



Computer Architecture

2024 – 2025 Academic Year

CEIT – 51023 Computer Architecture I

(Fifth Year – First Semester)

Dr. Theingi Myint

Professor

Department of Computer Engineering and Information Technology

Mandalay Technological University

Contents

Chapter (1): Digital Logic Circuits

Chapter (2): Digital Components

Chapter (4): Register Transfer and Microoperations

Chapter (6): Programming the Basic Computer

Chapter (8): Central Processing Unit

Chapter Four: Register Transfer and Microoperations

- **Introduces a register transfer language and shows how it is used to express microoperations in symbolic form**
- **Defines for arithmetic, logic, and shift microoperations**
- **Show the hardware design of the most common microoperations**

Register Transfer Language

- The operations executed on data stored in registers are called **microoperations**.
- A **microoperation** is an elementary operation performed 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.
- Examples of microoperations are:



Register Transfer Language

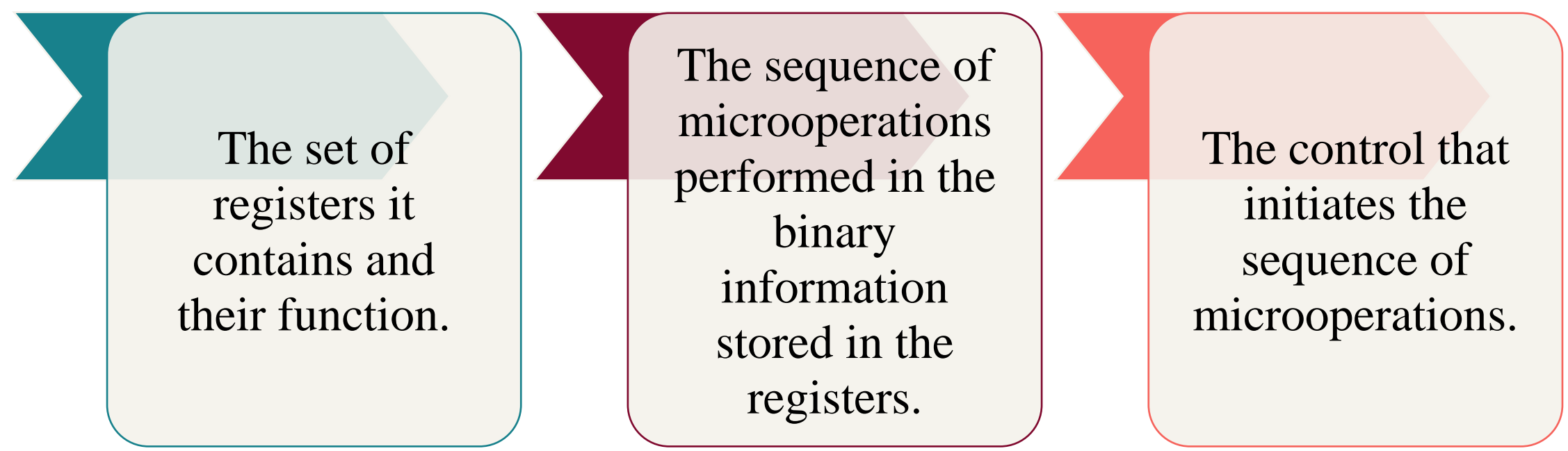
- Some of the digital components are registers that implement microoperations.

A **counter** with parallel load is capable of performing the microoperations **increment** and **load**.

A **bidirectional shift register** is capable of performing the **shift right** and **shift left** microoperations.

Register Transfer Language

- The internal hardware organization of a digital computer is best defined by specifying:



The set of registers it contains and their function.

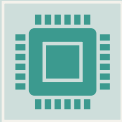
The sequence of microoperations performed in the binary information stored in the registers.

The control that initiates the sequence of microoperations.

Register Transfer Language



The symbolic notation used to describe the microoperation transfers among registers is called a **register transfer language** (RTL).



Register transfer implies the availability of hardware logic circuits that can perform a stated microoperation and transfer the results of the operation to the same or another register.

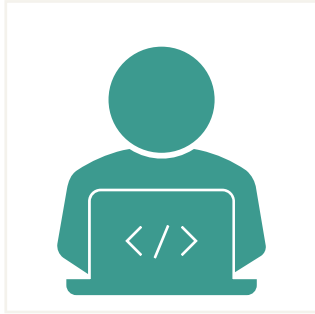


A **register transfer language** is a system for expressing in symbolic form the microoperation sequences among the registers of digital computers in concise and precise manner.



It can also be used to facilitate the design process of digital systems.

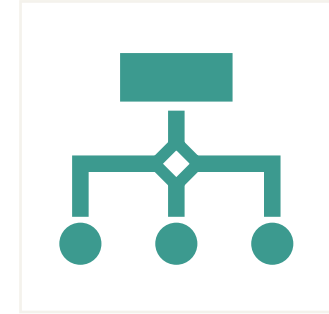
Register Transfer Language



The **register transfer language** (RTL) is a powerful high-level method of describing the architecture of a circuit.



VHDL code or Verilog code and schematics are often created from RTL.



RTL describes the transfer of data from register to register, known as **microinstructions or microoperations**.

Register Transfer

- Computer registers are designated by **capital letters** (sometimes followed by numerals) to denote the function of the register.
- **Example:** the register that holds an address for the memory unit is called **a memory address register** and is designated by the name **MAR**.
- Other designations for registers are **PC** (for program counter), **IR** (for instruction register), and **R1** (for processor register).

Register Transfer

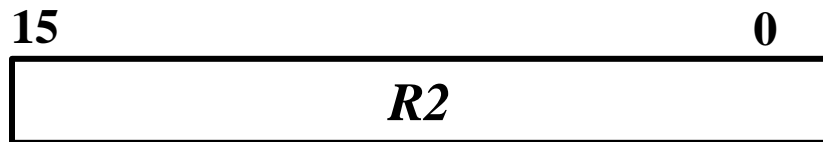
- The individual flip-flops in an n -bit register are numbered in sequence from 0 through $n-1$, starting from 0 in the rightmost position and increasing the numbers toward the left.



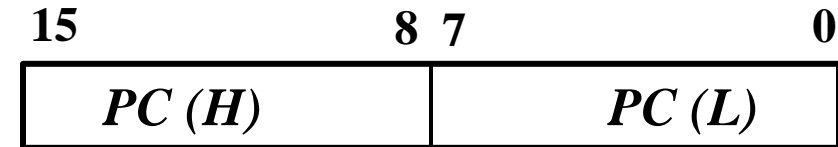
(a) Register R



(b) Showing individual bits



(c) Numbering of bits



(d) Divided into two parts

Block diagram of register

Register Transfer

Information transfer from one register to another is designated in **symbolic form** by means of a replacement operator.

The statement **$R2 \leftarrow R1$** denotes **a transfer of the content of register R1 into register R2.**

It designates a replacement of the content of R2 by the content of R1.

The content of the source register R1 does not change after the transfer.

A statement that specifies a register transfer implies that circuits are available from **the outputs of the source register** to **the inputs of the destination register** and that the destination register has a parallel load capability.

Register Transfer

- The transfer to occur only under a predetermined control condition can be shown by means of an *if-then* statement.

If ($P = 1$) then ($R2 \leftarrow R1$)

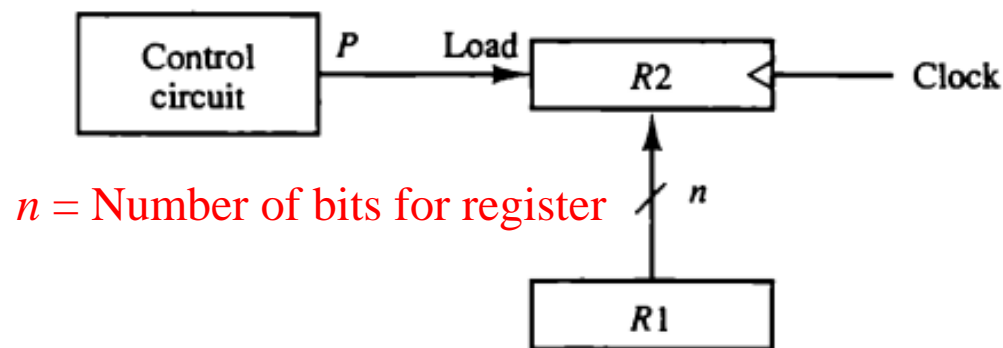
- Where P is a control signal generated.
- It is convenient to separate the control variables from the register transfer operation by specifying a **control function**.
- A **control function** is a Boolean variable that is equal to 1 or 0.

$P: R2 \leftarrow R1$

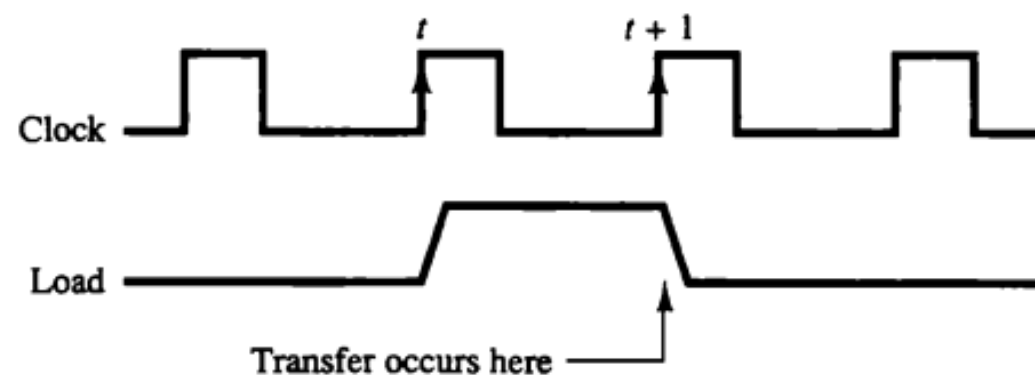
- The control function is terminated with a **colon**.
- The transfer operation can be executed by the hardware only if $P = 1$.

Register Transfer

- Transfer from R1 to R2 when $P = 1$.



(a) Block diagram



(b) Timing diagram

- A **comma** is used to **separate two or more operations** that are executed at the same time.

$$T: R2 \leftarrow R1, R1 \leftarrow R2$$

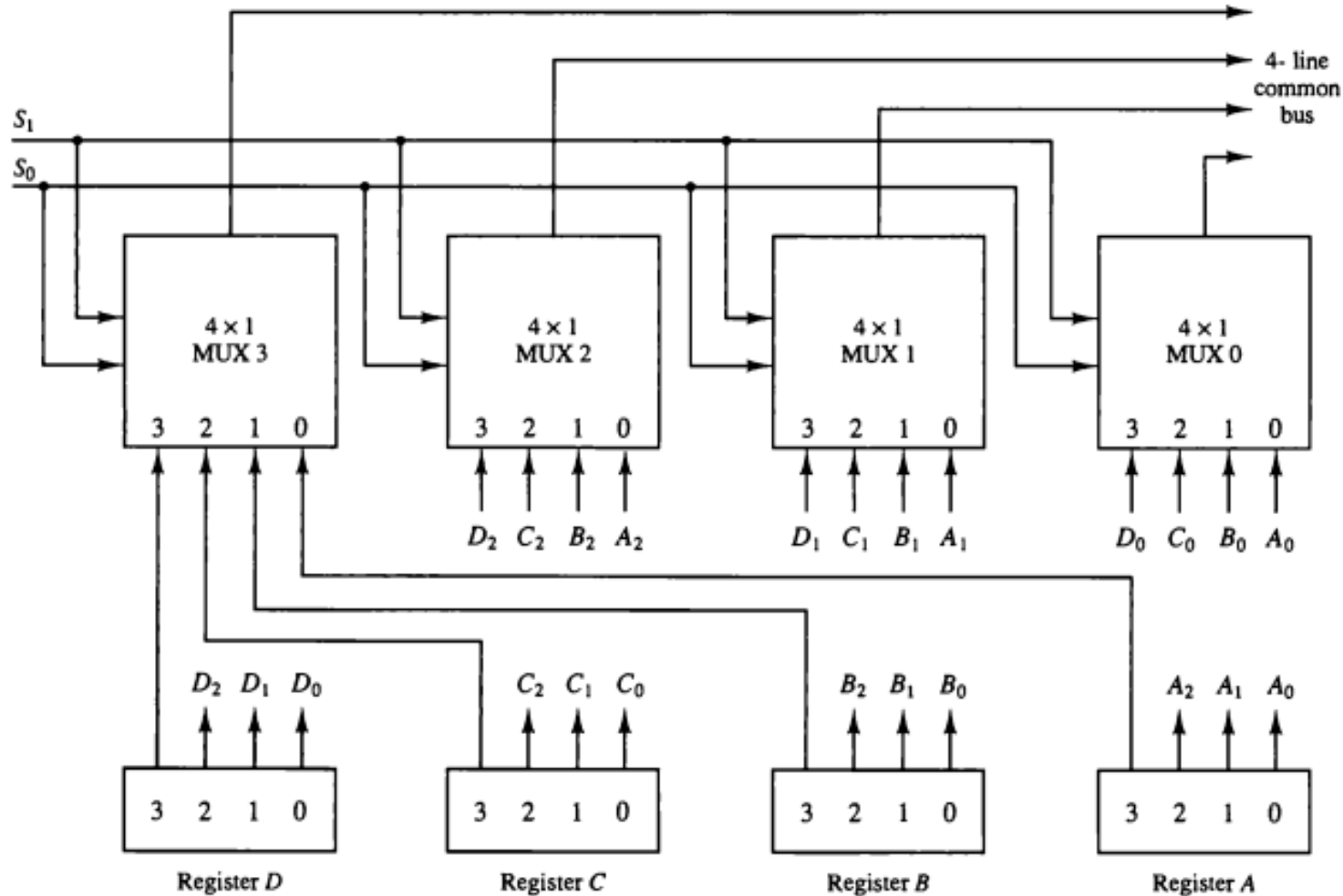
- It denotes an operation that exchanges the contents of two registers during one common clock pulse provided that $T = 1$.

Symbol	Description	Examples
Letters (and numerals)	Denotes a register	MAR, R2
Parentheses ()	Denotes a part of a register	R2(0-7), R2(L)
Arrow \leftarrow	Denotes transfer of information	$R2 \leftarrow R1$
Comma ,	Separates two microoperations	$R2 \leftarrow R1, R1 \leftarrow R2$

Bus and Memory Transfers

- A typical digital computer has **many registers**, and **paths** must be provided to transfer information **from one register to another**.
- The **number of wires** will be excessive if separate lines are used between each register and all other registers in the system.
- A more efficient scheme for transferring information between registers in a multiple register configuration is **a common bus system**.
- A **bus** structure consists of **a set of common lines**, one for each bit of a register, through which binary information is transferred one at a time.
- Control signals determine which register is selected by the bus during each particular register transfer.

Bus and Memory Transfers



- The bus contains **4×1 MUX** and each has **four data inputs, 0 through 3**, and **two selection inputs S_1 and S_0** .
- The two selection lines **S_1** and **S_0** are connected to the selection inputs of all four multiplexers.
- The **selection lines** choose **the four bits of one register** and transfer them into the four-line common bus.

Bus system for four registers

Bus and Memory Transfers

When $S_1S_0 = 00$, the 0 data inputs of all MUX are selected and applied to the output that form the bus.

This causes the bus lines to receive the content of register A since the output of this register are connected to the 0 data inputs of the MUX.

Register B is selected if $S_1S_0 = 01$, and register C is selected if $S_1S_0 = 10$, and register D is selected if $S_1S_0 = 11$.

S_1	S_0	Register selected
0	0	A
0	1	B
1	0	C
1	1	D

Bus and Memory Transfers

- In general, a bus system will multiplex k registers of n bits each to produce an n line common bus.
- The number of multiplexers needed to construct the bus is equal to n , the number of bits in each register.
- The size of each multiplexer must be $k \times 1$ since it multiplexes k data lines.
- For example, a common bus for eight registers of 16 bits each requires 16 multiplexers, one for each line in the bus.
- Each multiplexer must have **eight data input lines** and **three selection lines** to multiplex one significant bit in the eight registers.
- The transfer of information from a bus into one of many destination registers can be accomplished **by connecting the bus lines to the inputs of all destination registers** and activating the load control of the particular destination register selected.

Bus and Memory Transfers

- The transfer of information from a memory word to the outside environment is called **a read operation**.
- The transfer of new information to be stored into the memory is called **a write operation**.
- A memory word will be symbolized by the letter ***M***.
- The particular memory word among the many available is selected by the memory address during the transfer.
- Consider a memory unit that receives the address from a register, called the **address register**, symbolized by ***AR***.
- The data are transferred to another register, called the **data register**, symbolized by ***DR***.

Bus and Memory Transfers

- The **read operation** can be stated as follows:

Read: $DR \leftarrow M[AR]$

- This causes a transfer of information into **DR** from the memory word **M** selected by the address in **AR**.
- The **write operation** transfer the content of a data register to a memory word **M** selected by the address.
- Assume that the input data are in register **R1** and the address is in **AR**.
- The **write operation** can be started symbolically as follows :

Write : $M[AR] \leftarrow R1$

- This cause a transfer of information from **R1** into the memory word **M** selected by the address in **AR**.

Microoperations

- The operations on the data in registers are called **microoperations**.
- Alternatively, we can say that an elementary operation performed during one clock pulse on the information stored in one or more registers is called microoperation.
- The result of the operation may replace the previous binary information of the register or may be transferred to another register.
- Register Transfer Language (RTL) can be used to describe the (sequence of) microoperations.
- The microoperations most often encountered in digital computers are classified into 4 categories:



Microoperations → Register Transfer Microoperations

- Registers are designated by **capital** letters, sometimes followed by **numbers** (e.g., A, R13, IR).
- Often the names indicate function:

MAR	memory address register
PC	program counter
IR	instruction register
- Information transfer from one register to another is described in symbolic form by replacement operator.
- The statement “ $R2 \leftarrow R1$ ” denotes a transfer of the content of the R1 into register R2.

Microoperations → Register Transfer Microoperations

- Often actions need to only occur if a certain condition is true.
- In digital systems, this is often done via a **control signal**, called a **control function**.

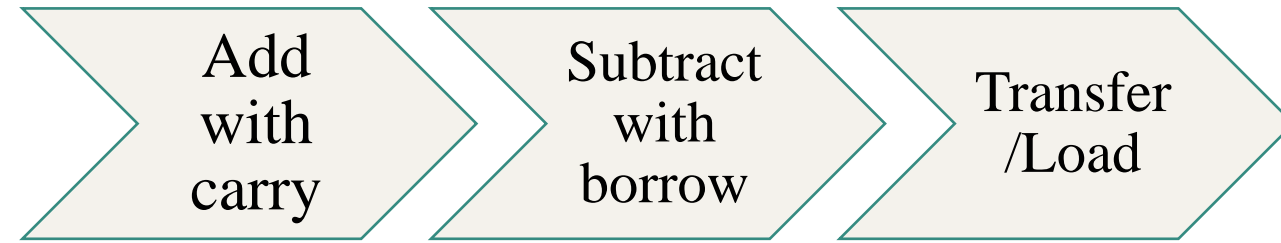
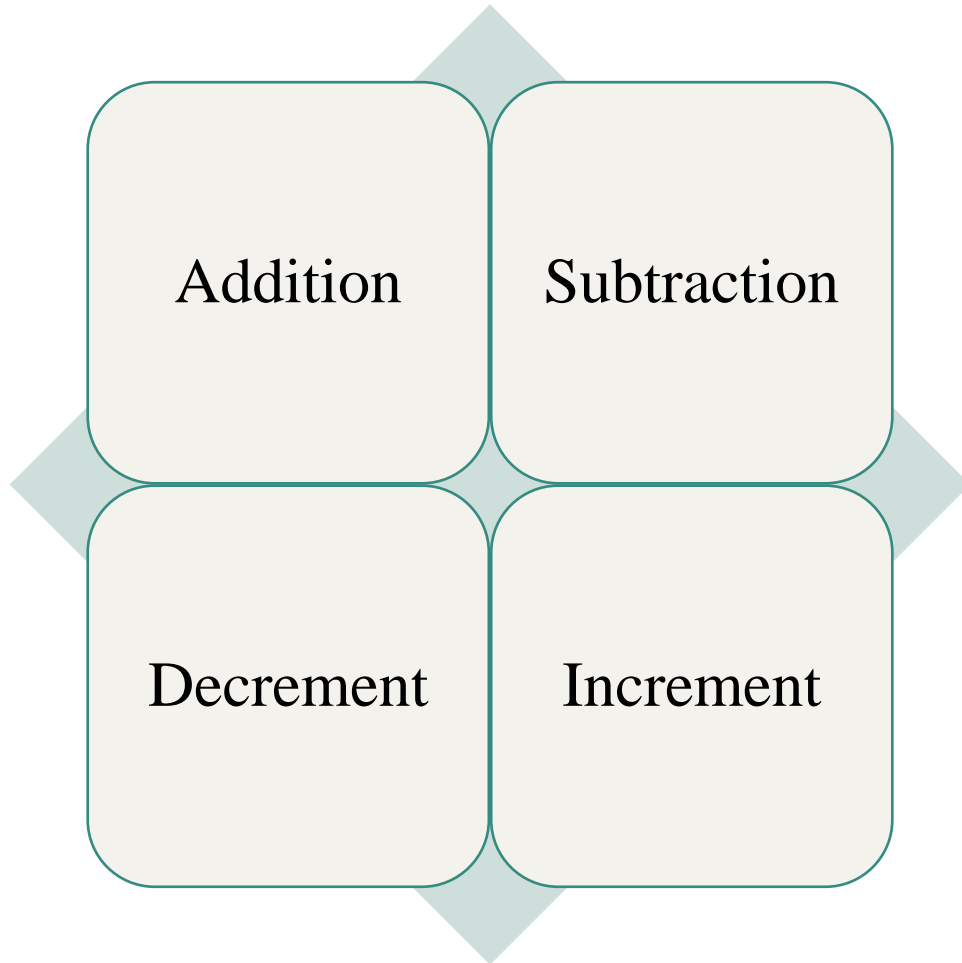
Example: $P: R2 \leftarrow R1$ i.e. if ($P = 1$) then ($R2 \leftarrow R1$)

- Which means “if $P = 1$, then load the contents of register $R1$ into register $R2$ ”.
- If two or more operations are to occur simultaneously, they are separated with commas.

Example: $P: R3 \leftarrow R5, MAR \leftarrow IR$

Microoperations → Arithmetic Microoperations

- The basic arithmetic microoperations are
- The additional arithmetic microoperations are



Microoperations → Arithmetic Microoperations

- The typical arithmetic microoperations are

Symbolic designation	Description
$R3 \leftarrow R1 + R2$	Contents of $R1$ plus $R2$ transferred to $R3$
$R3 \leftarrow R1 - R2$	Contents of $R1$ minus $R2$ transferred to $R3$
$R2 \leftarrow \overline{R2}$	Complement the contents of $R2$ (1's complement)
$R2 \leftarrow \overline{R2} + 1$	2's complement the contents of $R2$ (negate)
$R3 \leftarrow R1 + \overline{R2} + 1$	$R1$ plus the 2's complement of $R2$ (subtraction)
$R1 \leftarrow R1 + 1$	Increment the contents of $R1$ by one
$R1 \leftarrow R1 - 1$	Decrement the contents of $R1$ by one

Microoperations → Logic Microoperations

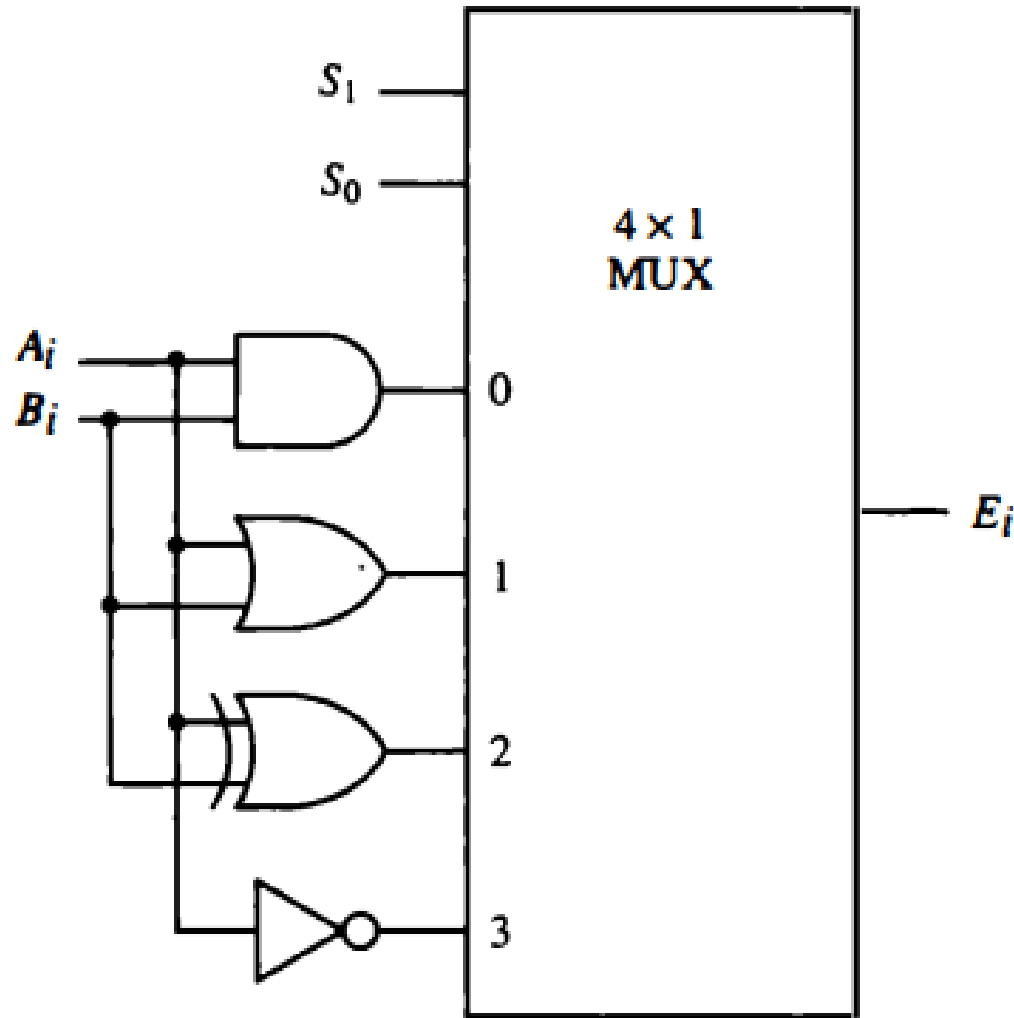
- Logic microoperations specify **binary operations** for strings of bits stored in registers.
- Logic microoperations are **bit-wise operations**, i.e., they work on the individual bits of data.
- Useful for bit manipulations on binary data and for making logical decisions based on the bit value.
- There are **16 different logic functions** that can be defined over two binary input variables.

Microoperations → Logic Microoperations

Boolean function	Microoperation	Name
$F_0 = 0$	$F \leftarrow 0$	Clear
$F_1 = xy$	$F \leftarrow A \wedge B$	AND
$F_2 = xy'$	$F \leftarrow A \wedge \bar{B}$	
$F_3 = x$	$F \leftarrow A$	Transfer A
$F_4 = x'y$	$F \leftarrow \bar{A} \wedge B$	
$F_5 = y$	$F \leftarrow B$	Transfer B
$F_6 = x \oplus y$	$F \leftarrow A \oplus B$	Exclusive-OR
$F_7 = x + y$	$F \leftarrow A \vee B$	OR
$F_8 = (x + y)'$	$F \leftarrow \overline{A \vee B}$	NOR
$F_9 = (x \oplus y)'$	$F \leftarrow \overline{A \oplus B}$	Exclusive-NOR
$F_{10} = y'$	$F \leftarrow \bar{B}$	Complement B
$F_{11} = x + y'$	$F \leftarrow A \vee \bar{B}$	
$F_{12} = x'$	$F \leftarrow \bar{A}$	Complement A
$F_{13} = x' + y$	$F \leftarrow \bar{A} \vee B$	
$F_{14} = (xy)'$	$F \leftarrow \overline{A \wedge B}$	NAND
$F_{15} = 1$	$F \leftarrow \text{all 1's}$	Set to all 1's

- The hardware implementation of logic microoperations requires that logic gates be inserted for each bit or pair of bits in the registers to perform the required logic function.
- Although there are 16 logic microoperations, most computers use only four: **AND**, **OR**, **XOR (exclusive-OR)**, and **complement** from which all others can be derived.

Microoperations → Logic Microoperations



(a) Logic diagram

Hardware Implementation

S_1	S_0	Output	Operation
0	0	$E = A \wedge B$	AND
0	1	$E = A \vee B$	OR
1	0	$E = A \oplus B$	XOR
1	1	$E = \bar{A}$	Complement

(b) Function table

Microoperations → Shift Microoperations



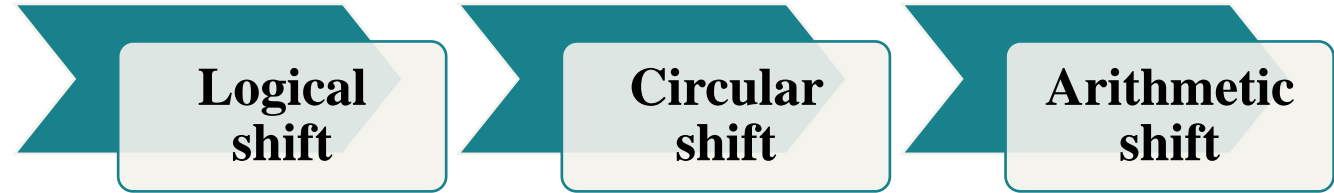
The operation that changes the adjacent bit position of the binary values stored in the register is known as **shift microoperation**.



They are used for **serial transfer of data**.



The shift microoperations are classified into **three types**:



Symbolic designation	Description
$R \leftarrow \text{shl } R$	Shift-left register R
$R \leftarrow \text{shr } R$	Shift-right register R
$R \leftarrow \text{cil } R$	Circular shift-left register R
$R \leftarrow \text{cir } R$	Circular shift-right register R
$R \leftarrow \text{ashl } R$	Arithmetic shift-left R
$R \leftarrow \text{ashr } R$	Arithmetic shift-right R

Microoperations → Shift Microoperations

- **Logical shift:** A logical shift transfer 0 through the serial input.
- It can be defined in RTL by:

$R \leftarrow \text{shl } R$ shift-left register R

$R \leftarrow \text{shr } R$ shift-right register R



Microoperations → Shift Microoperations

- **Circular shift:** A circular shift rotates the bit from one end of the register to another end of the register.
- It can be defined in RTL by:

$R \leftarrow cil\ R$ circular shift-left register R

$R \leftarrow cir\ R$ circular shift-right register R

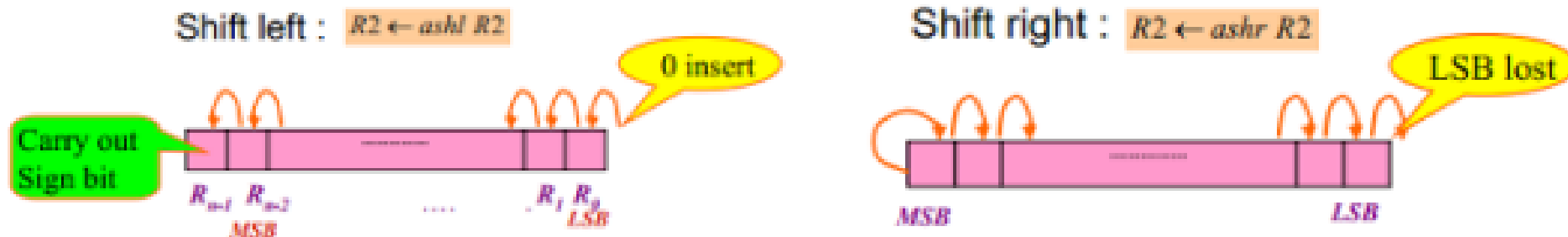


Microoperations → Shift Microoperations

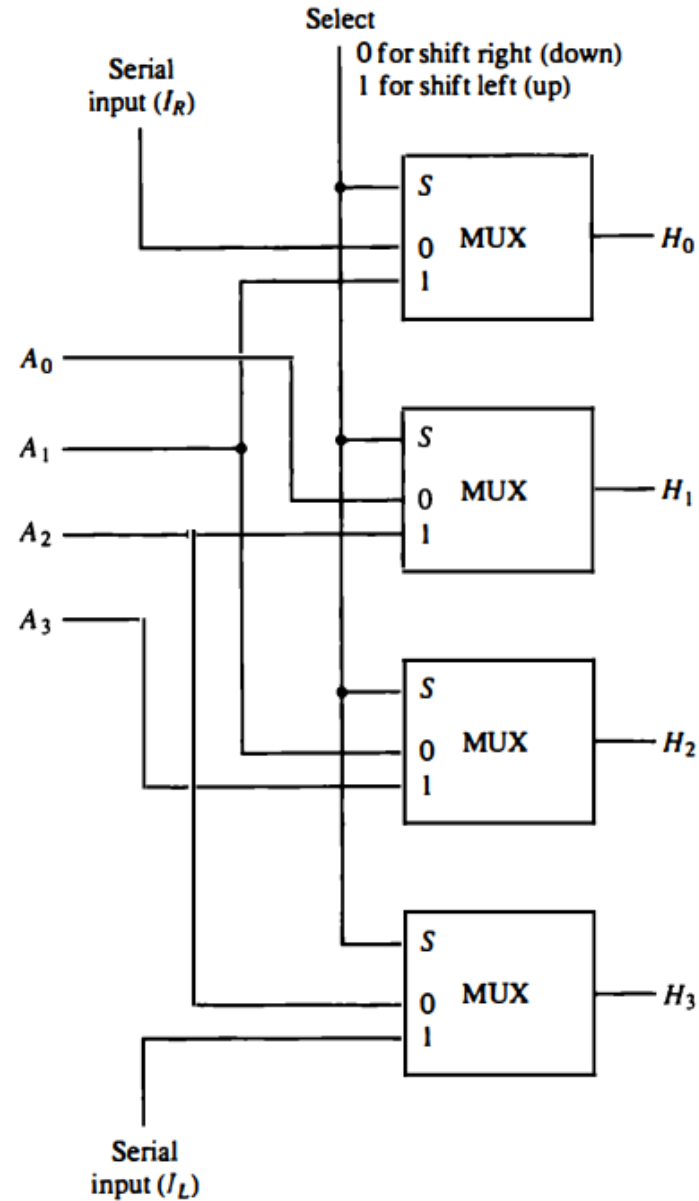
- **Arithmetic shift:** It shifts signed-binary number left or right.
- For **shift left** the content of the register is **multiplied by 2**.
- For **shift right** the content of the register is **divided by 2**.
- The arithmetic shift must leave the **sign bit unchanged**.
- It can be defined in RTL by:

$R \leftarrow \text{ashl } R$ arithmetic shift-left register R

$R \leftarrow \text{ashr } R$ arithmetic shift-right register R



Microoperations → Shift Microoperations

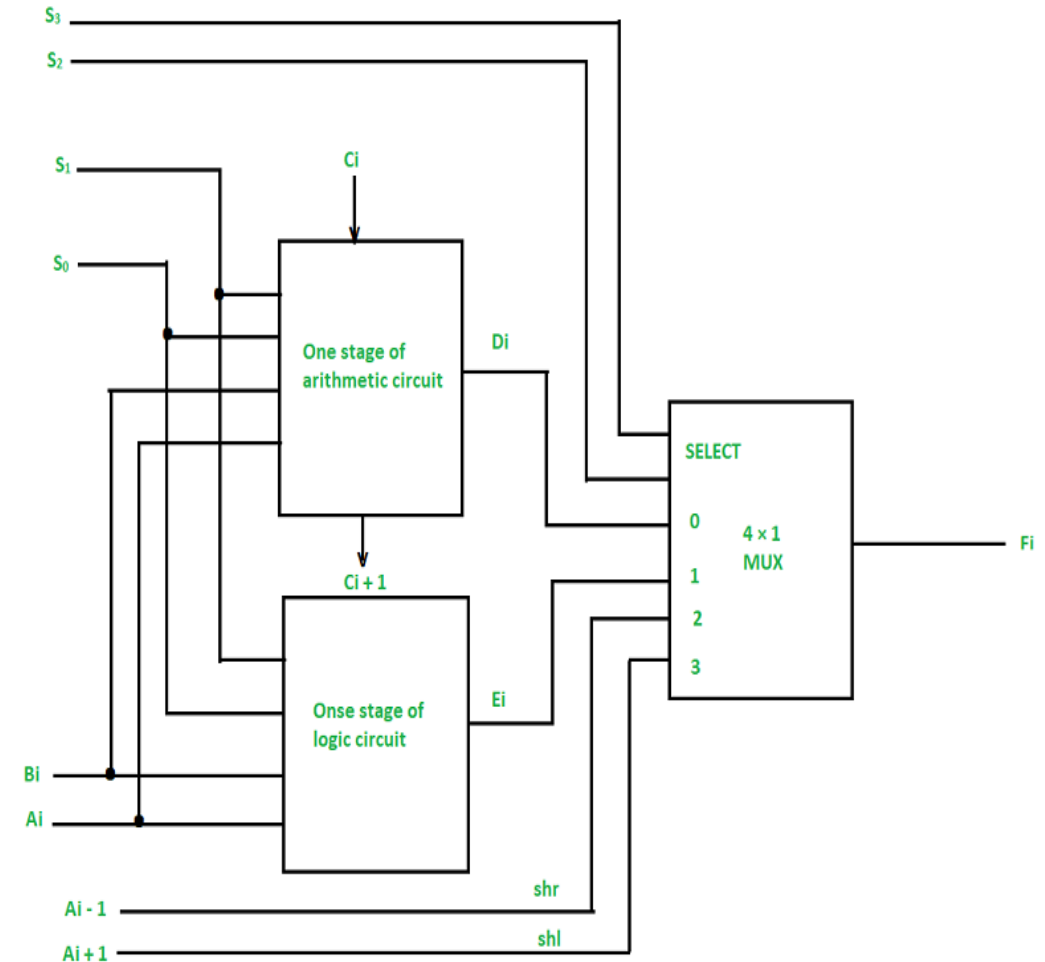


Hardware Implementation

Function table				
Select	Output			
	H_0	H_1	H_2	H_3
0	I_R	A_0	A_1	A_2
1	A_1	A_2	A_3	I_L

Arithmetic Logic Shift Unit

- Arithmetic logic shift unit is a digital circuit that performs **arithmetic calculations, logical manipulation and shift operation**.
- It is often abbreviated as **ALU**.
- The block diagram of ALU includes **one stage of arithmetic circuit, one stage of logic circuit and one 4*1 multiplexer**.
- The subscript *i* designates a typical stage.




Arithmetic Logic Shift Unit

- Inputs A_i and B_i are applied to both the arithmetic and logic units. A particular microoperation is selected with inputs S_1 and S_0 .
- A 4×1 MUX selects the final output. The two inputs of the MUX are received from the output of the arithmetic circuit and logic circuit.
- The other two is A_{i-1} for the shift-right operation and A_{i+1} for the shift left operation. The circuit is repeated n times for n -bit ALU.
- The output carry C_{i+1} is connected to the input carry C_{in} .
- In every stage the circuit specifies 8 arithmetic operations, 4 logical operations and 2 shift operations, where each operation is selected by the five variables S_3 , S_2 , S_1 , S_0 and C_{in} .

Arithmetic Logic Shift Unit

Operation select					Operation	Function
S_3	S_2	S_1	S_0	C_{in}		
0	0	0	0	0	$F = A$	Transfer A
0	0	0	0	1	$F = A + 1$	Increment A
0	0	0	1	0	$F = A + B$	Addition
0	0	0	1	1	$F = A + B + 1$	Add with carry
0	0	1	0	0	$F = A + \overline{B}$	Subtract with borrow
0	0	1	0	1	$F = A + \overline{B} + 1$	Subtraction
0	0	1	1	0	$F = A - 1$	Decrement A
0	0	1	1	1	$F = A$	Transfer A
0	1	0	0	\times	$F = A \wedge B$	AND
0	1	0	1	\times	$F = A \vee B$	OR
0	1	1	0	\times	$F = A \oplus B$	XOR
0	1	1	1	\times	$F = \overline{A}$	Complement A
1	0	\times	\times	\times	$F = \text{shr } A$	Shift right A into F
1	1	\times	\times	\times	$F = \text{shl } A$	Shift left A into F

A vibrant image of the Aurora Borealis (Northern Lights) in shades of green and blue, set against a dark, starry night sky. The aurora's light bands are dynamic and flowing, creating a sense of movement and natural beauty.

Next Lecture:

Chapter (6): Programming the Basic Computer