

Digital Systems

Prepared by Dr. Alaa Eldeen Sayed Ahmed

IT131

Register Transfer language (RTL)

Lecture 10



Register Transfer Language (RTL)

Contents

1. Register Transfer Language
 1. Register Transfer
 2. Arithmetic Microoperations
 3. Logic Microoperations
 4. Shift Microoperations

Register Transfer Language (RTL)

Simple Digital Systems

- **Combinational circuits** perform logical and arithmetic operations where the output depends only on the current inputs.
- **Sequential circuits** store and control data flow using memory elements like flip-flops, making output depend on both current inputs and past states.

Register Transfer Language (RTL)

- When these two are combined, they create simple digital systems, which are often described in terms of:
 - The registers they contain – storage elements that hold data.
 - The microoperations they perform – processing, moving, and transforming the stored data.
- To fully describe digital systems, we need to answer:
 - What microoperations are performed on the data in the registers?
 - What information is passed between registers?

Register Transfer Language (RTL)

- **Register Transfer Language (RTL)** addresses exactly this need.

It provides a precise and structured way to describe:

- How data is moved between registers.
- What logical or arithmetic microoperations are performed on the data.
- When these microoperations occur, based on control signals and clock timing.

Register Transfer Language (RTL)

- By combining combinational logic (for data processing) and sequential logic (for data storage and timing), RTL gives us a complete picture of how a digital system behaves at the register level – making it a bridge between conceptual system design and actual hardware implementation.

Register Transfer Language (RTL)

Why we use RTL

1. Clear data movement description – It precisely shows:

1. Which registers are involved
2. How data is transferred
3. What microoperations are performed on the data

Register Transfer Language (RTL)

2. Design and verification – RTL descriptions are used in:

1. Digital system documentation
2. Simulation and verification before building hardware

3. Ease of synthesis – RTL can be directly translated into actual hardware circuits using synthesis tools (like in Verilog or VHDL).

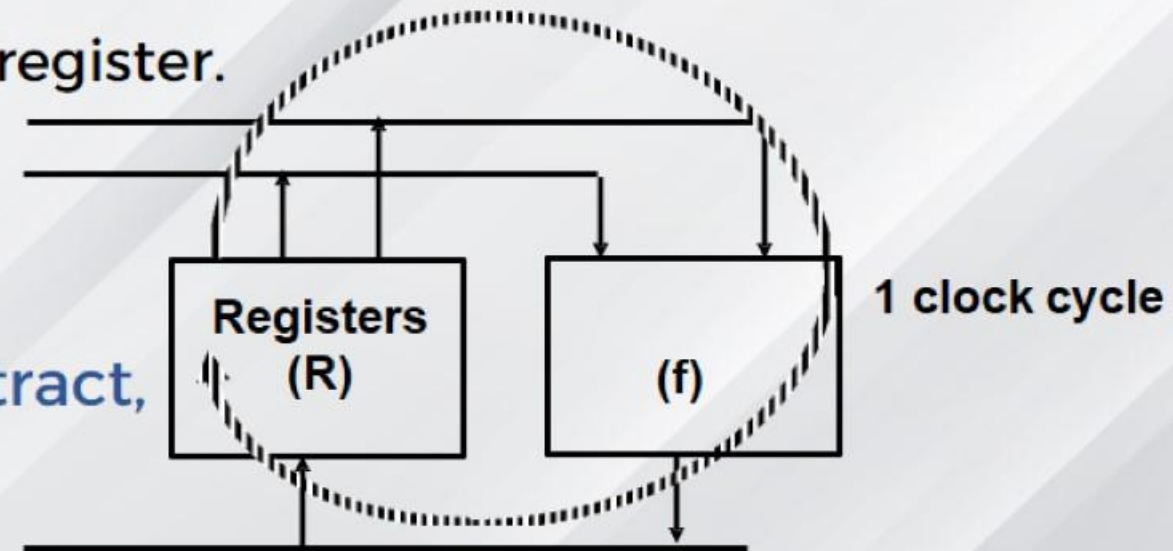
Register Transfer Language (RTL)

Micro-operations

- **Microoperation:** An elementary operation performed (during one clock pulse), on the information stored in one or more registers.
- The result of the operation may replace the previous binary information of a register or may be transferred to another register.

$$R \leftarrow f(R, R)$$

f : shift, load, clear, increment, add, subtract, complement, and, or, xor, ...



Register Transfer Language (RTL)

Organization of a Digital System

The (internal) hardware organization of a digital system (specially computer) is best defined by specifying:

- 1) Set of registers and their functions
 - 2) The sequence of microoperations preformed on binary information stored in the registers.
 - 3) Control signals that initiate the sequence of microoperations (to perform the functions)
- Every microoperation needs a special hardware implementation.

Register Transfer Language (RTL)

Designation of Registers

- Registers are designated by capital letters, sometimes followed by numbers (e.g., A, R13, IR)

R13

- Note:** FFs in a n-bit register are numbered in a sequence from 0 through n-1, starting from 0 in the rightmost position and increasing the numbers towards the left.

Register Transfer Language (RTL)

Designation of Registers

- Registers and their contents can be viewed and represented in *various ways (Block diagram)*:

1) Rectangular box with name inside



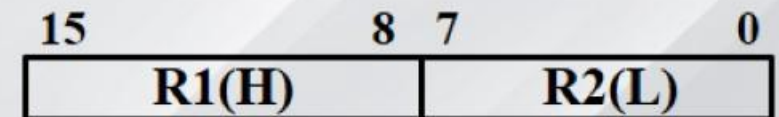
2) Showing individual bits



3) Numbering of bits on the tops of the register



4) Partitioned into two parts



Register Transfer Language (RTL)

Register Transfer

- **Register transfer:** Copying the contents of one register to another
- A register transfer is indicated as: $R2 \leftarrow R1$
- Contents of register R1 are copied (loaded) into register R2.
- A simultaneous transfer of all bits from the **source register R1** to the **destination register R2** is performed during **one clock pulse**
- Destination register has **parallel load capability**

Note: This is a non-destructive; i.e. the contents of R1 are not altered by copying (loading) them to R2

Register Transfer Language (RTL)

Register Transfer

- A register transfer such as

$R3 \leftarrow R5$

implies that the digital system has:

- The data lines from the source register (R5) to the destination register (R3)
- Parallel load in the destination register (R3)
- Control lines to perform the action

Register Transfer Language (RTL)

Control Functions

- Often actions need to only occur if a certain condition is true. This is similar to an “if” statement in a programming language.
- In digital systems, this is often done via a **control signal**, called a **control function**
 - The control function is a Boolean variable
 - If the signal is 1, the action takes place
 - The control function is included as following:

P: $R2 \leftarrow R1$

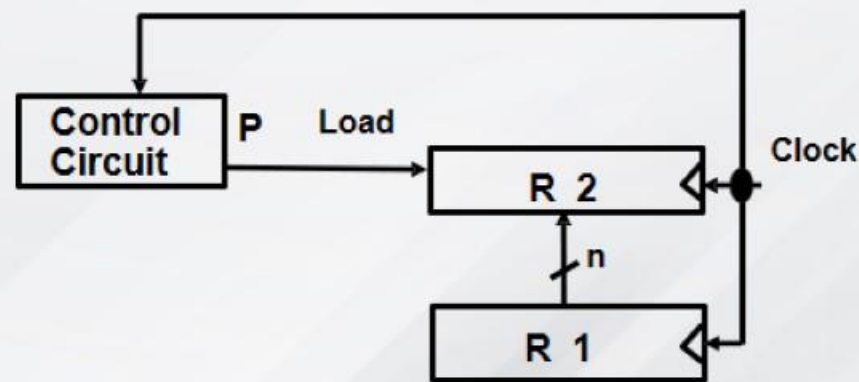
Which means “if $P = 1$, then load the contents of $R1$ into $R2$ ”, i.e., if $(P = 1)$ then $(R2 \leftarrow R1)$

Register Transfer Language (RTL)

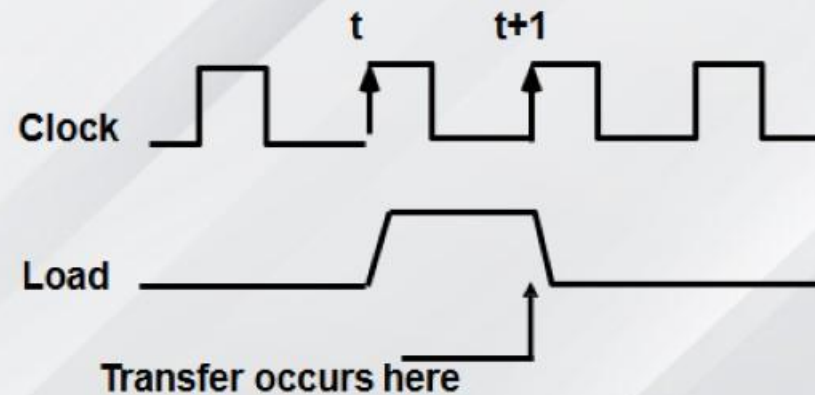
Hardware Implementation of Controlled Transfers

Implementation of controlled transfer **P: R2 $\leftarrow$ R1**

- The same clock controls the circuits that generate the control function and the destination register
- Registers are assumed to use *positive-edge-triggered* flip-flops



Block diagram



Timing diagram

Register Transfer Language (RTL)

Simultaneous Operations

- If two or more operations are to occur simultaneously, they are separated with commas

P: $R3 \leftarrow R5, R1 \leftarrow R6$

- Here, if the control function $P = 1$, load the contents of $R5$ into $R3$, and at the same time (clock), load the contents of register $R6$ into register $R1$

Register Transfer Language (RTL)

Example

Represent the following conditional control statement by two register transfer statements with control functions.

If ($P = 1$) then ($R1 \leftarrow R2$) else if ($Q = 1$) then ($R1 \leftarrow R3$)

Answer:

P: $R1 \leftarrow R2$

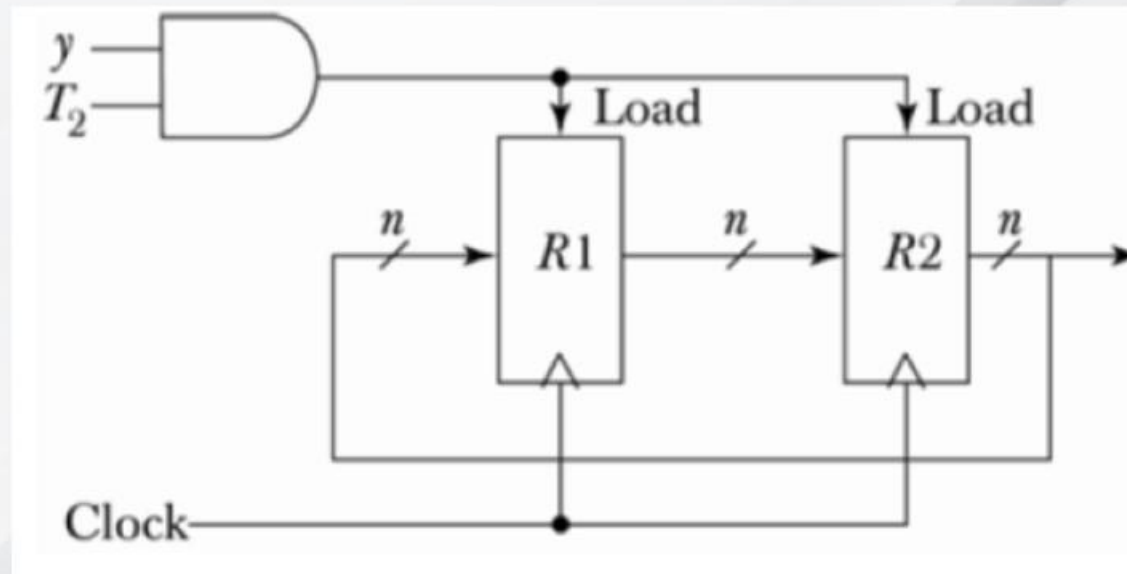
P'Q: $R1 \leftarrow R3$

Register Transfer Language (RTL)

Example

Show the block diagram of the hardware that implements the following register transfer statement:

$$yT_2 : R2 \leftarrow R1, R1 \leftarrow R2$$



Register Transfer Language (RTL)

Microoperations

Computer system microoperations are of four types:

- 1) Register transfer microoperations
- 2) Arithmetic microoperations
- 3) Logic microoperations
- 4) Shift microoperations

Register Transfer Language (RTL)

1) Register Transfer Microoperations

Symbols	Description	Examples
Capital letters & numerals	Denotes a register	R1, R2
Parentheses ()	Denotes a part of a register	R2(0-7), R2(L)
Arrow $\leftarrow$	Denotes transfer of information	$R2 \leftarrow R1$
Colon :	Denotes termination of control function	P:
Comma ,	Separates two simultaneous microoperations	$A \leftarrow B, B \leftarrow A$

Register Transfer Language (RTL)

2) Arithmetic Microoperations

- The basic arithmetic microoperations are
 - Addition
 - Subtraction
 - Increment
 - Decrement
- The additional arithmetic microoperations are
 - Add with carry
 - Subtract with borrow
 - Transfer/Load
 - etc. ...

Register Transfer Language (RTL)

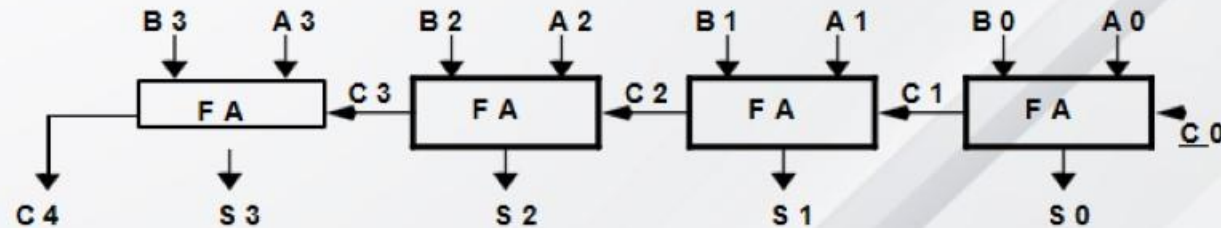
Summary of Typical Arithmetic Microoperations

$R3 \leftarrow R1 + R2$	Contents of R1 plus R2 transferred to R3
$R3 \leftarrow R1 - R2$	Contents of R1 minus R2 transferred to R3
$R2 \leftarrow R2'$	Complement the contents of R2
$R2 \leftarrow R2' + 1$	2's complement the contents of R2 (negate)
$R3 \leftarrow R1 + R2' + 1$	Subtraction
$R1 \leftarrow R1 + 1$	Increment
$R1 \leftarrow R1 - 1$	Decrement

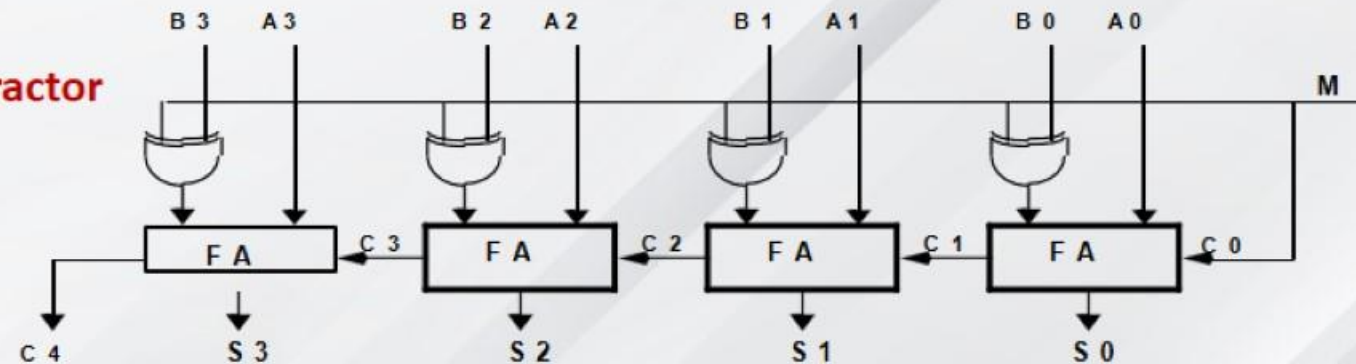
Register Transfer Language (RTL)

Binary Adder / Subtractor

Binary Adder

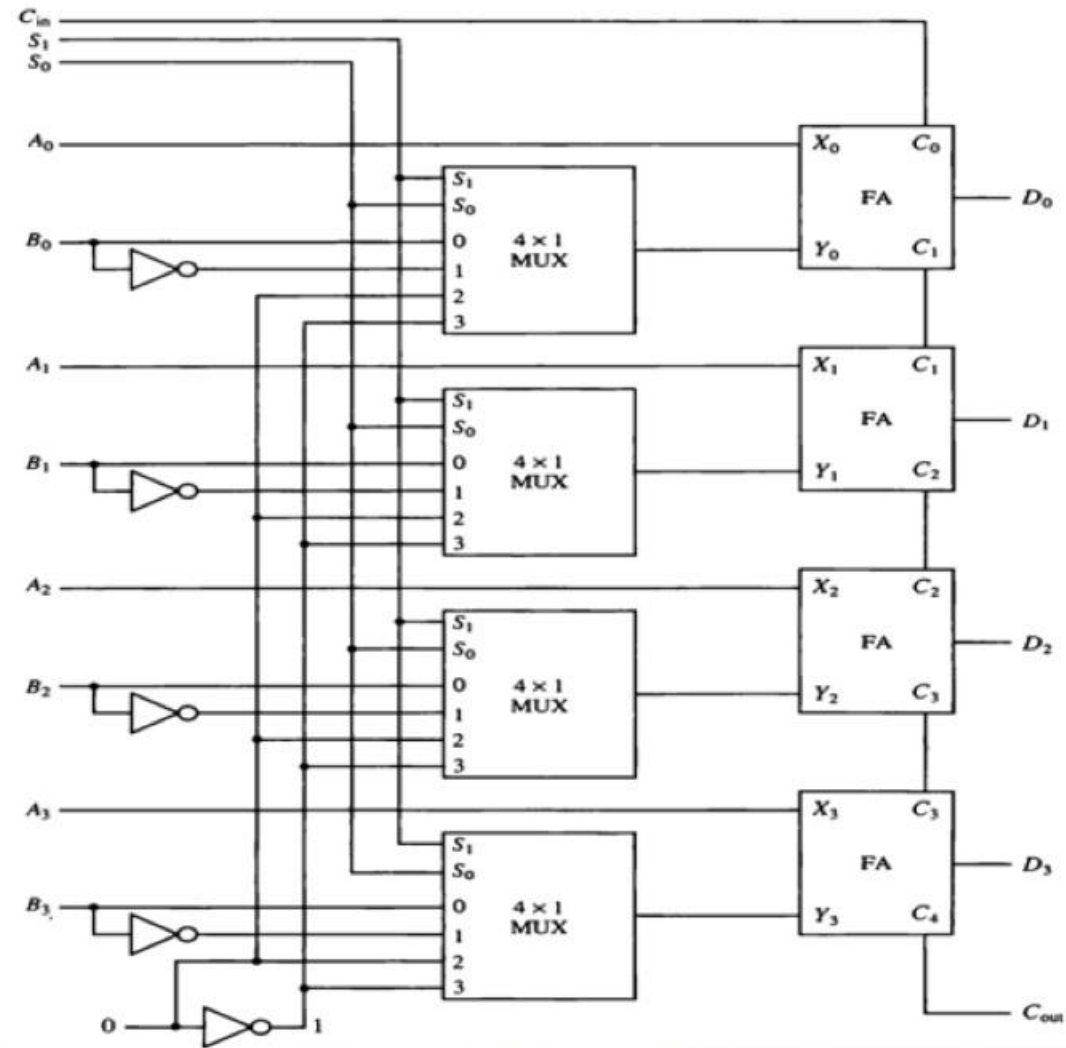


Binary Adder - Subtractor



Register Transfer Language (RTL)

Arithmetic Circuit: Implementation



A, B and D are 4 bit width

S_1	S_0	C_{in}	Y	Output	Microoperation
0	0	0	B	$D = A + B$	Add
0	0	1	B	$D = A + B + 1$	Add with carry
0	1	0	B'	$D = A + B'$	Subtract with 1's complement
0	1	1	B'	$D = A + B' + 1$	Subtract with 2's complement
1	0	0	0	$D = A$	Transfer <u>A</u>
1	0	1	0	$D = A + 1$	Increment A
1	1	0	1	$D = A - 1$	Decrement A
1	1	1	1	$D = A$	Transfer A

Register Transfer Language (RTL)

Arithmetic Circuit: Function Table

- The output can be calculated as: $D = A + Y + C_{in}$
- By controlling the input Y by the MUX (S_0, S_1)
- With C_{in} equal to 0 or 1
- All 8 operations can be done

S_1	S_0	C_{in}	Y	Output	Microoperation
0	0	0	B	$D = A + B$	Add
0	0	1	B	$D = A + B + 1$	Add with carry
0	1	0	B'	$D = A + B'$	Subtract with 1's comp
0	1	1	B'	$D = A + B' + 1$	Subtract with 2's comp
1	0	0	0	$D = A$	Transfer <u>A</u>
1	0	1	0	$D = A + 1$	Increment A
1	1	0	1	$D = A - 1$	Decrement A
1	1	1	1	$D = A$	Transfer A

Example

- $x + yz$: $AR \leftarrow AR + BR$**

-

Register Transfer Language (RTL)

3) Logic Microoperations

- Specify binary operations on the strings of bits in registers
 - Logic microoperations are bit-wise operations, i.e., they work on the individual bits of data
 - useful for bit manipulations on binary data
 - useful for making logical decisions based on the bit value

Register Transfer Language (RTL)

x	y	F_0	F_1	F_2	F_3	F_4	F_5	F_6	F_7	F_8	F_9	F_{10}	F_{11}	F_{12}	F_{13}	F_{14}	F_{15}
0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
0	1	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
1	0	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
1	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1

- There are, in principle, 16 different logic functions that can be defined over two binary input variables

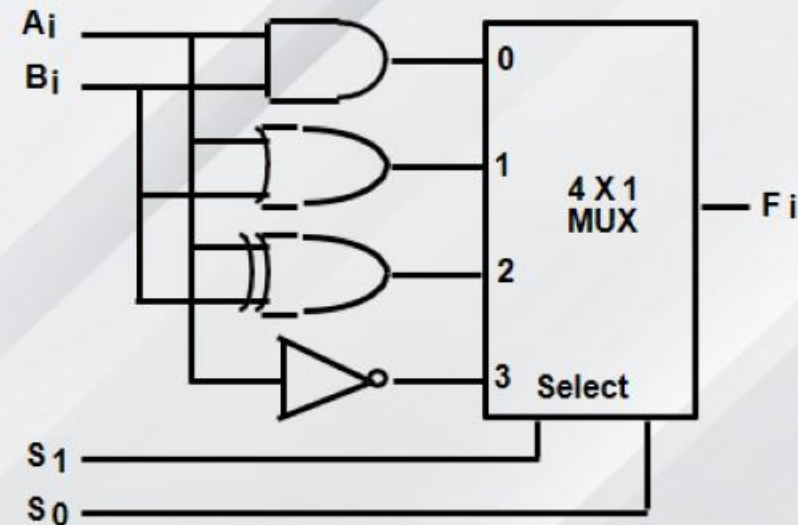
Register Transfer Language (RTL)

Hardware Implementation of Logic Microoperations

Figure below shows one stage of a circuit that generates the four basic logic microoperations . It consists of **four gates and a multiplexer**.

Function table

S_1	S_0	Output	μ -operation
0	0	$F = A \wedge B$	AND
0	1	$F = A \vee B$	OR
1	0	$F = A \oplus B$	XOR
1	1	$F = A'$	Complement



Register Transfer Language (RTL)

4) Shift Microoperations

- There are three types of shifts:
 - **Logical** shift
 - **Circular** shift
 - **Arithmetic** shift

What differentiates them is the information that goes into the serial input

A right shift operation



A left shift operation



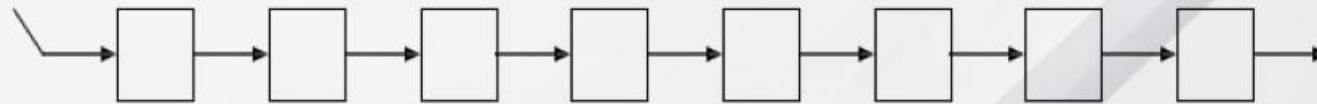
Register Transfer Language (RTL)

Logical Shift

In a logical shift the serial input to the shift is a **0**

A right logical shift operation:

0



A left logical shift operation:

0



Register Transfer Language (RTL)

Example:

- In a Register Transfer Language, the following notation is used
 - ***shl*** for a logical shift left
 - ***shr*** for a logical shift right

Examples: $R2 \leftarrow shr R2$ (100110 after *shr* it will be: 010011~~0~~) = 010011

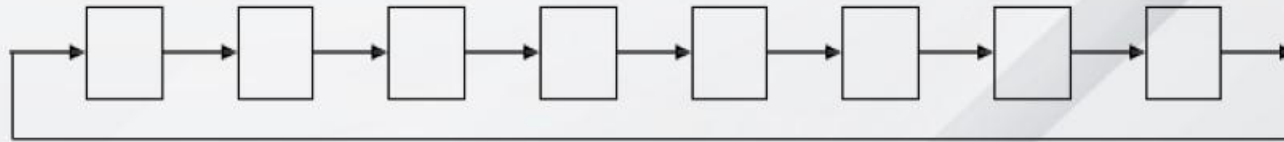
Examples: $R3 \leftarrow shl R3$ (100110 after *shl* it will be: ~~1~~001100) = 001100

Register Transfer Language (RTL)

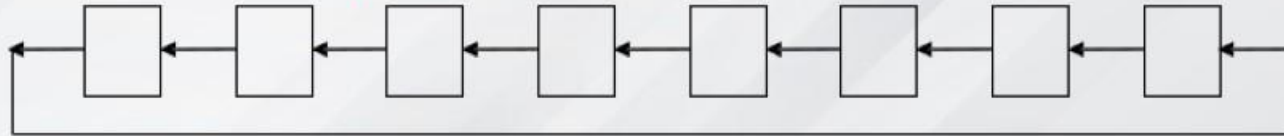
Circular Shift

In a circular shift the serial input is the bit that is shifted out of the other end of the register.

A right circular shift operation:



A left circular shift operation:



Register Transfer Language (RTL)

Example:

- In a RTL, the following notation is used
 - ci*** for a circular shift left
 - ir*** for a circular shift right

Examples: $R2 \leftarrow \text{cir } R2$ (100110 after *cir* it will be: **0**10011~~0~~) = 010011

Examples: $R3 \leftarrow \text{ci/} R3$ (100110 after *ci/* it will be: ~~1~~00110**1**) = 001101

Register Transfer Language (RTL)

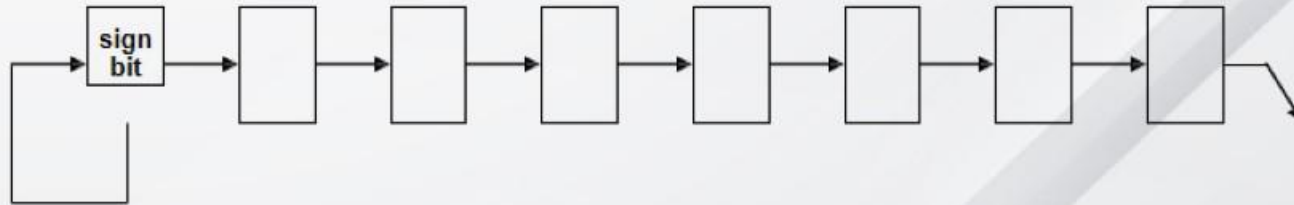
Arithmetic Shift

- An arithmetic shift is meant for **signed binary numbers (integer)**
 - An arithmetic left shift multiplies a signed number by two
 - An arithmetic right shift divides a signed number by two
- The main distinction of an arithmetic shift is that it must keep the sign of the number the same as it performs the multiplication or division
- In a RTL, the following notation is used
 - ***ashl*** for an arithmetic shift left
 - ***ashr*** for an arithmetic shift right

Register Transfer Language (RTL)

Arithmetic Shift

A right arithmetic shift operation:



A left arithmetic shift operation:



Register Transfer Language (RTL)

Examples:

$R2 \leftarrow \text{ashr } R2$ (100110 after *ashr* it will be 1100110 = 110011)

$R3 \leftarrow \text{ashl } R3$ (000110 after *ashl* it will be: 0001100 = 001100)

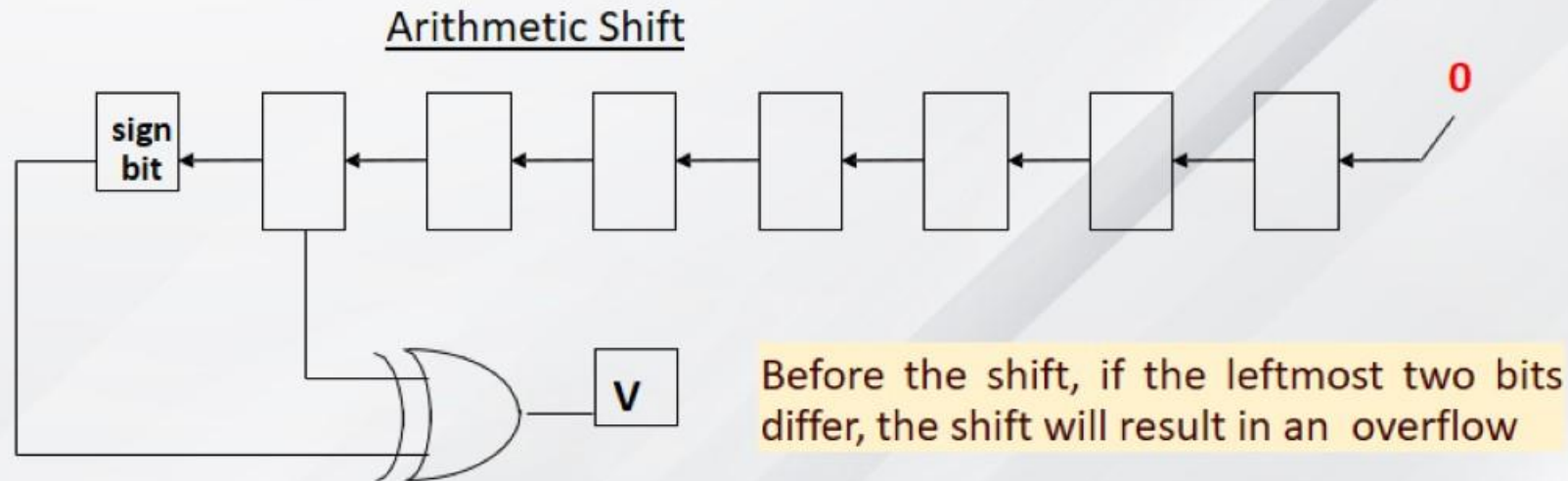
$R2 \leftarrow \text{ashl } R2$ (100110 after *ashl* it will be: 1001100 = 001100)

Change of sign $\rightarrow$ overflow!

Register Transfer Language (RTL)

Arithmetic Shift

A left arithmetic shift operation must be checked for the overflow



Recent Trends in Digital System Design

The next slides introduces how digital system design has evolved in recent years.

It highlights the major forces driving modern design choices, mainly:

- Higher performance demands
- Lower power consumption requirements
- Need for scalability across many devices
- Growth of AI, edge computing, and specialized hardware

Recent Trends in Digital System Design

AI-Driven Digital Design Automation

Design automation tools now integrate artificial intelligence and machine learning to speed up hardware design, optimization, verification, and layout generation.

- **Key Points:**

- Predictive design space exploration
- Automated error detection & correction
- ML models assisting synthesis and placement

References:

- 1) H. Bagherzadeh and A. Leon-Garcia, "Machine Learning in Electronic Design Automation: A Survey," IEEE Trans. CAD, vol. 42, no. 3, pp. 453–468, 2024.
- 2) X. Chen et al., "AI-Assisted Hardware Design Automation for Modern VLSI," IEEE Des. Test, vol. 41, no. 1, pp. 22–32, 2025.

Recent Trends in Digital System Design

High-Level Synthesis & Heterogeneous Computing

- High-Level Synthesis (HLS) tools enable system designers to generate hardware from high-level languages (e.g., C/C++, Python, VHDL Verilog) and map functions to heterogeneous platforms (CPU/FPGA/ASIC).
- **Key Points:**
 - Abstract hardware description languages
 - Automatic partitioning across heterogeneous units
 - Better productivity and reuse

References:

- 1) T. Sato and Y. Matsumoto, "Advances in High-Level Synthesis for Heterogeneous Platforms," IEEE Comput. Archit. Lett., vol. 23, pp. 55-58, 2024.
- 2) A. Gupta et al., "Design Challenges in Heterogeneous Embedded Systems," IEEE Embedded Syst. Lett., vol. 15, no. 4, pp. 101-105, 2025.

Recent Trends in Digital System Design

Hardware Security & Trustworthy Systems

- Security is increasingly built into the digital design process, including hardware root-of-trust, secure boot, side-channel resistance, and anti-tamper logic at design time.
- Key Points:
 - Secure hardware primitives
 - Side-channel attack mitigation
 - Formal verification of security properties
- References:
 - 1) J. Lee and M. Tehranipoor, "Hardware Security: Fundamentals and Design," IEEE Trans. Inf. Forensics Security, vol. 19, pp. 117-134, 2024.
 - 2) S. Kim et al., "Formal Hardware Security Verification Tools: A Review," IEEE Access, vol. 12, pp. 45 678-45 692, 2024.

Recent Trends in Digital System Design

Low-Power and Energy-Efficient Design

- With IoT, edge computing, and mobile systems proliferating, power consumption is vital. Designs target adaptive voltage scaling, near-threshold computing, and power-aware architectures.
- **Key Points:**
 - Power gating & DVFS at RTL/architectural levels
 - Energy-aware synthesis
 - Battery/thermal-aware design

References:

- 1) L. Wang et al., "Low-Power Digital Design Techniques for IoT Devices," IEEE Trans. VLSI Syst., vol. 32, no. 2, 2024.
- 2) R. Singh and K. Roy, "Energy-Efficient Architectures for Edge AI," IEEE Circuits Syst. Mag., vol. 25, no. 1, pp. 80-95, 2025.



Thank You