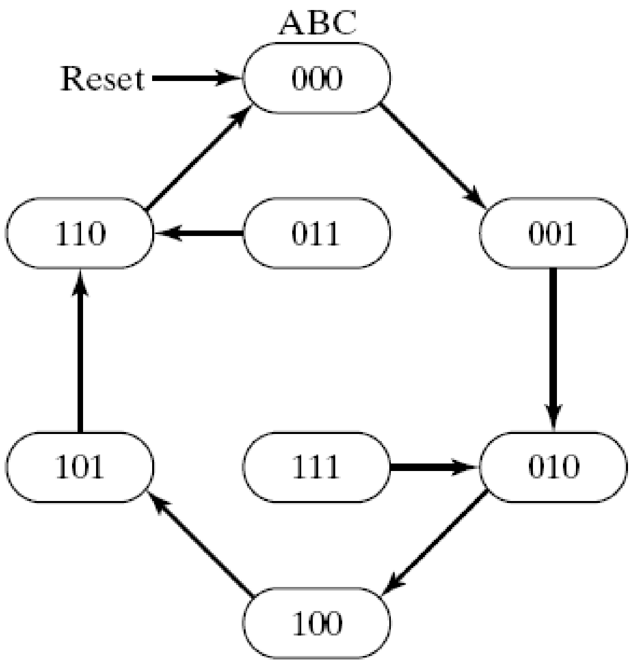
- $D_A = A \oplus B$
- $D_B = C$
- $D_C = B'C'$

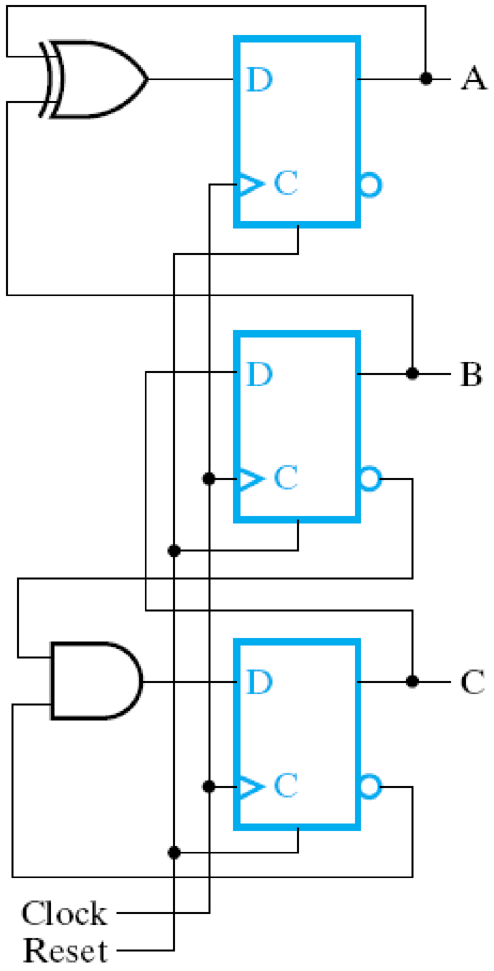


Present State			Next State		
A	B	C	DA = A(t+1)	DB = B(t+1)	DC= C(t+1)
0	0	0	0	0	1
0	0	1	0	1	0
0	1	0	1	0	0
1	0	0	1	0	1
1	0	1	1	1	0
1	1	0	0	0	0

Arbitrary Count Sequence - 2

- Since there are 2 unused states, we analyze the circuit to determine their effect



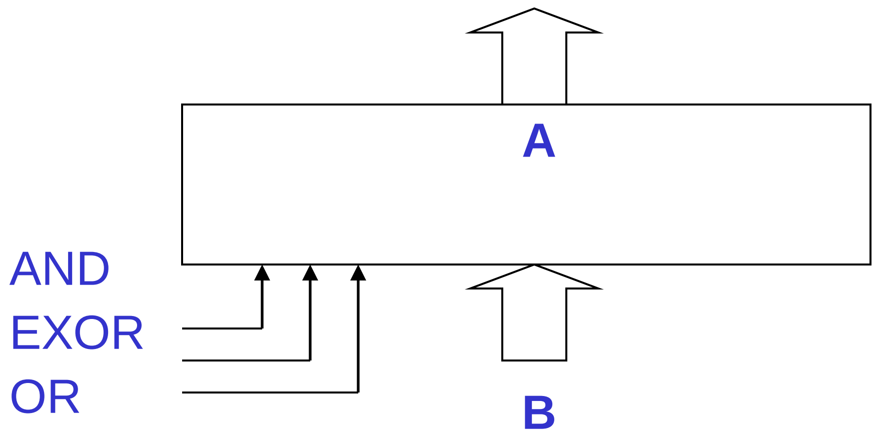


Register Cell Design

- We can design an n-bit register with one or more associated microoperations by
  - Designing a register cell and make n copies of it
- Relationship between next state and flip-flop output determine the design procedure

Register Cell Design: Example

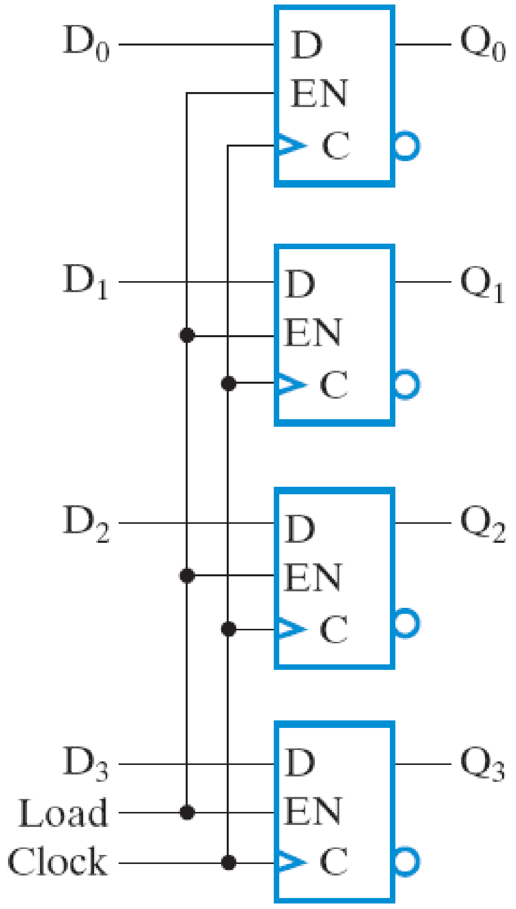
- A register A is to implement the following register transfer:
  - AND: $A \leftarrow A \wedge B$
  - EXOR: $A \leftarrow A \oplus B$
  - OR: $A \leftarrow A \vee B$



- Assume that
  - Only one of AND, EXOR, OR is equal to 1
  - For AND, EXOR, OR equal to 0, A remains unchanged

Register Cell Design : Example - 1

- Approach 1 (ad hoc):
  - Uses register with parallel load from D flip-flops with EN
    - LOAD = AND + EXOR + OR
    - $D_i = A(t+1)_i = AND \cdot A_iB_i + EXOR \cdot (A_i \oplus B_i) + OR \cdot (A_i + B_i)$
    - *LOAD initiate the transfer, and the combinational logic at Di execute the transfer depending on operation selected*



to add combinational circuit at inputs

Register Cell Design: Example – 2.1

	Next State A(t + 1)						
Present State A	(AND = 0) (EXOR=0) (OR=0)	(OR = 1) (B=0)	(OR = 1) (B=1)	(EXOR = 1) (B=0)	(EXOR = 1) (B=1)	(AND = 1) (B=0)	(AND = 1) (B=1)
0	0	0	1	0	1	0	0
1	1	1	1	1	0	0	1

- Approach 2 (seq. circuit design):
  - Input equation
    - $D_i = A(t+1)_i = AND \cdot A_iB_i + EXOR \cdot (A_iB_i' + A_i'B_i) + OR \cdot (A_i + B_i) + AND' \cdot EXOR' \cdot OR' \cdot A_i$

In Approach 1, the last term of the above equation was implemented through the LOAD

Register Cell Design: Example – 2.2

– Simplified

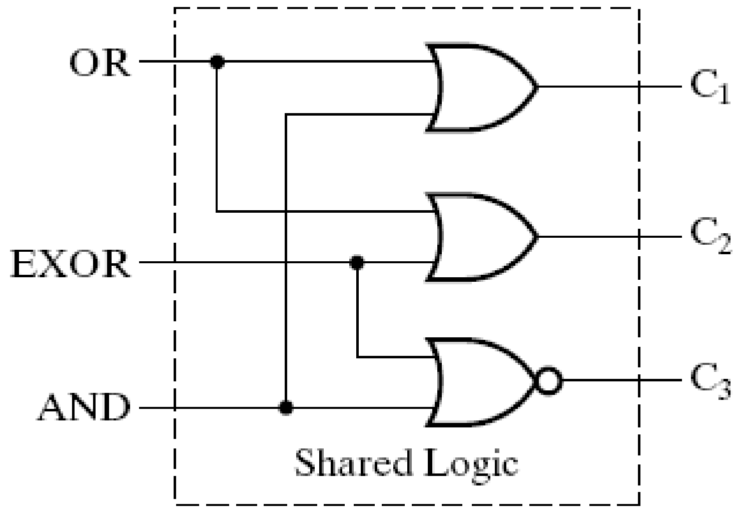
- $D_i = AND \cdot A_iB_i + EXOR \cdot (A_iB_i' + A_i'B_i) + OR \cdot (A_i + B_i) + AND' \cdot EXOR' \cdot OR' \cdot A_i$

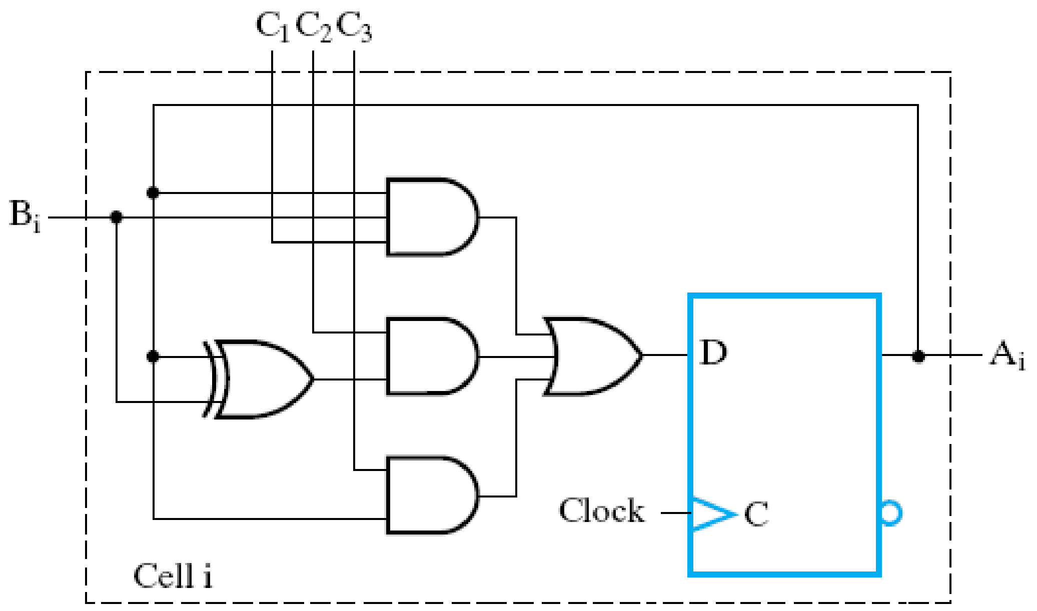
  $= (OR + AND) \cdot A_iB_i + (OR + EXOR) \cdot (A_iB_i' + A_i'B_i) + (AND + EXOR)' \cdot A_i$
- Let
  - $C_1 = OR + AND$
  - $C_2 = OR + EXOR$
  - $C_3 = (AND + EXOR)'$
- $D_i = C_1A_iB_i + C_2(A_iB_i' + A_i'B_i) + C_3A_i$

Observations:

- *Control variables (AND, OR, EXOR) are common to all cells and can be shared*
- *Individual signals (A, B) are for each cell*
- *Split the above in the equations*

Register Cell Design: Example – 2.3





Common to all cells

Individual cell

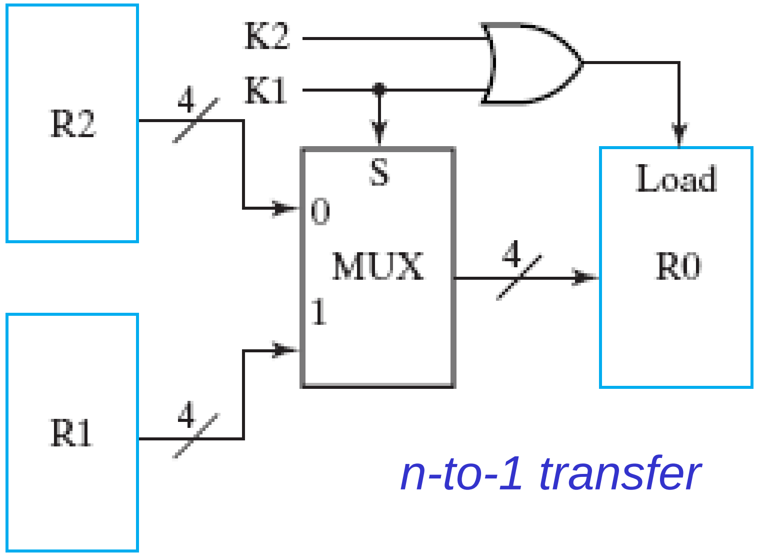
Register Transfer Structures

- Multiplexer-Based Transfers - Multiple inputs are selected by a multiplexer dedicated to the register
- Bus-Based Transfers - Multiple inputs are selected by a shared multiplexer driving a bus that feeds inputs to multiple registers
- Three-State Bus - Multiple inputs are selected by 3-state drivers with outputs connected to a bus that feeds multiple registers
- Other Transfer Structures - Use multiple multiplexers, multiple buses, and combinations of all the above

Multiplexer-based Transfers - 1

Example: If (K1=1) then (R0 ← R1) else if (K2=1) then (R0 ← R2)

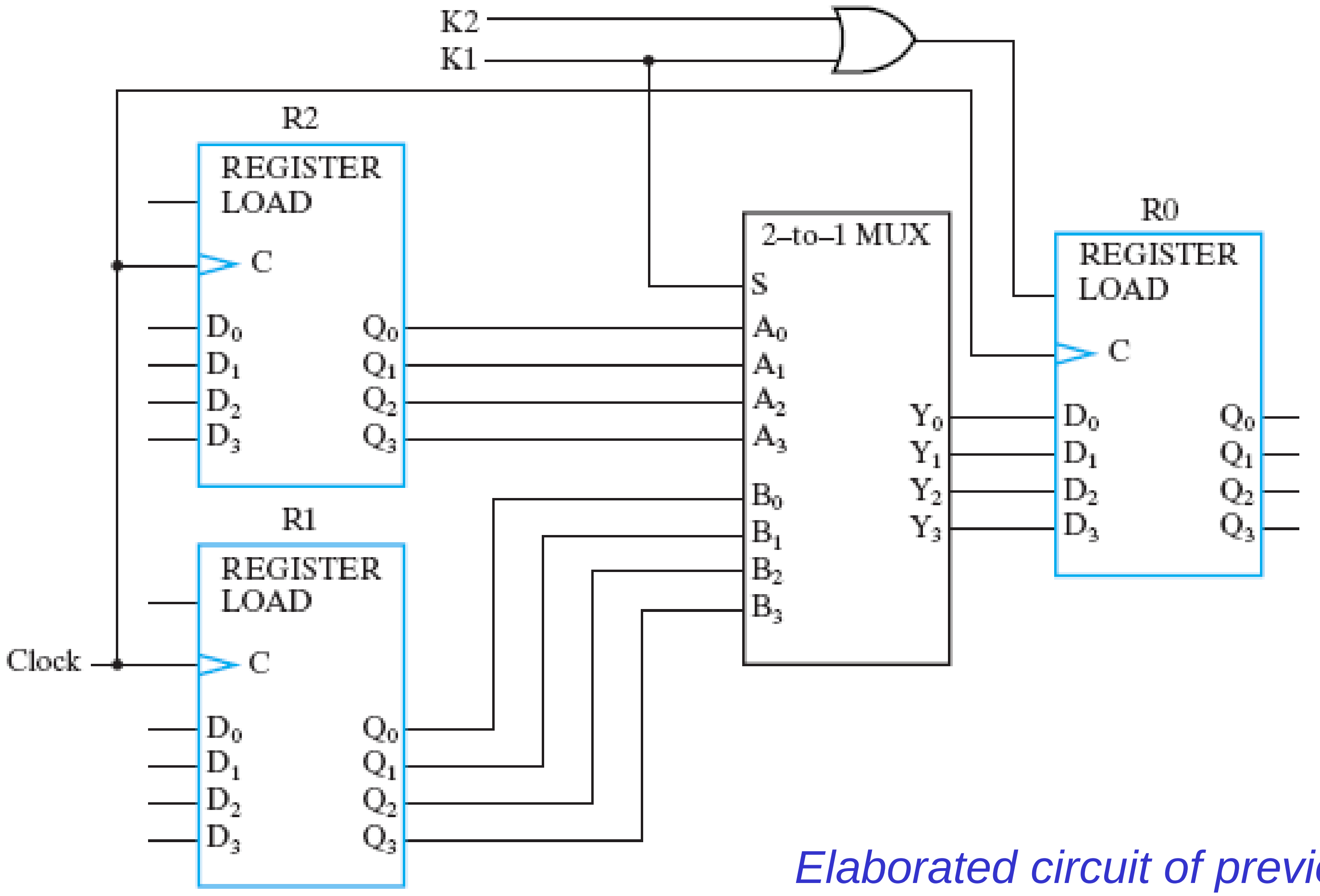
- RTL: K_1:R0←R1, K_1'K_2:R0←R2
  - when K_1=1, R1 loaded into R0 irrespective of K_2
  - when K_1=0 and K_2=1, R2 loaded into R0
  - else no change



n-to-1 transfer

RTL format: controls: register transfer, controls: register transfer, …

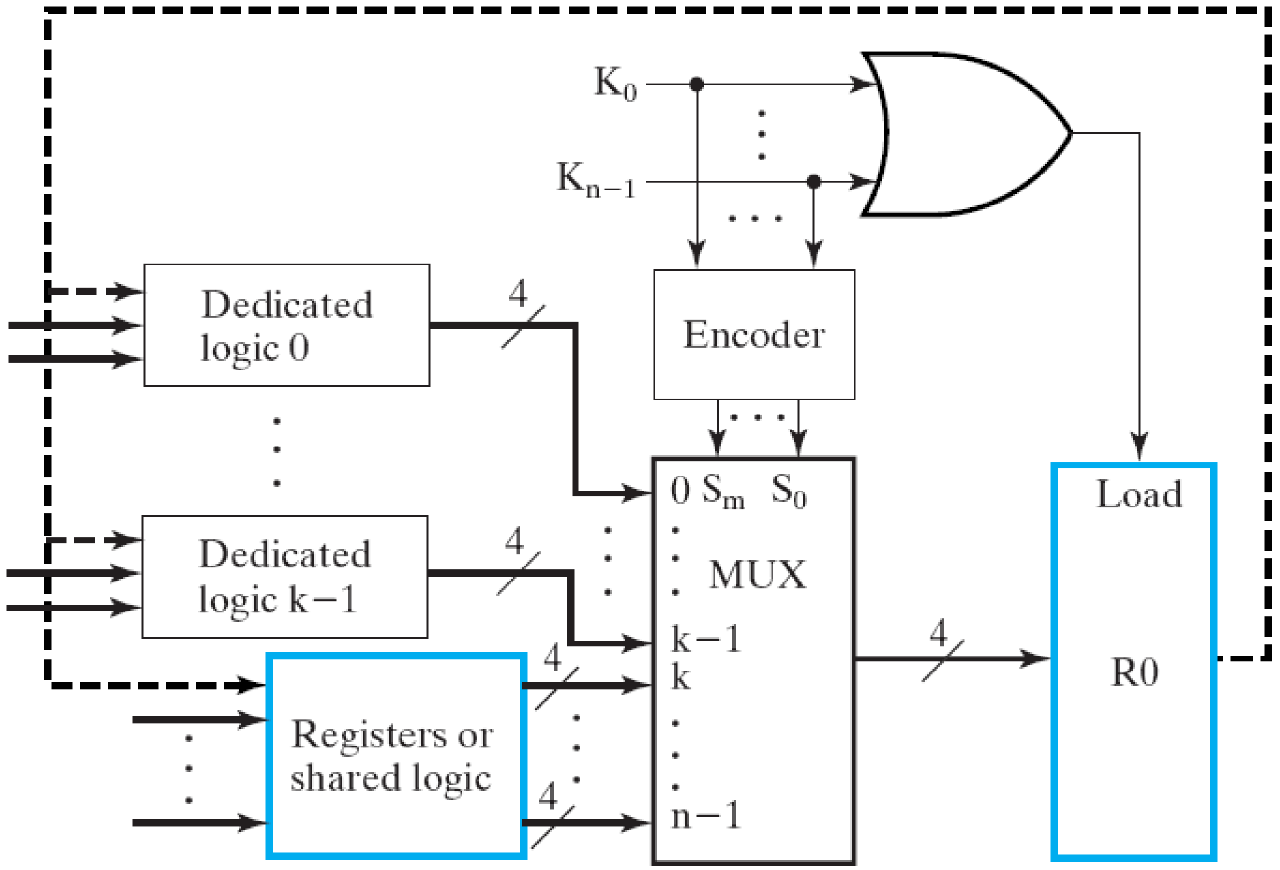
Multiplexer-based Transfers - 2



Elaborated circuit of previous slide

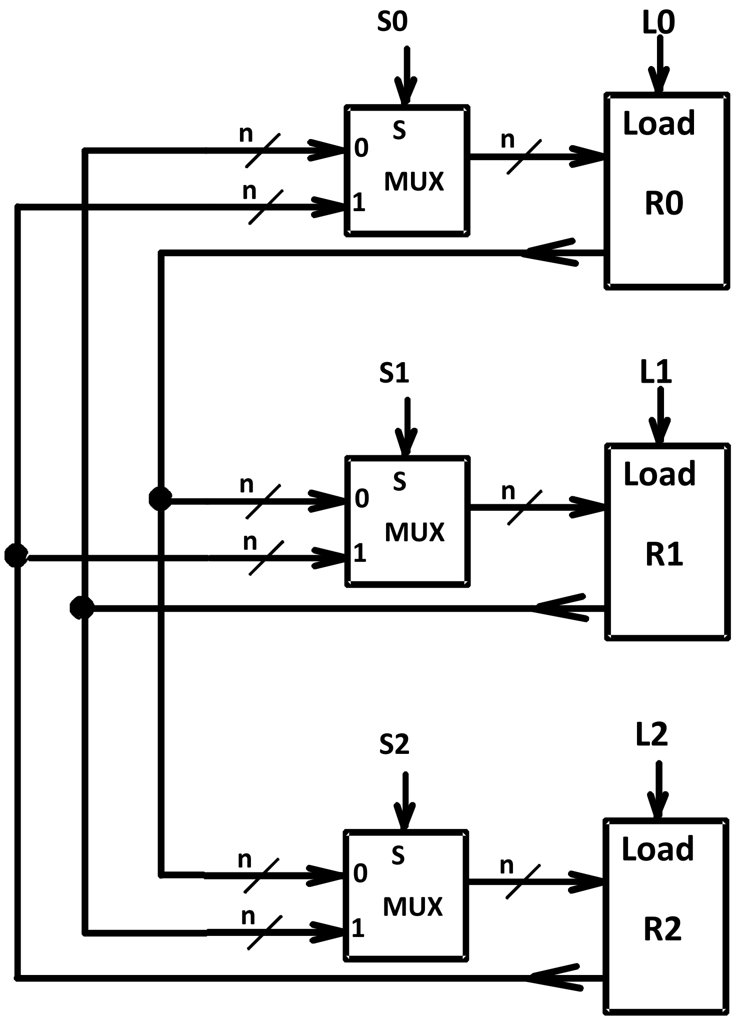
Multiplexer-based Transfers - 3

- Generalization for n sources



Dedicated MUX-Based Transfers

- Multiplexer connected to each register input produces a very flexible transfer structure =>
- Characterize the simultaneous transfers possible with this structure.

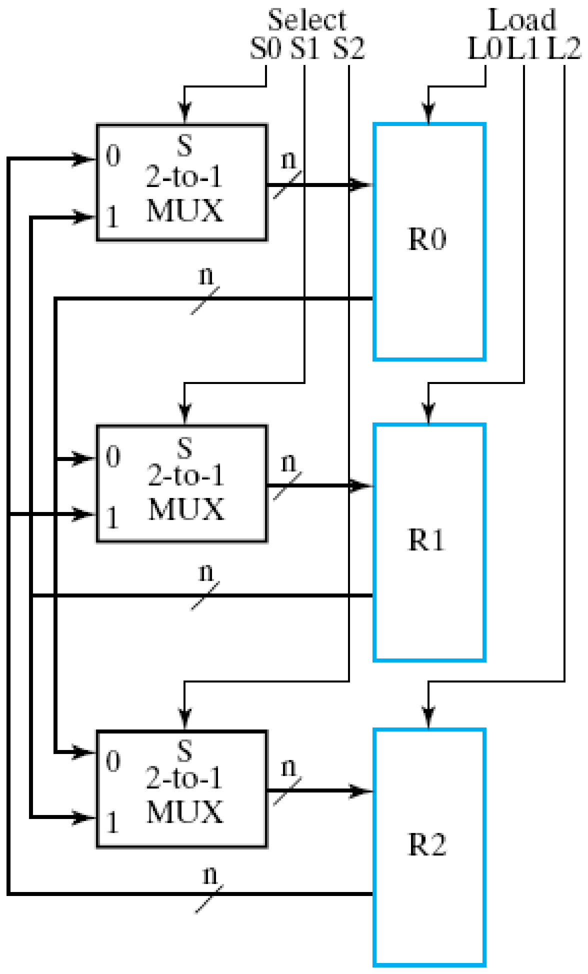


n*(n-to-1) transfer

Bus-Based Transfers - 1

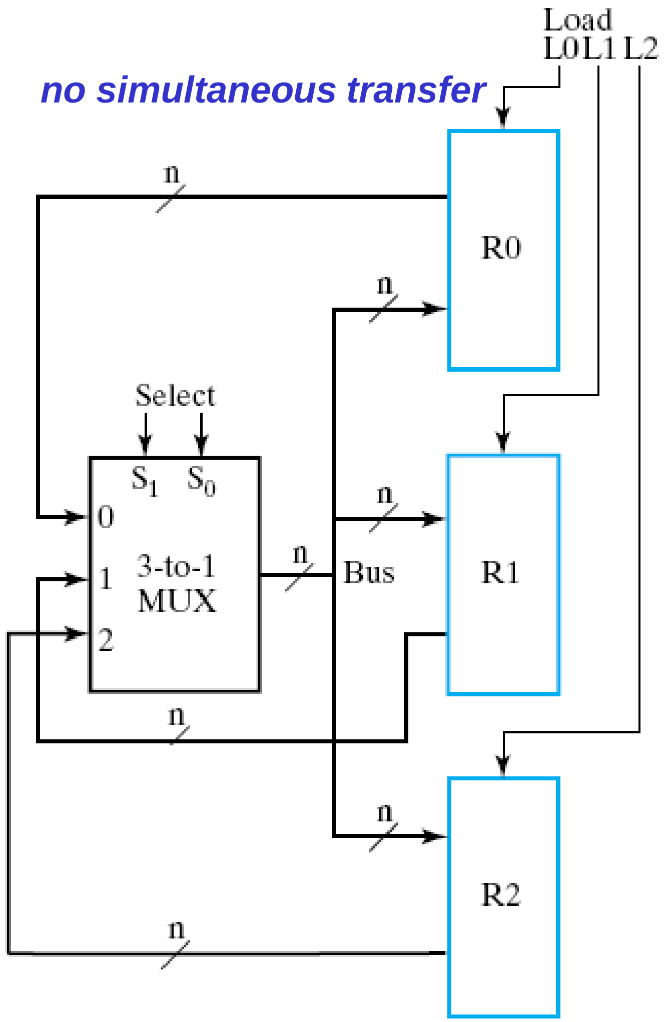
- A typical digital system has many registers
- Path needed to transfer data from one register to another
- Excessive logic and interconnection if each register is to own dedicated multiplexers
- Use shared transfer path - *bus*

Bus-Based Transfers - 2



(a) Dedicated multiplexers

no simultaneous transfer

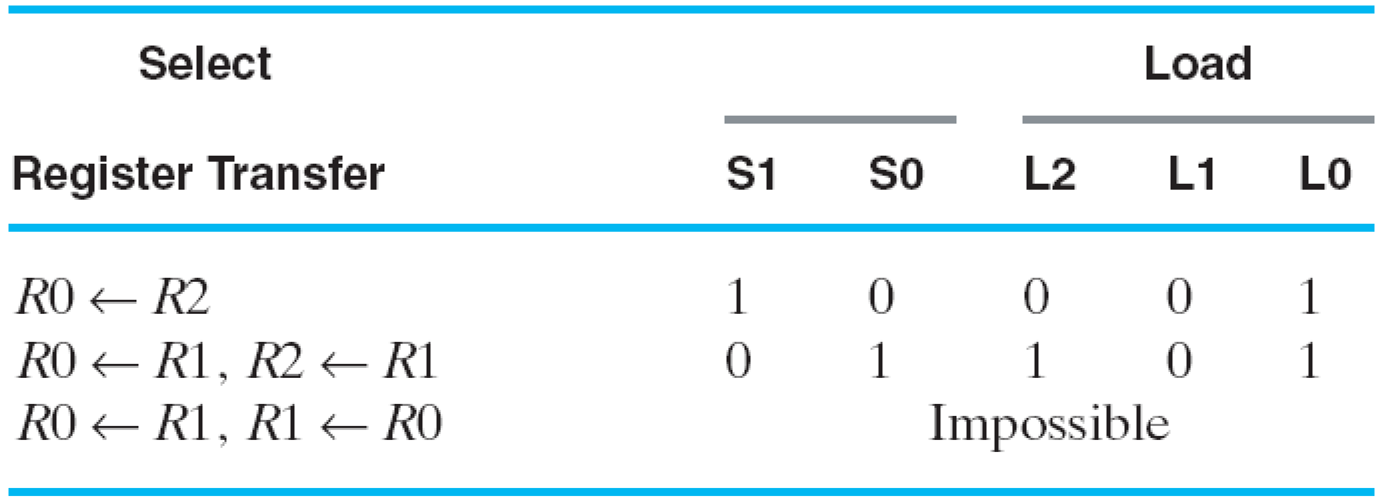


(b) Single Bus

comparison of mux and bus based: number of wires

Bus-Based Transfers - 3

- Select – determines the source register that will provide the multiplexer output
- Load – determines the destination register(s)



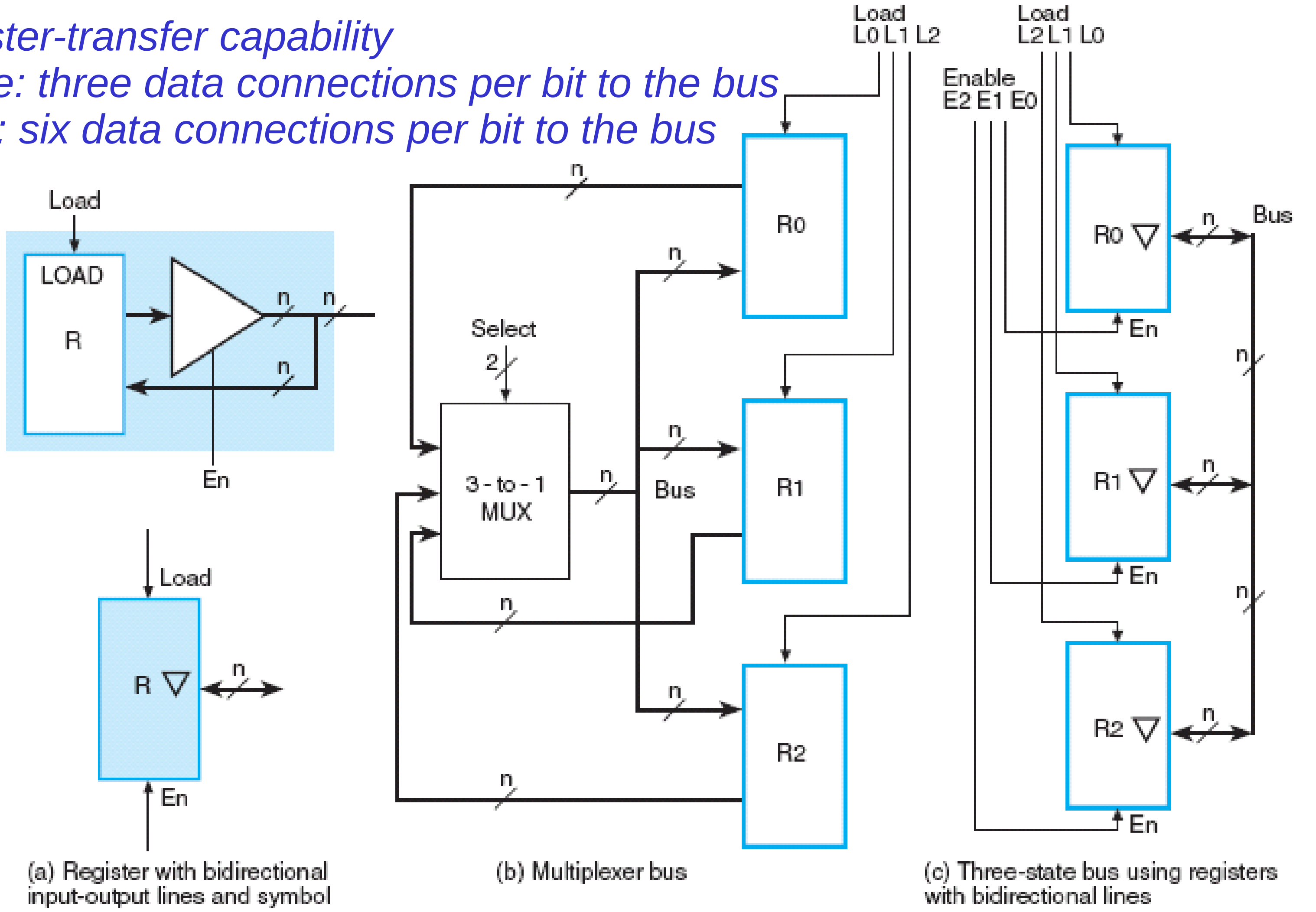
	Select		Load		
Register Transfer	S1	S0	L2	L1	L0
$R0 \leftarrow R2$	1	0	0	0	1
$R0 \leftarrow R1,\ R2 \leftarrow R1$	0	1	1	0	1
$R0 \leftarrow R1,\ R1 \leftarrow R0$	Impossible				

Three-State Bus - 1

- A bus can be constructed with tri-state buffers instead of multiplexers
  - Many tri-state buffer outputs can be combined to form a bit line of a bus, implemented using only one level of logic gates
  - Potential for reductions in number of connections
  - Signals can travel in 2 directions on a three-state bus
  - Multiplexer requires a high fan-in OR, which requires multiple layers of OR gates, introducing more logic and increasing delay
  - Practical way to construct fast buses with many sources

Three-State Bus - 2

Same register-transfer capability
Three-State: three data connections per bit to the bus
Multiplexer: six data connections per bit to the bus



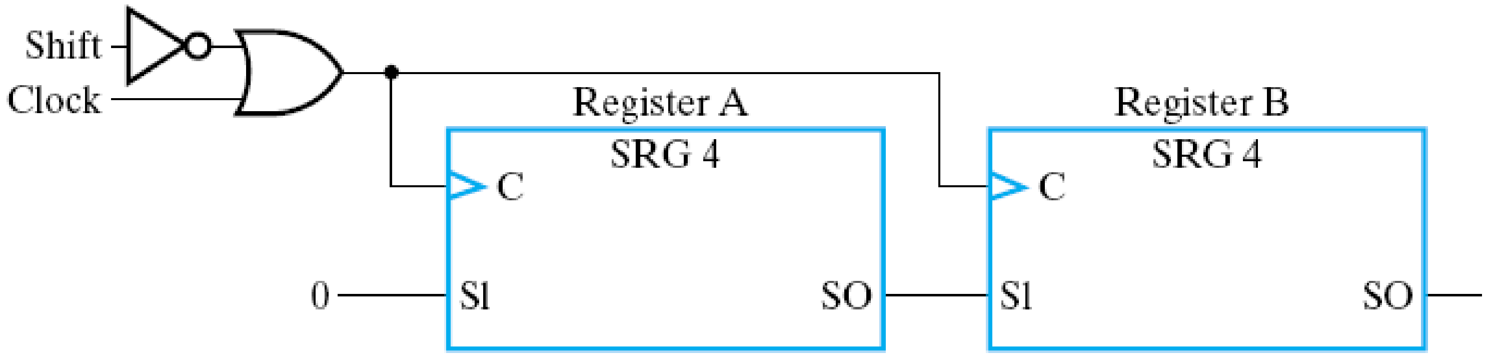
(a) Register with bidirectional input-output lines and symbol
(b) Multiplexer bus
(c) Three-state bus using registers with bidirectional lines

Serial vs Parallel Transfer

- Serial transfer
  - info transferred and manipulated one bit at a time
  - by shifting bits out of source register into destination register using shift registers
  - slow, less wires/connections, long distance
- Parallel transfer
  - all bits of the register transferred at the same time
  - fast, more wires/connections, limited distance

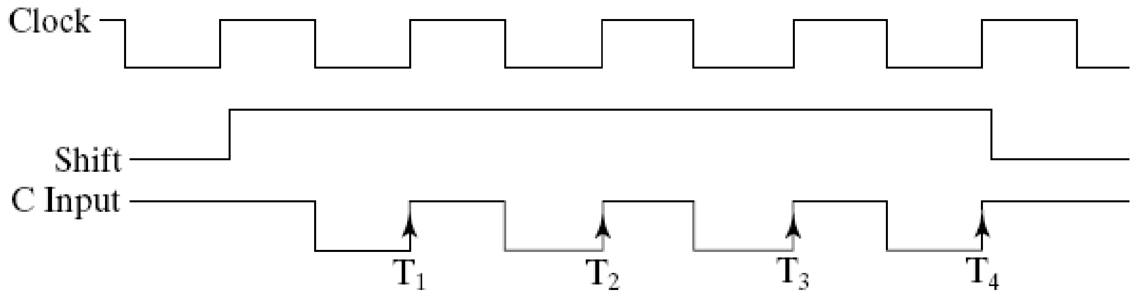
Serial Transfer - 1

- Serial transfer from Reg A to B:
  - While data is transferred to Reg B, Reg A can:
  - receive 0's, other info or its own data
- Shift input controls when and how many times the registers are shifted: clock pass through only when Shift is 1



Serial Transfer - 2

For 4-bit shift register, Shift will be maintained at high for 4 clock pulses

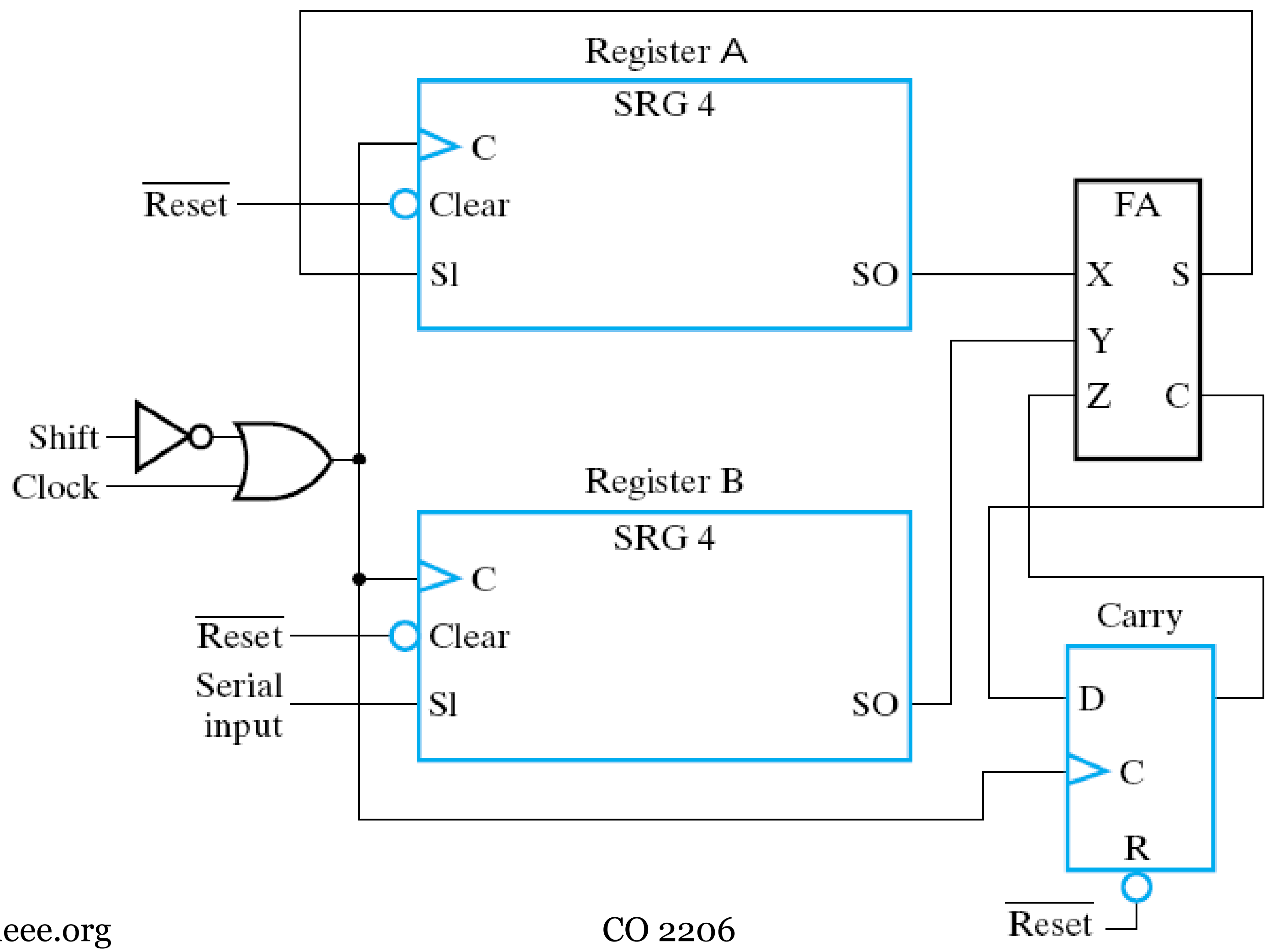


Timing pulse	Shift Register A				Shift Register B			
Initial value	1	0	1	1	0	0	1	0
After T_1	0	1	0	1	1	0	0	1
After T_2	0	0	1	0	1	1	0	0
After T_3	0	0	0	1	0	1	1	0
After T_4	0	0	0	0	1	0	1	1

Serial Microoperation

- Parallel mode
  - All bits transferred/manipulated simultaneously during one clock pulse, thus faster mode of operation
- Serial mode
  - Slower because of the time it takes to transfer/manipulate info in and out of shift registers, one bit at a time
  - Requires less hardware
  - Control signals must be maintained for a period equal to one word time

Serial Addition - 1



Serial Addition - 2

- 2 binary numbers to be added serially stored in 2 shift registers
  - Initially, A holds augend, B holds addend and carry flip-flop cleared to 0
- Bits added sequentially through a single full-adder
  - Carry out of FA is transferred to a D flip-flop.
  - Output of D flip-flop used as input carry for next pair of significant bits (next clock)

Serial Addition - 3

- Sum shifted into A
  - A used for storing augend and sum bits
- SI of B able to receive new binary number while addend shifted out during addition
- Shift control enables registers and flip-flop for a number of clk pulses equal to no. of bits in registers

Serial Adder vs. Parallel Adder

- Registers used
  - Parallel adder - registers with parallel-load
  - Serial adder - uses shift registers
- No. of FA
  - parallel adder – *n* FA for *n* bits binary number
  - serial adder – one FA and a carry flip-flop
- Excluding the registers
  - parallel adder is a combinational circuit
  - whereas serial adder is a sequential circuit

Summary

- Registers store n-bit information, using n flip-flops
- A register has the capability to perform one or more elementary operations such as load, count, add, subtract and shift; these operations are called microoperations
- Register Transfer Language is symbolic description of register transfer operations
- Different types of registers: Shift Registers, Counters
- Register Cell Design: ad hoc or sequential circuit design
- Register Transfer Structures: multiplexer-based, bus-based, three-state bus and others
- Serial Operations: transfer and addition