Digital Logic Circuit (SE273 – Fall 2020)

Lecture 11: Register Transfer Level (RTL) Design

Jaesok Yu, Ph.D. (jaesok.yu@dgist.ac.kr)

Assistant Professor

Department of Robotics Engineering, DGIST

Goal of this lecture

- Learn about register transfer operations
  - Microoperation
  - Register transfer language
  - Symbolic form of data transfer

Register Transfer Operations

Register Transfers

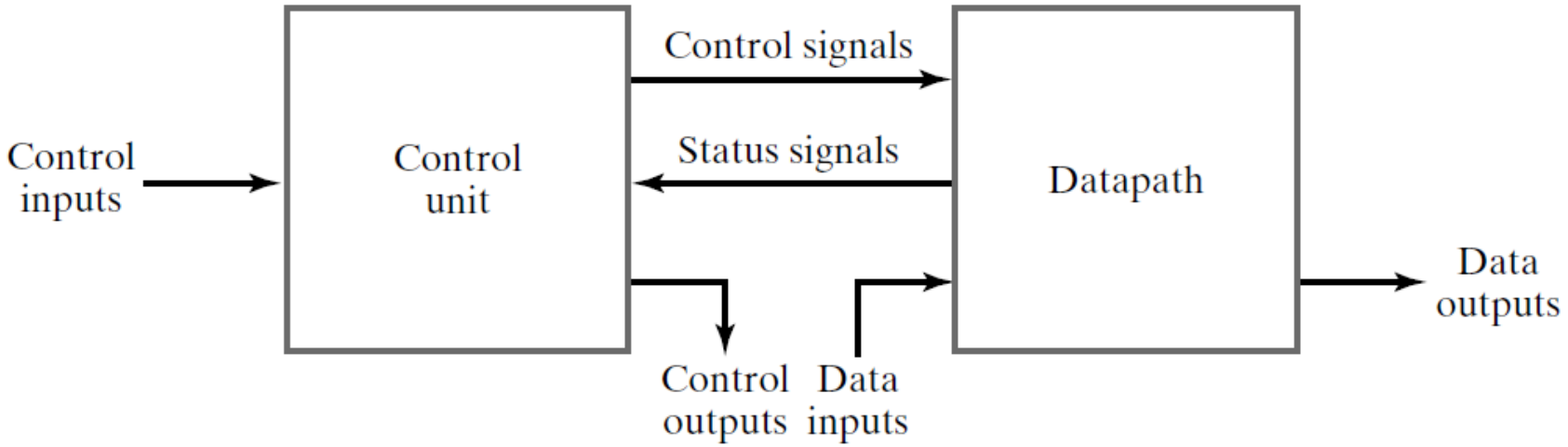
- In digital circuit design, **register-transfer level (RTL)** is a design abstraction which models a synchronous digital circuit in terms of the flow of digital signals (data) between hardware registers, and the logical operations performed on those signals.

- Register-transfer-level abstraction is used in hardware description languages (HDLs) like Verilog and VHDL to create high-level representations of a circuit, from which lower-level representations and ultimately actual wiring can be derived. Design at the RTL level is typical practice in modern digital design.

Register Transfers

- A digital system is a sequential circuit made up of interconnected FFs and logic gates
  - To design a large system with state tables is very difficult
  - Thus, as we have discussed, the system needs to be partitioned into sub-systems or modules

- There are two types of modules: **datapath** and **control unit**



Register Transfer Operations

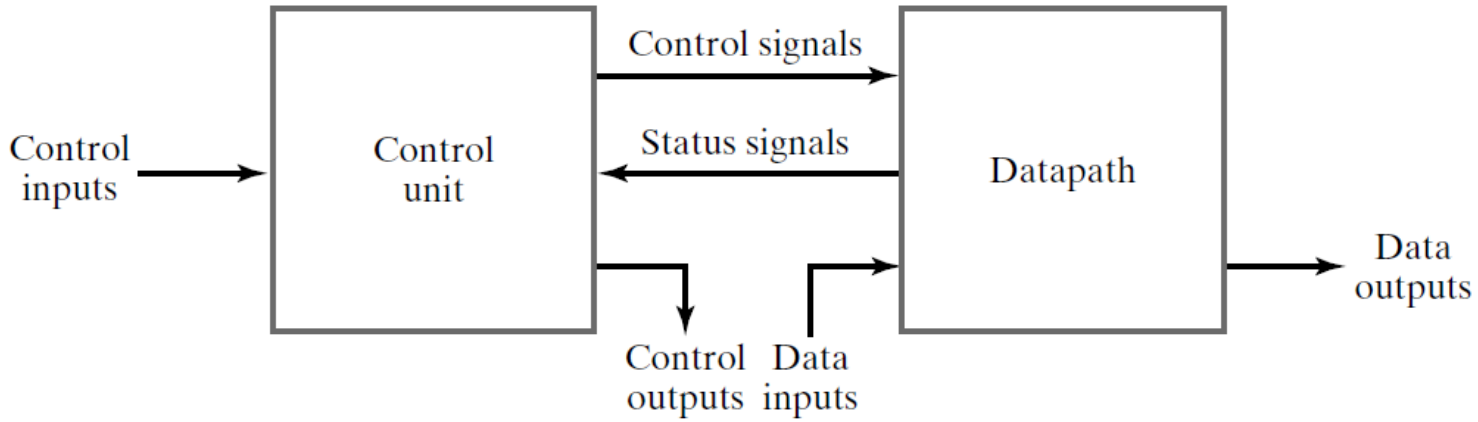
- The movement of the data stored in registers and the processing on the data are referred to as "register transfer operations"
  - It can be specified by three basic components
  - A register is capable of some **elementary operations**: load, count, add, subtract, and shift

1. the set of registers in the system
2. the operations that are performed on the data stored in the registers
3. the control that supervises the sequence of operations in the system

Interaction btw Datapath and Control Unit

- There are two types of modules: **datapath** and **control unit**
  - **Datapath**: do data-processing operations
  - **Control unit**: determines the sequence of those operations

- These modules may interact with other parts of a digital system
  - Memory unit
  - Input-output device



Microoperation

- An elementary operation performed on data is called **microoperation**
  - Loading contents of one register to another
  - Adding the contents of two registers
  - Incrementing the contents of a register

- A microoperation is usually, but not always, performed in parallel on a vector of bits during one clock cycle
  - The result of the microoperation may replace the previous binary data in the register
  - The result may be transferred to another register, leaving the previous data unchanged

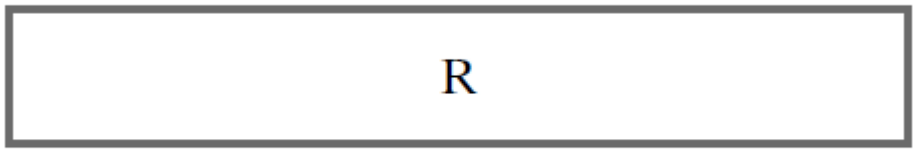
Register Transfer Language (RTL)

- It represents registers and specify the operations on their contents
  - RTL uses a set of expressions and statements that resemble statements in HDLs and programming languages

- Registers in a digital system
  - Address Register (AR)
  - Program Counter (PC)
  - Instruction Register (IR)
  - R2: simple for register 2

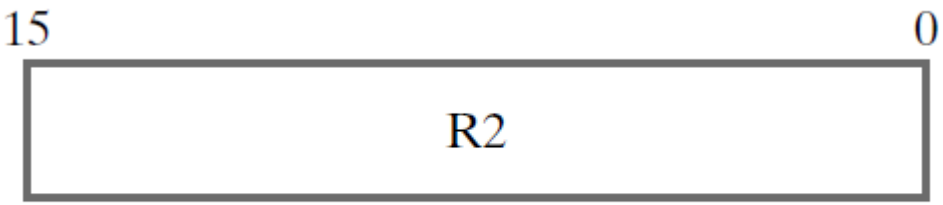
Block Diagram of Registers

- The most common way to represent a register is by a rectangular box
  - We put the register name within a box
  - Individual bit position is normally marked by “little-endian”

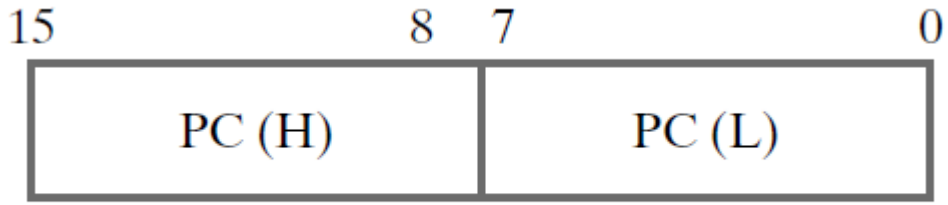


Register R

Individual bits of 8-bit register



Numbering of 16-bit register



Two-part 16-bit register

Symbolic Form for Data Transfers

- It can be represented by the replacement operator (←)
  - Transfer of the contents of register R1 (source) into R2 (destination)
  - Only the contents of the destination register, *R*2, change

$$R2 \leftarrow R1$$

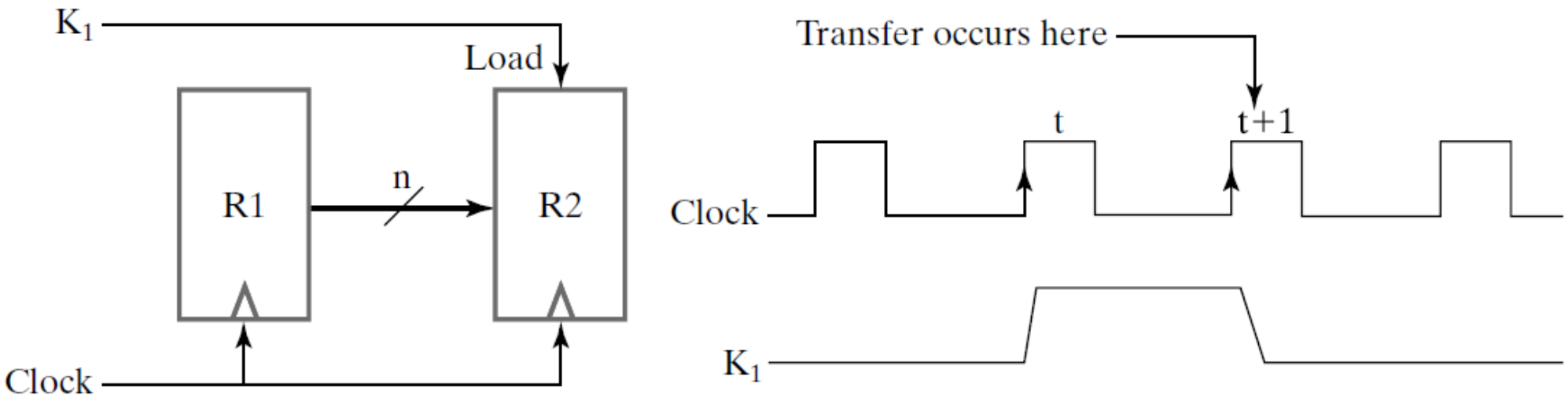
- One may want a transfer to happen at a specific clock cycle
  - We assign specific values of the control signal (say K_1)
  - It can be any Boolean function

$$K_1: R2 \leftarrow R1$$

Hardware Support for Data Transfers

- There should be hardware support for each data transfer operations
  - Register *R*2 has a load control input that is activated by K_1

$$K_1: R2 \leftarrow R1$$



Basic Symbols for Register Transfers

- Note that clock is not included in register-transfer statements
- We may write multiple register transfers in a single statement that are executed at the same time

$$K_3: R2 \leftarrow R1,\ R1 \leftarrow R2$$ **(Exchange 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, VHDL, Verilog Symbols for Register Transfers

- Different notation is used in each of the three languages

Operation	Text RTL	VHDL	Verilog
Combinational assignment	=	<= (concurrent)	assign = (nonblocking)
Register transfer	←	<= (concurrent)	<= (nonblocking)
Addition	+	+	+
Subtraction	–	–	–
Bitwise AND	∧	and	&
Bitwise OR	∨	or	|
Bitwise XOR	⊕	xor	^
Bitwise NOT	¯ (overline)	not	~
Shift left (logical)	sl	sll	<<
Shift right (logical)	sr	srl	>>
Vectors/registers	*A*(3:0)	*A*(3 down to 0)	*A*[3:0]
Concatenation	||	&	{ , }

Microoperations

- Frequently used microoperations
  - **Transfer** : transfer binary data from one register to another
  - **Arithmetic**: perform arithmetic on data in registers
  - **Logic**: perform bit manipulation on data in registers
  - **Shift**: shift data in registers

Arithmetic Microoperations

- Basic arithmetic microoperations are add, subtract, increment, decrement, and complement
  - The following statement specifies an add operation

$$R0 \leftarrow R1 + R2$$

It requires three registers and
a combinational component that performs addition

  - For subtraction, we may utilize 2s complement subtraction

$$R0 \leftarrow R1 + \overline{R2} + 1$$

Arithmetic Microoperations

- List of some arithmetic operations

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

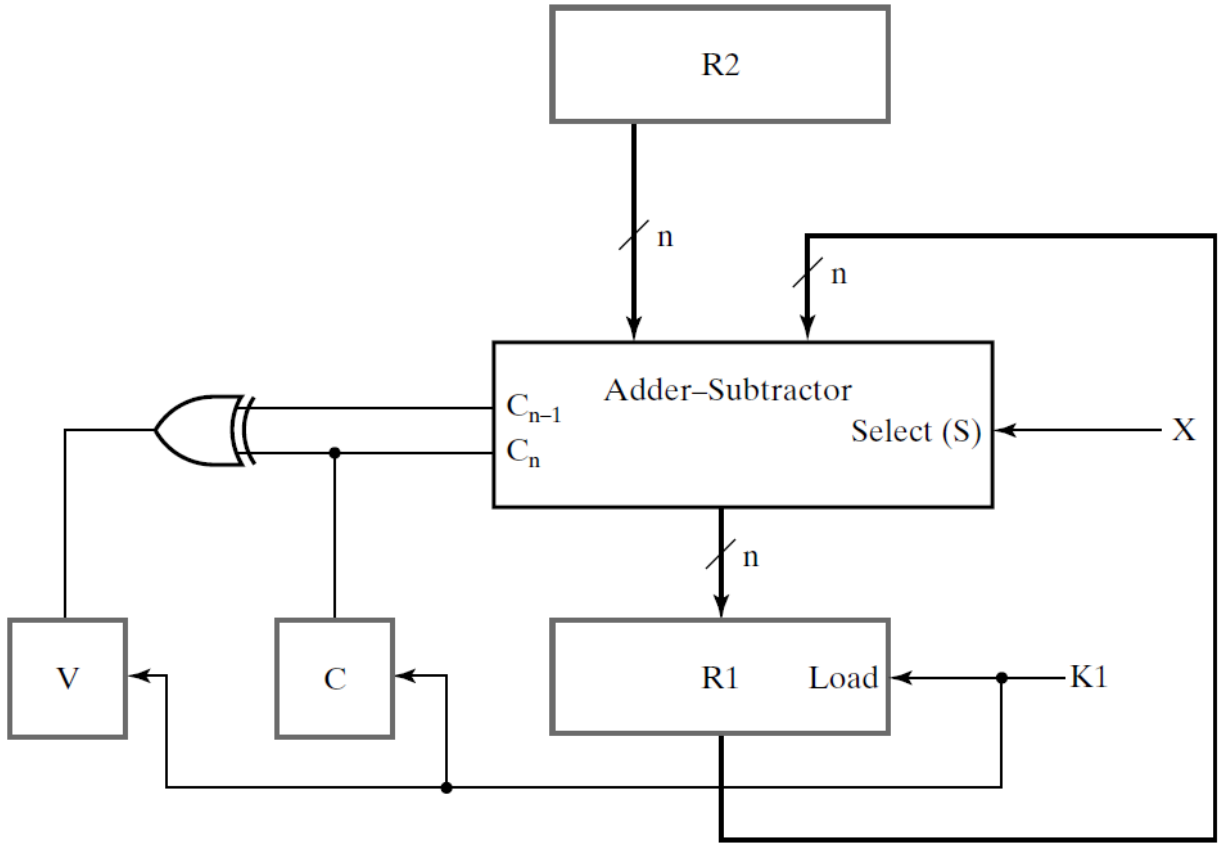
Which hardware module do we need to design?

Arithmetic Microoperations

- There is a direct relationship btw statements written in register-transfer notation and the registers and digital functions

$$\overline{X}K_1: \quad R1 \leftarrow R1 + R2$$

$$XK_1: \quad R1 \leftarrow R1 + \overline{R2} + 1$$



Logical Microoperations

- Useful in manipulating the bits stored in a register
  - The four basic logic operations are as follows:

Symbolic Designation	Description
$R0 \leftarrow \overline{R1}$	Logical bitwise NOT (1s 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)

Logical Microoperations

- The + symbol has two meanings
  - If '+' appears in a microoperation, it denotes addition
  - If '+' appears in a control or Boolean function, it denotes OR

$$(K_1 + K_2): \quad R1 \leftarrow R2 + R3,\ R4 \leftarrow R5 \vee R6$$

- The logic microoperations can change bit values, clear a group of bits, or insert new bit values into a register

10101101 10101011	$R1$	(data)
00000000 11111111	$R2$	(mask)
00000000 10101011	$R1 \leftarrow R1 \wedge R2$	

Logical Microoperations

- The OR microoperation is used to set one or more bits in a register

10101101 10101011	$R1$	(data)
11111111 00000000	$R2$	(mask)
11111111 10101011	$R1 \leftarrow R1 \vee R2$	

- The XOR microoperation can be used to complement one or more bits in a register

10101101 10101011	$R1$	(data)
11111111 00000000	$R2$	(mask)
01010010 10101011	$R1 \leftarrow R1 \oplus R2$	

Shift Microoperations

- They are used for lateral movement of data
  - The contents of a source register can be shifted either right or left
  - A *left shift* is toward the most significant bit, and a *right shift* is toward the least significant bit
  - The destination register for a shift microoperation may be the same as or different from the source register

$$R0 \leftarrow \text{sr } R0, \qquad R1 \leftarrow \text{sl } R2$$

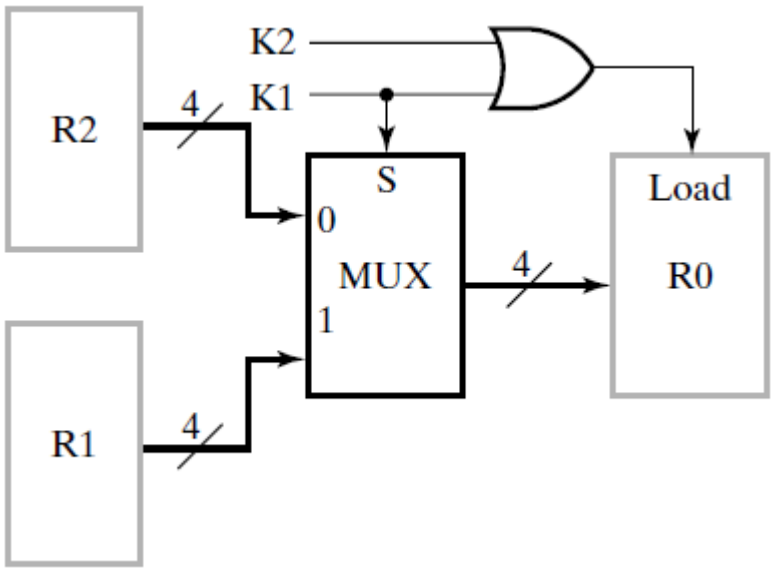
Type	Symbolic Designation	Eight-Bit examples	
		Source *R2*	After Shift: Destination *R1*
Shift left	$R1 \leftarrow \text{sl } R2$	10011110	00111100
Shift right	$R1 \leftarrow \text{sr } R2$	11100101	01110010

Multiplexer-Based Transfers

- There are occasions when a register receives data from two or more different sources at different times

$$\text{if } (K_1 = 1) \text{ then } (R0 \leftarrow R1) \text{ else if } (K_2 = 1) \text{ then } (R0 \leftarrow R2)$$

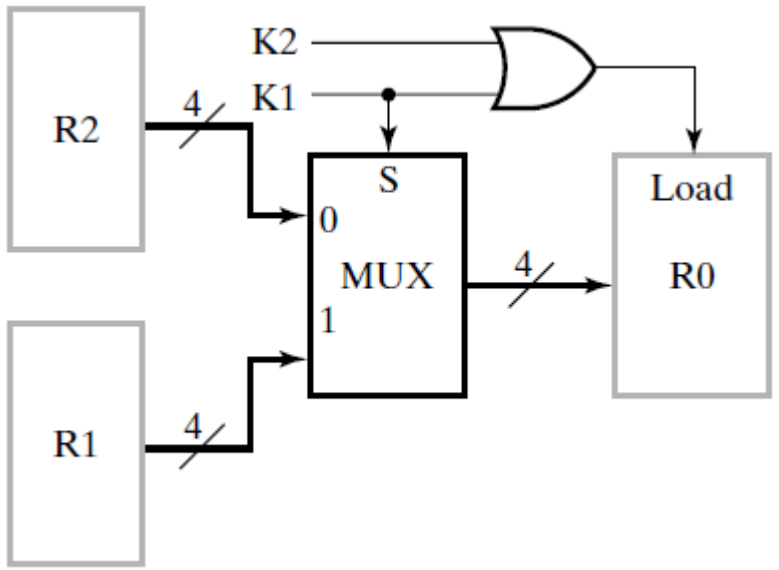
$$K_1: R0 \leftarrow R1, \quad \overline{K}_1 K_2: R0 \leftarrow R2$$

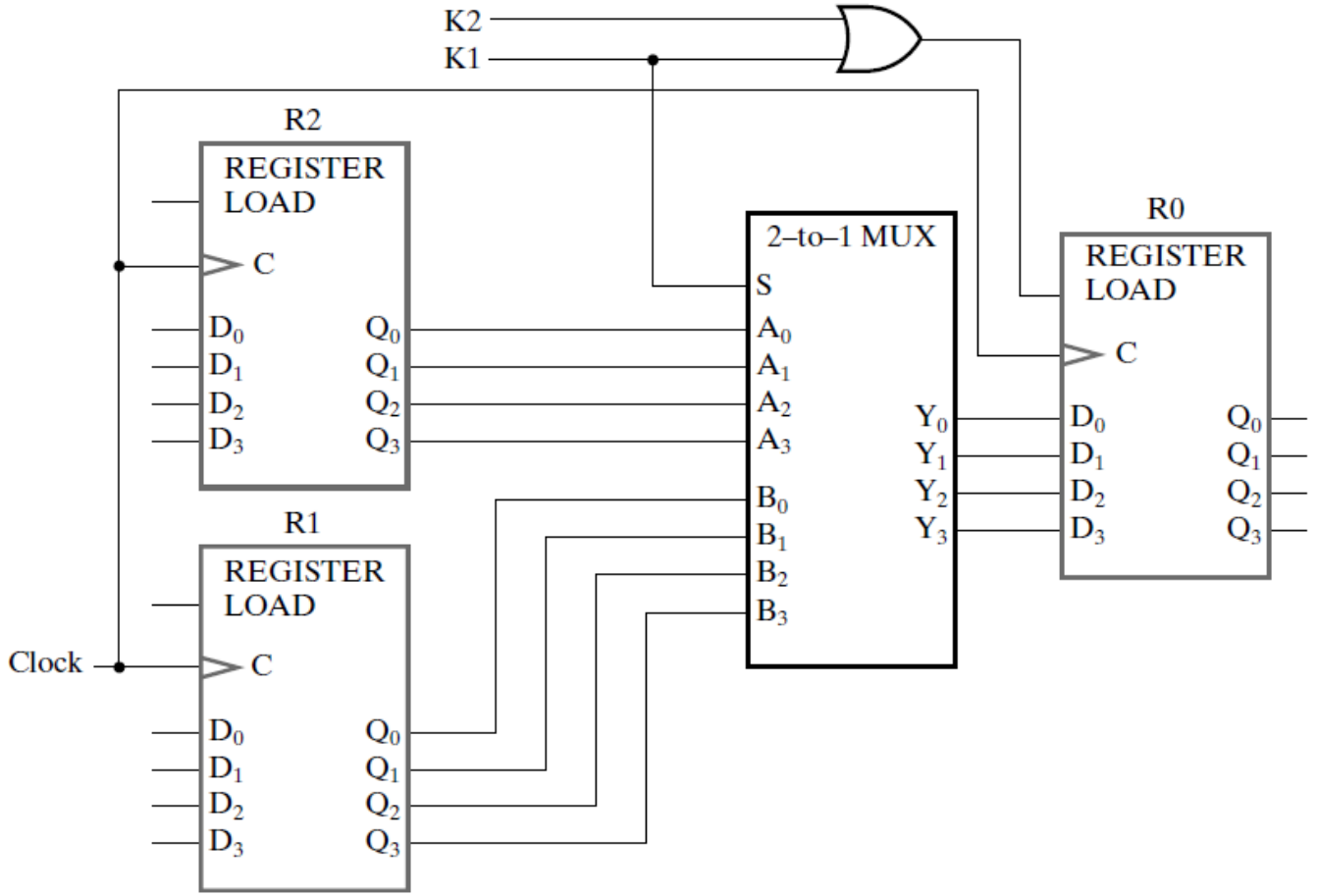


Multiplexer-Based Transfers

- There are occasions when a register receives data from two or more different sources at different times

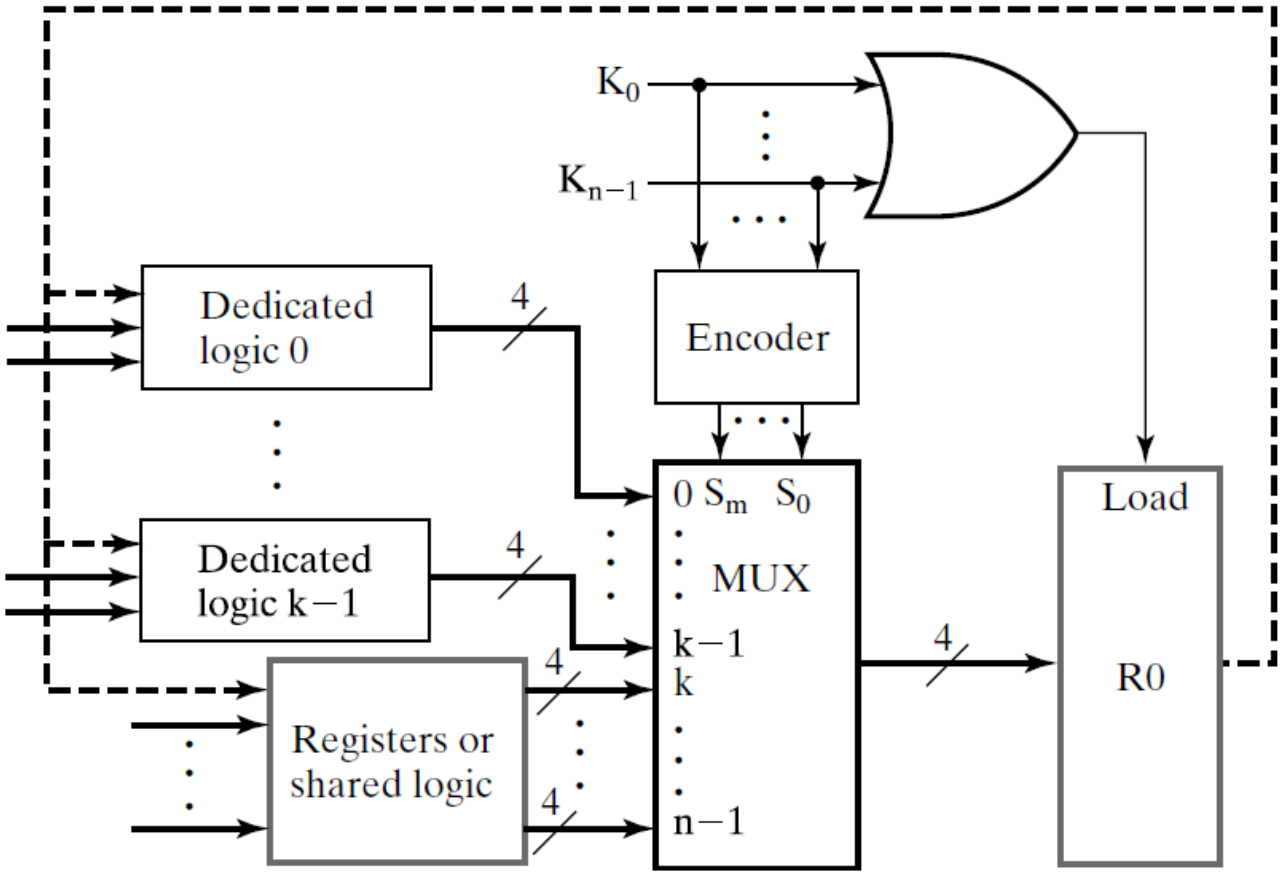
$K_1: R0 \leftarrow R1, \quad \overline{K}_1 K_2: R0 \leftarrow R2$





Multiplexer Selection for Multiple Sources

- The preceding example can be generalized by allowing multiplexer to have ‘n’ sources
  - These sources to be register outputs or combinational logic of microoperations
  - To force R0 to load for a micro-operation, these control signals are ORed together to form **Load** signal



Summary of This Lecture

- Learned about register transfer operations
  - Microoperation
  - Register transfer language
  - Symbolic form of data transfer