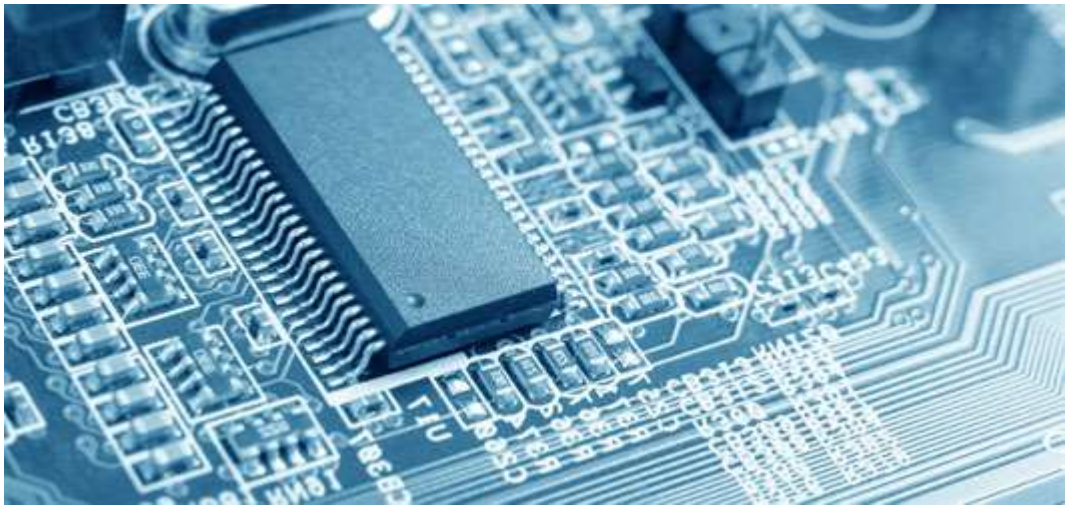
NAZARBAYEV UNIVERSITY
SCHOOL OF SCIENCE AND TECHNOLOGY



ROBT206 – Microcontrollers with Lab

Lecture 20 – Register Cells, Buses

5 April, 2018

Course Logistics

Important Dates and Tasks

Homework #4 due to end of 14 April
7.3, 7.4, 7.5, 7.10, 7.11, 7.12, 7.15, 7.19, 7.21, 7.22, 7.28, 7.37

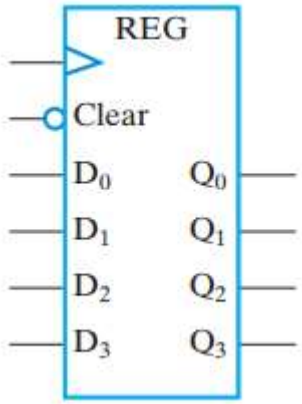
Topics

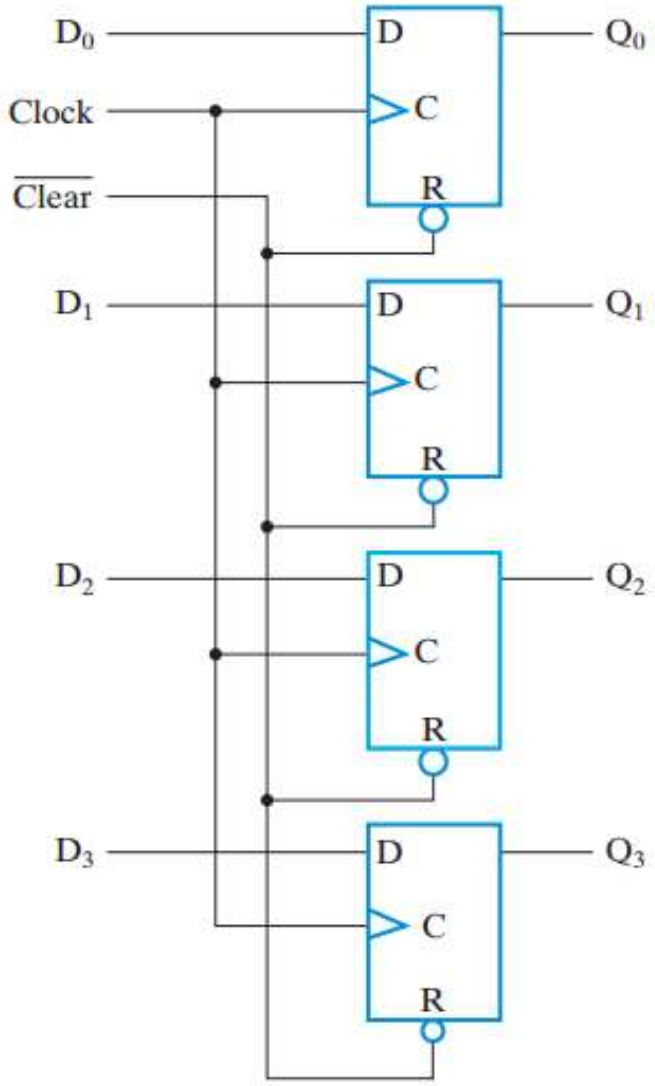
Today's Topics

- Register cell design
- Multiplexer and bus-based transfers for multiple registers
- Serial transfers and micro-operations

Recap: Register with Parallel Load

- Register: Group of Flip-Flops
- Ex: D Flip-Flops
- Holds a Word (Nibble) of Data
- Loads in Parallel on Clock Transition
- Asynchronous Clear (Reset)





Register Cell Design

- Assume that a register consists of identical cells
- Then register design can be approached as follows:
  - Design representative cell for the register
  - Connect copies of the cell together to form the register
  - Applying appropriate "boundary conditions" to cells that need to be different
- Register cell design is the first step of the above process

Register Cell Specifications

- A register
- Data inputs to the register
- Control input combinations to the register
  - **Example 1: Not encoded**
    - Control inputs: **Load, Shift, Add**
    - At most, one of **Load, Shift, Add** is 1 for any clock cycle
      (0,0,0), (1,0,0), (0,1,0), (0,0,1)
  - **Example 2: Encoded**
    - Control inputs: S1, S0
    - All possible binary combinations on S1, S0
      (0,0), (0,1), (1,0), (1,1)

Register Cell Specifications

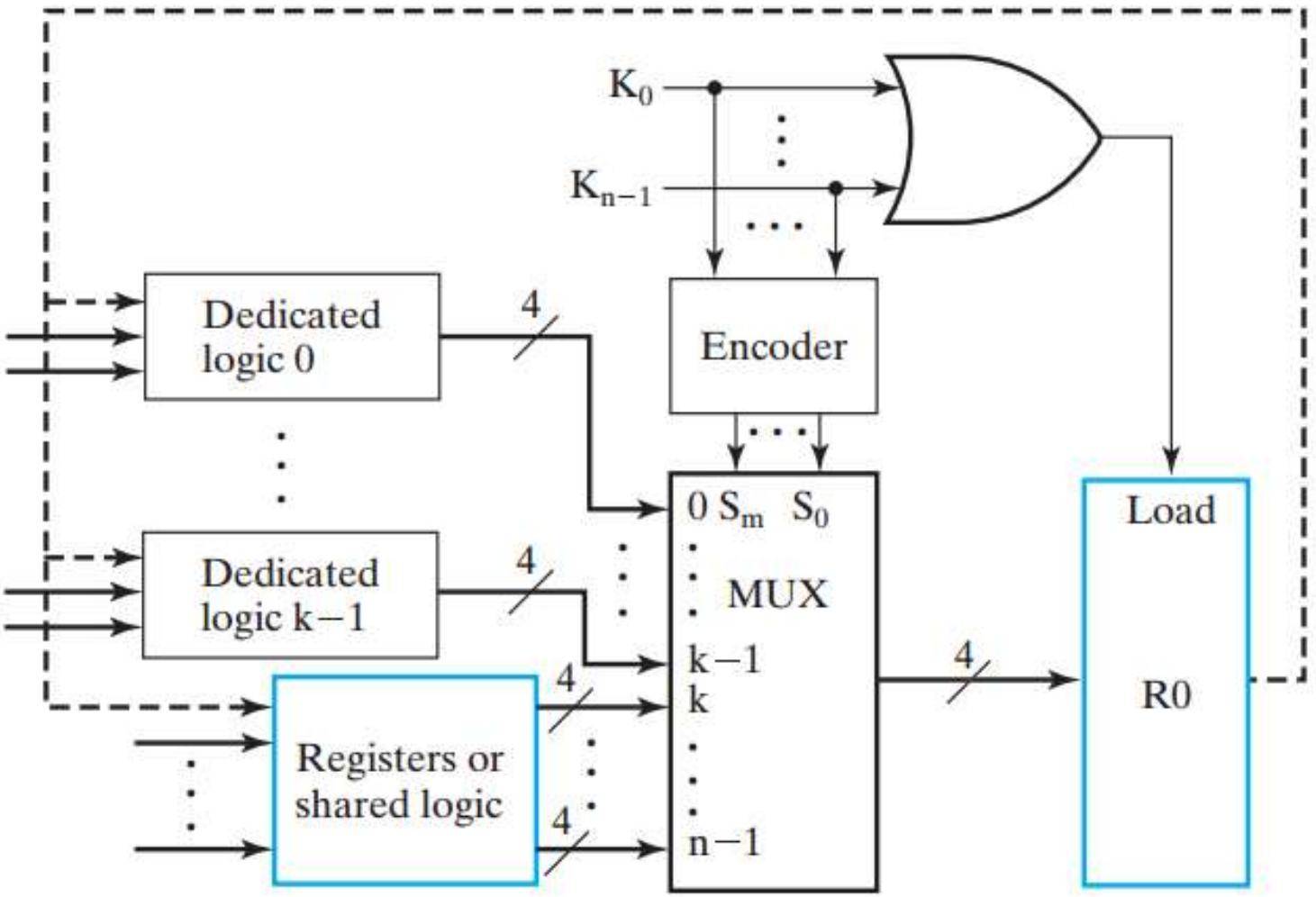
- A set of register functions (typically specified as register transfers)
  - Example:

    Load: $A \leftarrow B$
    Shift: $A \leftarrow srB$
    Add: $A \leftarrow A + B$

- A hold state specification
  - Example:
    - Control inputs: Load, Shift, Add
    - If all control inputs are 0, hold the current register state

Recap: Multiplexer Approach

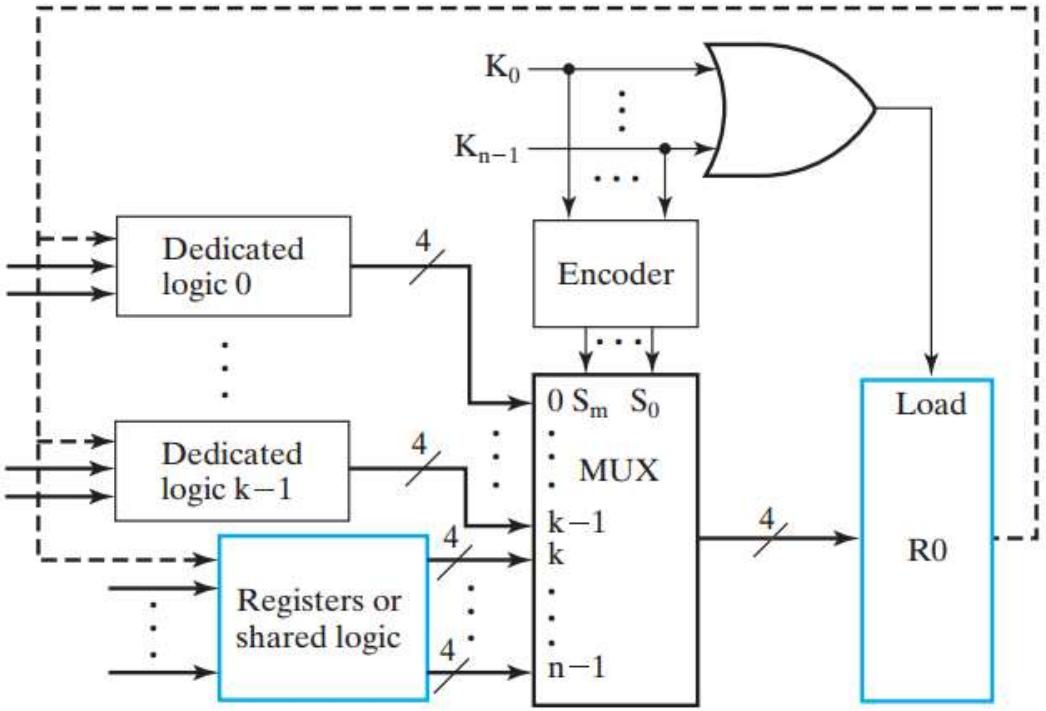
- Uses an n-input multiplexer with a variety of transfer sources and functions



Recap: Multiplexer Approach

- Load enable by OR of control signals $K_0, K_1, \ldots K_{n-1}$
  - assumes no load for 00…0
- Use:
  - **Encoder + Multiplexer (shown) or**
  - **n x 2 AND-OR**

to select sources and/or transfer functions



Example 1: Register Cell Design

- Register A (m-bits) Specification:
  - Data input: B
  - Control inputs (AND, OR, EXOR)
  - Control input combinations (0,0,0), (0,0,1) (0,1,0)(1,0,0)
  - Register transfers:
  - AND: $A \leftarrow A \wedge B$
  - OR: $A \leftarrow A \vee B$
  - EXOR: $A \leftarrow B \oplus A$
  - Hold state: (0,0,0)

- Load Control

  Load = AND+OR + EXOR

Sequential Circuit Design Approach

- Find a state diagram or state table
  - Note that there are only two states with the state assignment equal to the register cell output value
- For optimization:
  - Use K-maps for up to 4 to 6 variables
  - Otherwise, use computer-aided or manual optimization

Example 1 Again

- **State Table:**

Present State A	Next State A(t + 1)						
	(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

- A is a state variable and output,
- AND, EXOR, OR and B are inputs
- Don't care conditions (for AND = OR = EXOR = 1)

Example 1 Continued

- The resulting SOP equation:

$$D_i = A(t+1) = \text{AND}\ A_i B_i + \text{EXOR}\ (A_i \bar{B}_i + \bar{A}_i B_i) + \text{OR}\ (\text{A}_\text{i} + \text{B}_\text{i}) + \overline{\text{AND}}\ \overline{\text{EXOR}}\ \overline{\text{OR}}\ A_i$$

- Note that control variables AND, OR, EXOR can be shared between register cells since they are the same for each cell
- Factor including variables A and B are implemented in each cell

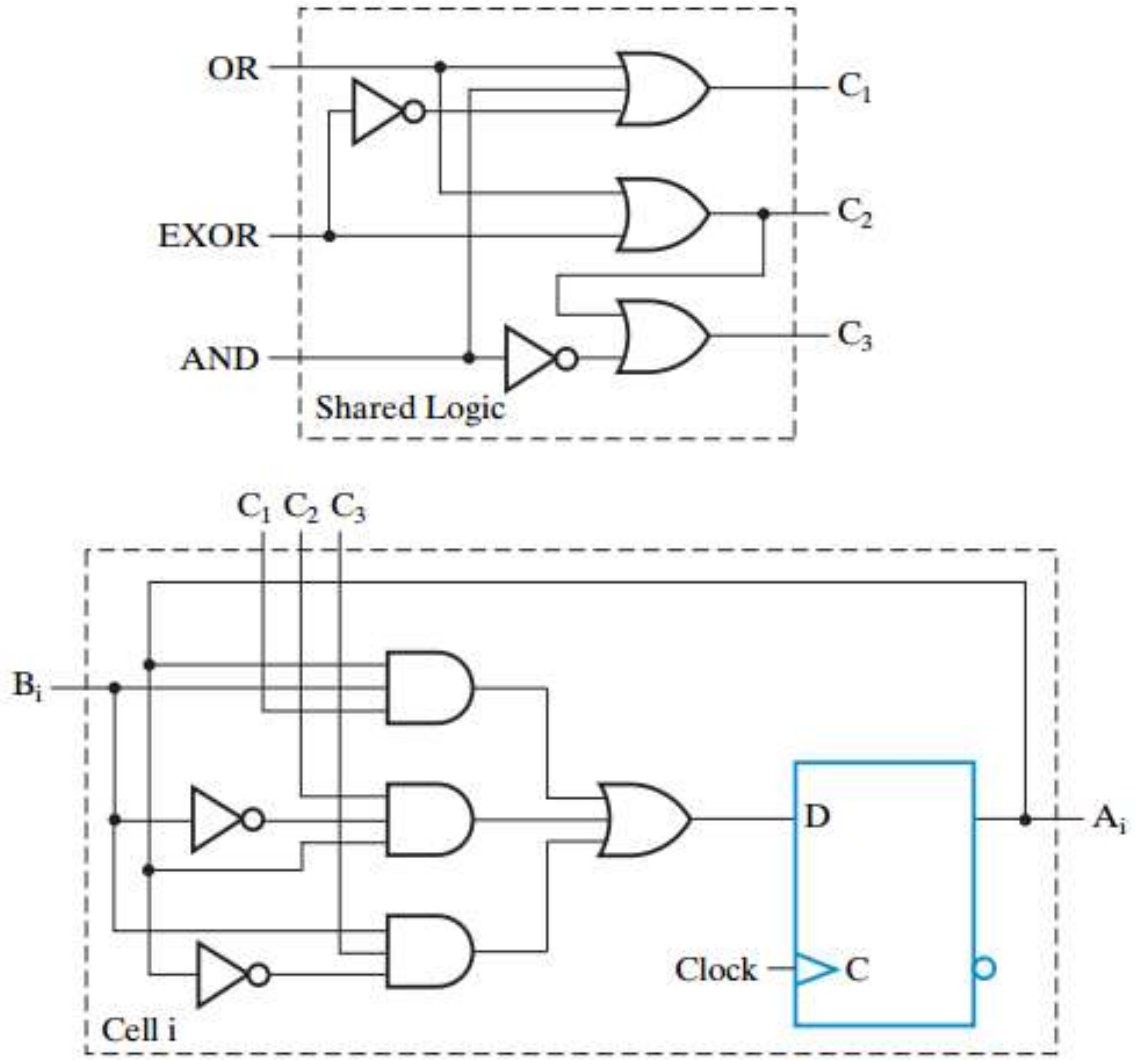
Example 1 Continued

- Separating factor involving condition variables only, the governing equation is rewritten as

$$D_i = (\text{AND} + OR + \overline{\text{AND}}\ \overline{\text{EXOR}}\ \overline{\text{OR}})\ (A_i\ B_i)$$
$$+(\text{EXOR} + OR + \overline{\text{AND}}\ \overline{\text{EXOR}}\ \overline{\text{OR}})(A_i\bar{B}_i) + (EXOR + OR)(\bar{A}_i B_i)$$
$$= (\text{AND} + OR + \overline{\text{EXOR}})\ (A_i\ B_i) + (\text{EXOR} + OR + \overline{\text{AND}}\ )(A_i\bar{B}_i)$$
$$+(EXOR + OR)(\bar{A}_i B_i)$$

- $C_1 = \text{OR} + \text{AND} + \overline{\text{EXOR}}$
- $C_2 = \text{OR} + \text{EXOR}$
- $C_3 = C_2 + \overline{\text{AND}}$
- $D_i = C_1 A_i B_i + C_3 A_i \bar{B}_i + C_2 A_i \bar{B}_i$

Example 1 Continued

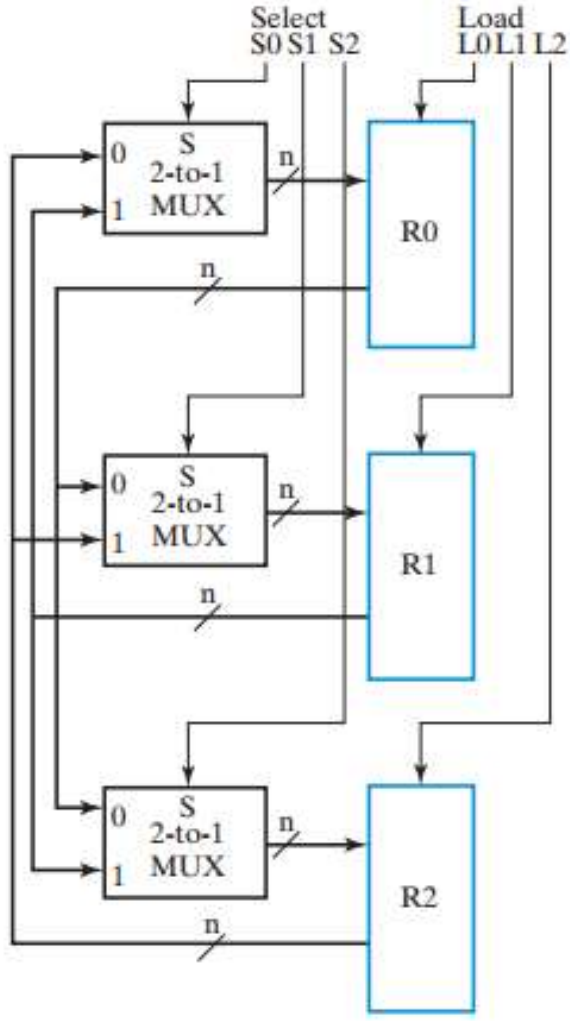


Multiplexer and Bus-Based Transfers for Multiple Registers

- Multiplexer dedicated to each register
- Shared transfer paths for registers
  - A shared transfer object is a called a *bus* (Plural: *buses*)
- Bus implementation using:
  - multiplexers
  - three-state nodes and drivers
- In most cases, the number of bits is the length of the receiving register

Dedicated MUX-Based Transfers

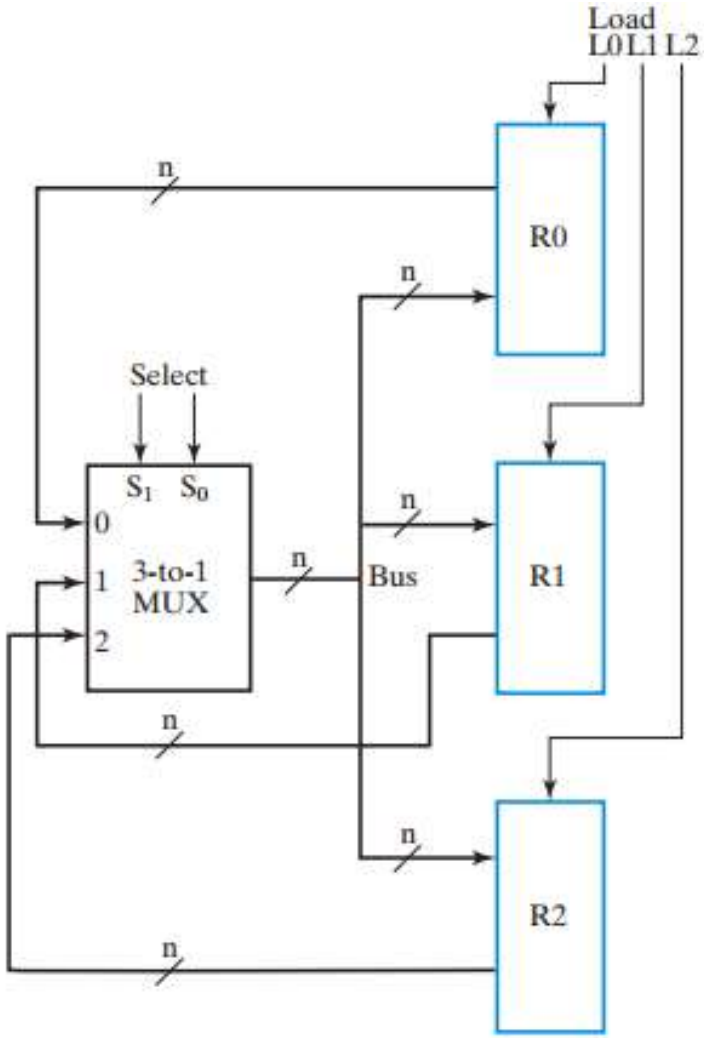
- Multiplexer connected to each register input produces a very flexible transfer structure =>
- Characterize the simultaneous transfers possible with this structure.



(a) Dedicated multiplexers

Multiplexer Bus

- A single bus driven by a multiplexer lowers cost, but limits the available transfers =>
- Characterize the simultaneous transfers possible with this structure.

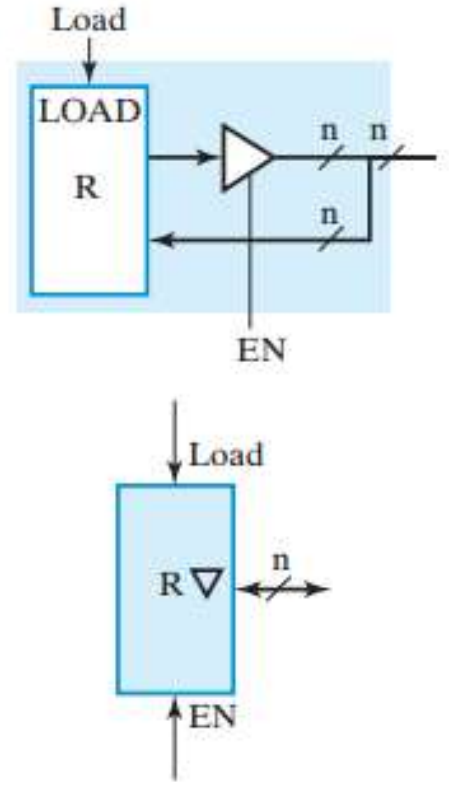


(b) Single bus

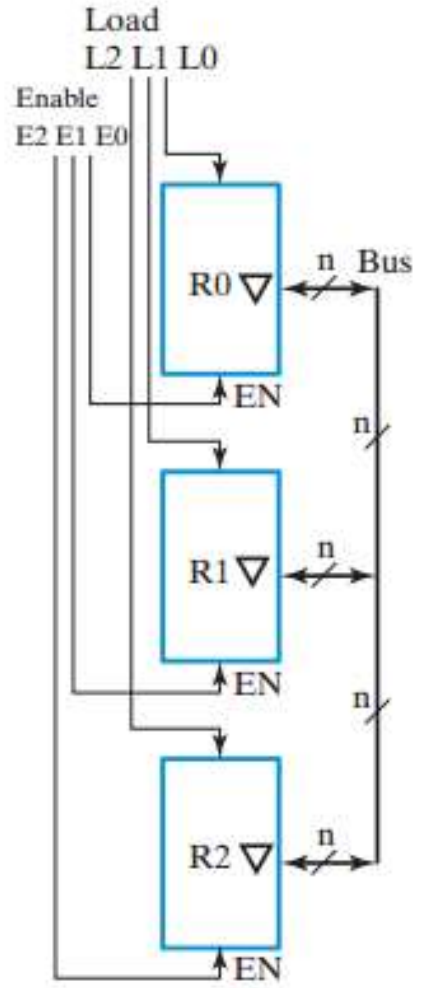
Register Transfer	Select S1	Select S0	Load L2	Load L1	Load L0
$R0 \leftarrow R2$	1	0	0	0	1
$R0 \leftarrow R1, R2 \leftarrow R1$	0	1	1	0	1
$R0 \leftarrow R1, R1 \leftarrow R0$	Impossible				

Three-State Bus

- The 3-input MUX can be replaced by a 3-state node (bus) and 3-state buffers.
- Cost is further reduced, but transfers are limited
- Characterize the simultaneous transfers possible with this structure.



(a) Register with bidirectional input–output lines and symbol



(c) Three-state bus using registers with bidirectional lines

Serial Transfers and Micro-operations

- Serial Transfers
  - Used for "narrow" transfer paths
  - Example 1: Telephone or cable line
    - Parallel-to-Serial conversion at source
    - Serial-to-Parallel conversion at destination

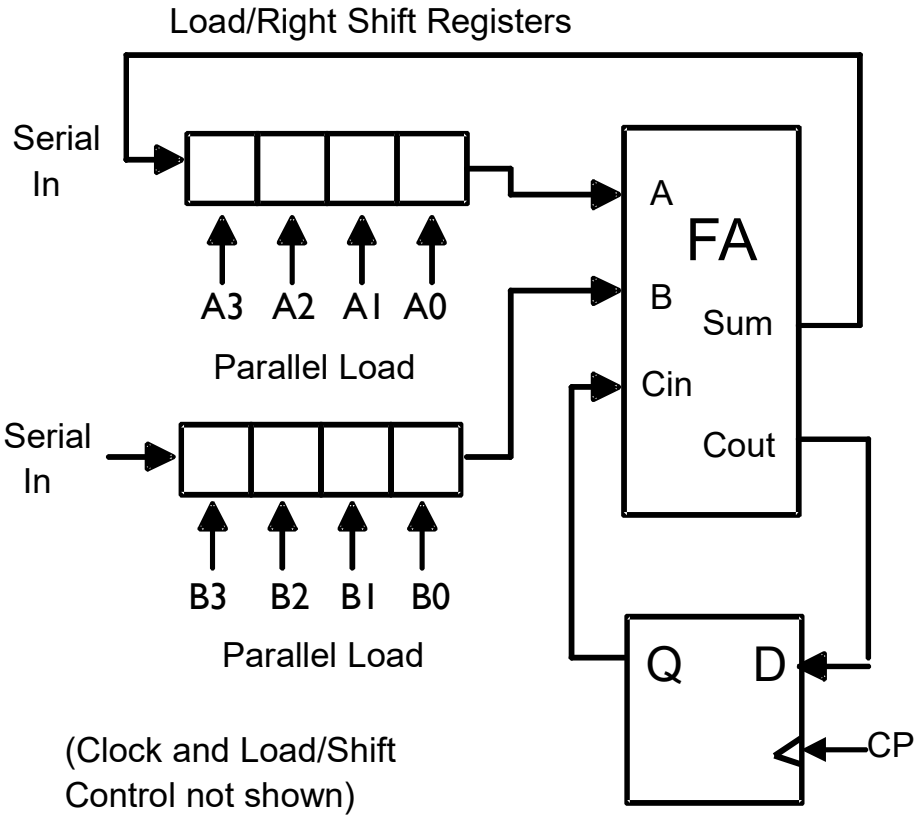
- Serial micro-operations
  - Example: Addition

Serial Micro-operations

- By using two shift registers for operands, a full adder, and a flip flop (for the carry), we can add two numbers serially, starting at the least significant bit.
- Serial addition is a low cost way to add large numbers of operands, since a "tree" of full adder cells can be made to any depth, and each new level doubles the number of operands.
- Other operations can be performed serially as well, such as parity generation/checking or more complex error-check codes.
- Shifting a binary number left is equivalent to multiplying by 2.
- Shifting a binary number right is equivalent to dividing by 2.

Serial Adder

- The circuit shown uses two shift registers for operands A(3:0) and B(3:0).
- A full adder, and one more flip flop (for the carry) is used to compute the sum.
- The result is stored in the A register and the final carry in the flip-flop



- With the operands and the result in shift registers, a tree of full adders can be used to add a large number of operands. Used as a common digital signal processing technique.

Any Questions?