Unit-1

Computer Data Representation & Register Transfer and Micro-operations

Outline

- Basic computer data types & Complements
- Fixed point & Floating point representation
- Register transfer language
- Bus and Memory transfers (Three-State Bus Buffers, Memory Transfer)
- Arithmetic Micro-operations
- Logic Micro-operations
- Shift Micro-operations

- Arithmetic logical shift unit

Basic computer data types & Complements

Section - 1

Basic Computer Data Types & Complements

Basic Computer Data Types

- Decimal Number System
  - Radix(r) = 10, Number range = 0 - 9
- Binary Number System
  - Radix(r) = 2, Number range = 0 - 1
- Octal Number System

Complements

(r-1)'s complement

- 9's complement
- 1's complement
- 7's complement
- 15's complement
- r's

complement

- Radix(r) = 8, Number range = 0 - 7
  - 10’s complement
- Hexadecimal Number System
  - 2’s complement
  - Radix(r) = 16, Number range = 0 - 9 & A - F
  - 8’s complement
- Binary Coded Decimal Numbers
  - 16’s complement

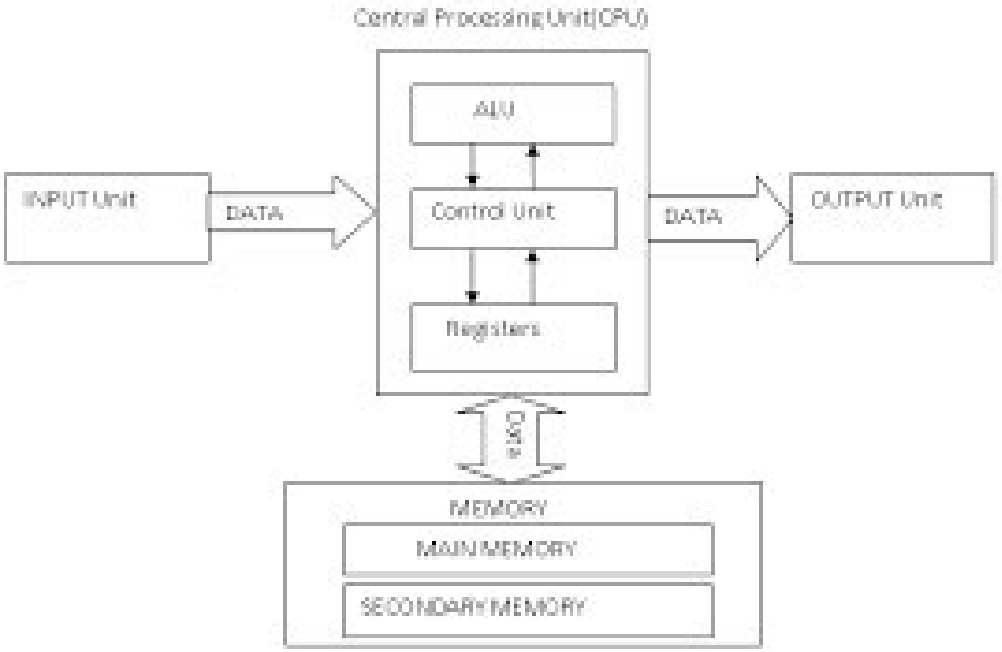
Computer Architecture	Computer Organization

Computer Architecture is concerned with the way hardware components are connected together to form a computer system.	Computer Organization is concerned with the structure and behaviour of a computer system as seen by the user.
It acts as the interface between hardware and software.	It deals with the components of a connection in a system.
Computer Architecture helps us to understand the functionalities of a system.	Computer Organization tells us how exactly all the units in the system are arranged and interconnected.

A programmer can view architecture in terms of instructions, addressing modes and registers.	Whereas Organization expresses the realization of architecture.
While designing a computer system architecture is considered first.	An organization is done on the basis of architecture.
Computer Architecture deals with high-level design issues.	Computer Organization deals with low-level design issues.
Architecture involves Logic (Instruction sets, Addressing modes, Data types, Cache	Organization involves Physical Components (Circuit design, Adders, Signals, Peripherals)

optimization)

FUNCTIONAL UNITS OF COMPUTER SYSTEM



- Input unit
- Input units are used by the computer to read the data. The most commonly used input devices are keyboards, mouse, joysticks, trackballs, microphones, etc.
- Central processing unit
- Central processing unit commonly known as CPU can be referred as an electronic circuitry within a computer that carries out the instructions given by a computer program by performing the basic arithmetic, logical, control and input/output (I/O) operations specified by the instructions.
- Memory unit

- The Memory unit can be referred to as the storage area in which programs are kept which are running, and that contains data needed by the running programs.
- The Memory unit can be categorized in two ways namely, primary memory and secondary memory
- Arithmetic & logical unit
- Most of all the arithmetic and logical operations of a computer are executed in the ALU (Arithmetic and Logical Unit) of the processor. It performs arithmetic operations like addition, subtraction, multiplication, division and also the logical operations like AND, OR, NOT operations.

- Control unit
- The control unit is a component of a computer's central processing unit that coordinates the operation of the processor. It tells the computer's memory, arithmetic/logic unit and input and output devices how to respond to a program's instructions.
- Output Unit
- The primary function of the output unit is to send the processed results to the user. Output devices display information in a way that the user can understand.

LOGIC GATES

- The logic gates are the main structural part of a digital system.

- Logic Gates are a block of hardware that produces signals of binary 1 or 0 when input logic requirements are satisfied.
- Each gate has a distinct graphic symbol, and its operation can be described by means of algebraic expressions.
- The seven basic logic gates includes: AND, OR, XOR, NOT, NAND, NOR, and XNOR.
- The relationship between the input-output binary variables for each gate can be represented in tabular form by a truth table.
- Each gate has one or two binary input variables designated by A and B and one binary output variable designated by x.
- AND GATE:

 The AND gate is an electronic circuit which gives a high output only if all its inputs are high.

The AND operation is represented by a dot (.) sign.

AND Gate:

Algebraic Function: x = AB

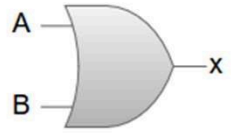
Truth Table:

A	B	x
0	0	0
0	1	0
1	0	0
1	1	1

OR GATE:

The OR gate is an electronic circuit which gives a high output if one or more of its inputs are high. The operation performed by an OR gate is represented by a plus (+) sign.

OR Gate:



Algebraic Function: x = A + B

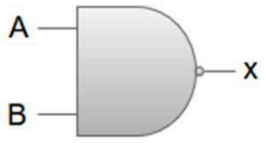
Truth Table:

A	B	x
0	0	0
0	1	1
1	0	1
1	1	1

- NOT GATE:
- The NOT gate is an electronic circuit which produces an inverted version of the input at its output. It is also known as an Inverter.
- NAND GATE:
- The NOT-AND (NAND) gate which is equal to an AND gate followed by a NOT gate. The NAND gate gives a high output if

any of the inputs are low. The NAND gate is represented by a AND gate with a small circle on the output. The small circle represents inversion.

NAND Gate:



Algebraic Function: x = (AB)'

Truth Table:

A	B	x
0	0	1
0	1	1
1	0	1
1	1	0

- NOR GATE:
- The NOT-OR (NOR) gate which is equal to an OR gate followed by a NOT gate. The NOR gate gives a low output if any of the

inputs are high. The NOR gate is represented by an OR gate with a small circle on the output. The small circle represents inversion.

NOR Gate:

Algebraic Function: x = (A+B)'

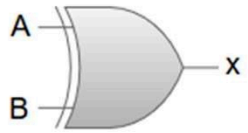
Truth Table:

A	B	x
0	0	1
0	1	0
1	0	0
1	1	0

- Exclusive-OR/ XOR GATE:
- The 'Exclusive-OR' gate is a circuit which will give a high output if one of its inputs is high but not both of them. The XOR operation

is represented by an encircled plus sign.

XOR Gate:



Algebraic Function: $x = A \oplus B$

or

$x = A'B + AB'$

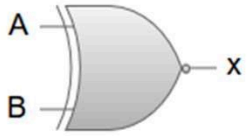
Truth Table:

A	B	x
0	0	0
0	1	1
1	0	1
1	1	0

- EXCLUSIVE-NOR/Equivalence GATE:
- The 'Exclusive-NOR' gate is a circuit that does the inverse operation to the XOR gate. It will give a low output if one of its inputs is high but not both of them. The small circle represents

inversion.

Exclusive-NOR Gate:



Algebraic Function: $x = (A \oplus B)'$

or

$x = A'B' + AB$

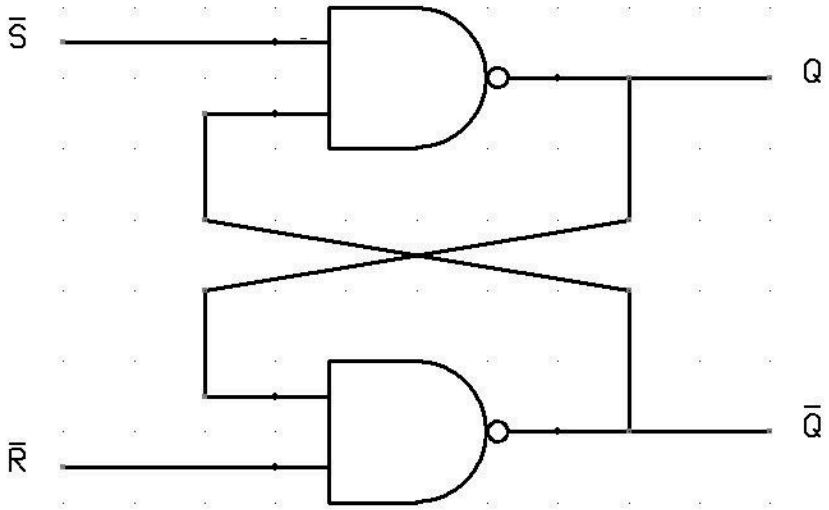
Truth Table:

A	B	x
0	0	1
0	1	0
1	0	0
1	1	1

FLIP FLOP

- A flip flop in digital electronics is a circuit with two stable states that can be used to store binary data. The stored data can be changed by applying varying inputs. Flip-flops and latches are

fundamental building blocks of digital electronics systems used in computers, communications, and many other types of systems. Both are used as data storage elements.



Register transfer language

Section - 3

Register

- A register is a group of flip-flops with each flip-flop capable of storing one bit of information.
- An n-bit register has a group of n flip-flops and is capable of storing any binary information of n bits

- In addition to the flip-flops, a register may have combinational gates that perform certain dataprocessing tasks.
- In its broadest definition, a register consists of a group of flip-flops and gates that effect their transition. The flip-flops hold the binary information and the gates control when and how new information is transferred into the register.
- Various types of registers are available commercially. The simplest register is one that consists only of flip-flops, with no

REGISTERS

external gates

- A Register is a fast memory used to accept, store, and transfer data and instructions that are being used immediately by the CPU.
- A Register can also be considered as a group of flip-flops with each flip-flop capable of storing one bit of information.
- A register with n flip-flops is capable of storing binary information of n-bits.

- The flip-flops contain the binary information whereas the gates control the flow of information,
  i.e. when and how the information's are transferred into a register.
- Different types of registers are available commercially. A simple register consists of only flipflops with no external gates.
- The transfer of new data into a register is referred to as loading the register.

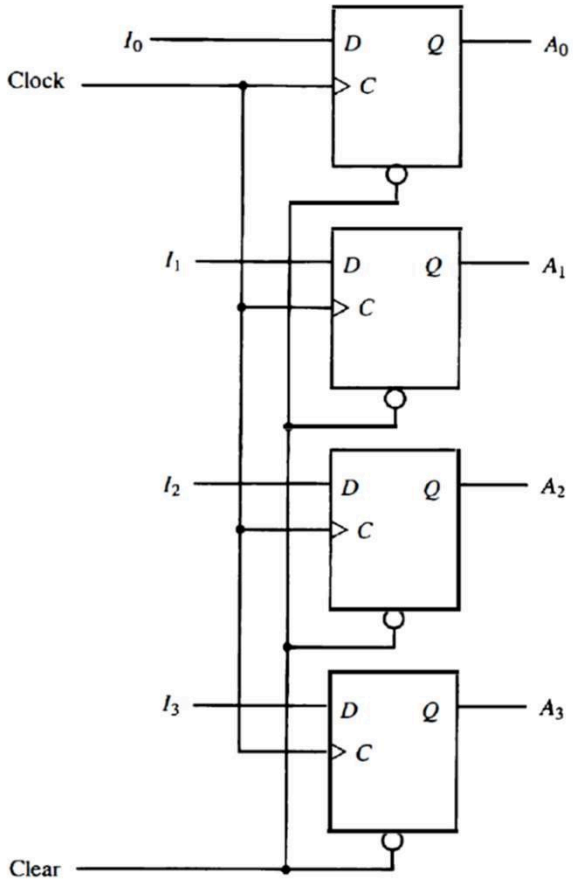


Figure 2-6 4-bit register.

- The four outputs can be sampled at any time to obtain the binary information stored in the register.
- The clear input goes to a special terminal in each flip-flop. When this input goes to 0, all flipflops are reset asynchronously. The clear input is useful for clearing the register to all O's prior to its clocked operation.
- The clear input must be maintained at logic 1 during normal clocked operation.
- Note that the clock signal enables the D input but that the clear. input is independent of the clock.

- Digital modules are best defined by the register & they contain and the operations that are performed on the data stored in them. The operations executed on data stored in registers are called microoperations.
- 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 transformed to another register.
- Examples of microoperations are shift, count, clear, and load.

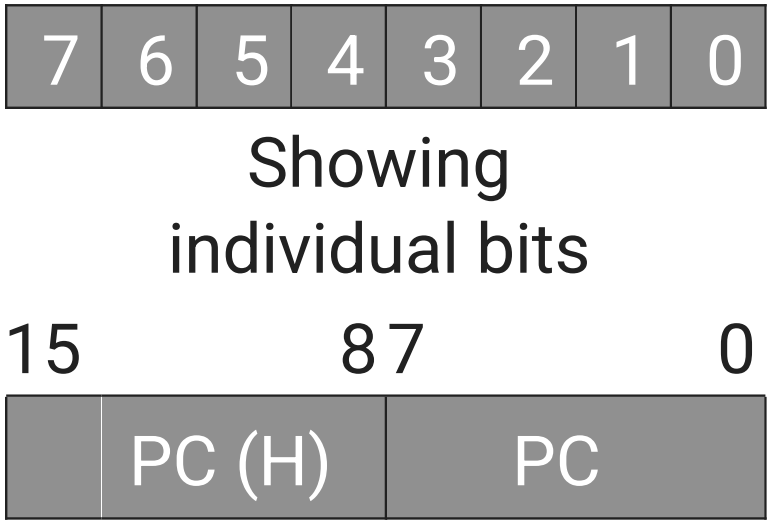
 The internal hardware organization of a digital computer is best defined by specifying:

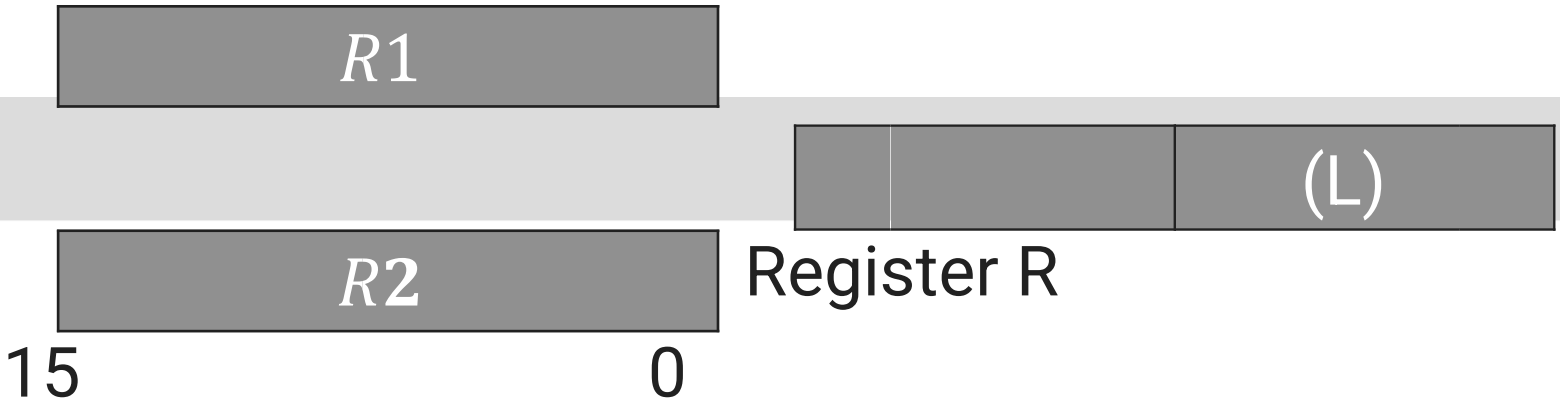
- 1. The set of registers it contains and their function.
- 2. The sequence of microoperations performed on the binary information stored in the registers.
- 3. The control that initiates the sequence of microoperations.

Register

 Computer Registers are designated by capital letters. For example,

- MAR – Memory Address Register
- PC – Program Counter
- IR – Instruction Register
- R1 – Processor Register





Numbering of bits

Divided into two parts

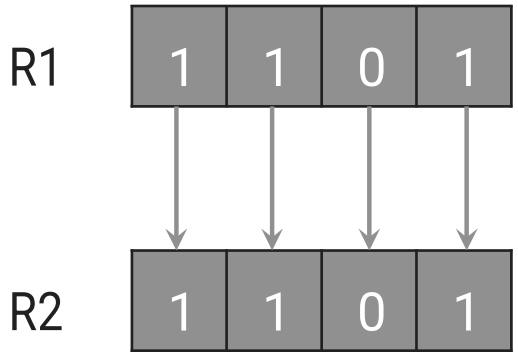
Register Transfer Language

- Information transfer from one register to another is designated in symbolic form by means of a replacement operator is known as Register Transfer.
- The statement

$$R2 \leftarrow R1$$

- denotes a transfer of the

contentof
register $R1$ into register $R2$.



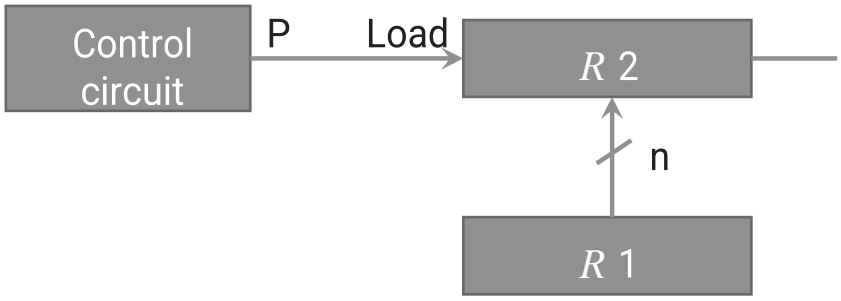
- The symbolic notation used to describe the microoperation transfers among registers is called a register transfer language.
- The term "register transfer" implies the availability of hardware logic circuits that can perform a stated microoperation and transfer the result of the operation to the same or another register.
- A register transfer language is a system for expressing in symbolic form the microoperation sequences among the registers of a digital module.

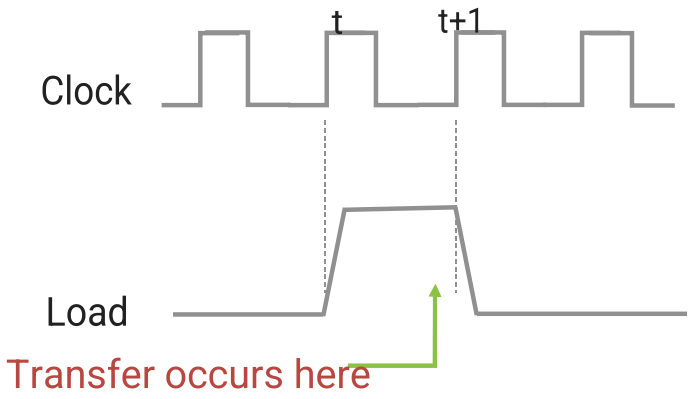
Register Transfer with Control Function

- Normally, we want the transfer to occur only under a predetermined control condition using ifthen statement.

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

$$P : R2 \leftarrow R1$$





- where P is a control signal generated in the control section.
- A control function is a boolean variable that is equal to 1 or 0. The control function is included in the statement as follows:

- The basic symbols o f the register transfer notation are listed in Table .Registers are denoted by capital letters, and numerals may follow the letters.
- Parentheses are used to denote a part of a register by specifying the range of bits or by giving a symbol name to a portion of a

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

TABLE 4-1 Basic Symbols for Register Transfers

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

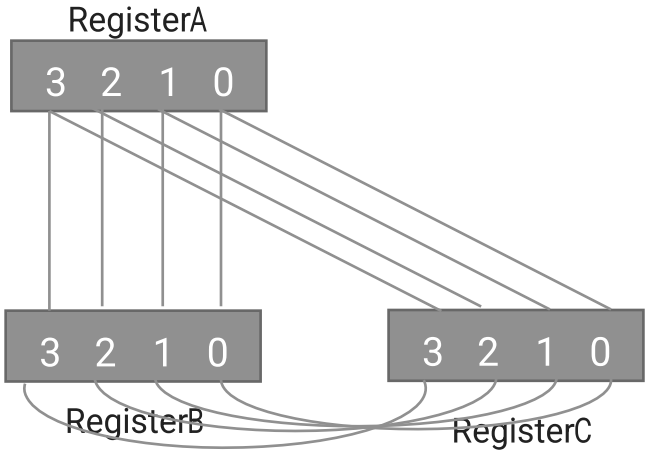
register.

- The arrow denotes a transfer of information and the direction of transfer.
- A comma is used to separate two or more operations that are

executed at the same time.

Bus and Memory transfers

Section - 4 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

Common Bus System for 4 registers

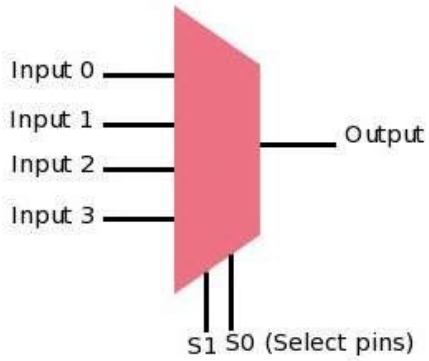
each bit of a register, through which binary information is transferred one at a time.

- One way of constructing a common bus system is with multiplexers.
- The multiplexers select the source register whose binary information is then placed on the bus.

Multiplexer

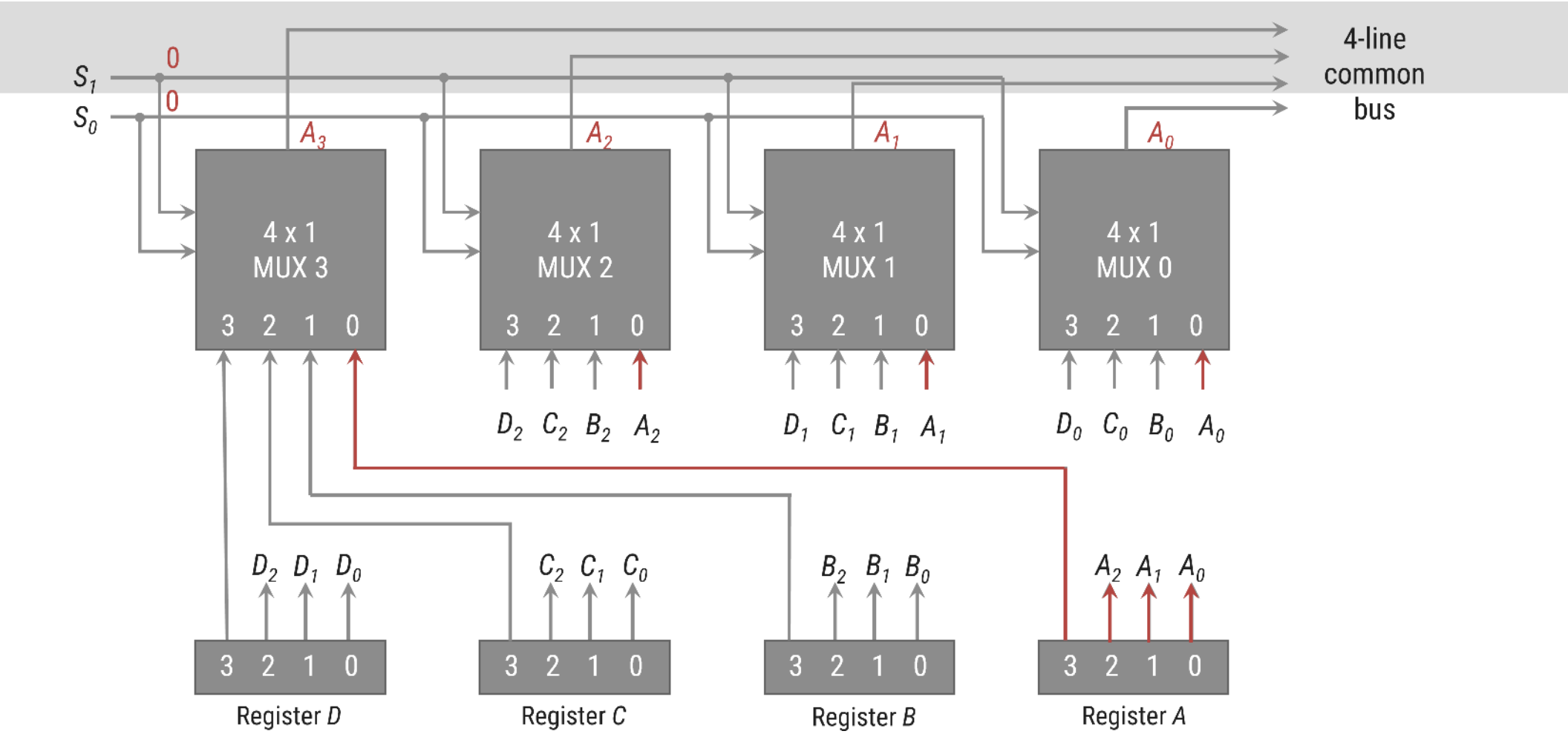
- A Multiplexer is a device that allows one of several analog or digital input signals which are to be selected and transmits the input that is selected into a single medium.
- Multiplexer is also known as Data Selector.
- A multiplexer of 2^n inputs has n select lines that will be used to

select input line to send to the output. Multiplexer is abbreviated as Mux.



Select	Inputs		Output
0	0	0	0
0	0	1	1
1	1	0	1
1	1	1	1

Common Bus System for 4 registers



Common Bus System for 4 registers

- The construction of a bus system for four registers is explained earlier.
- Each register has four bits, numbered 0 through 3.
- The bus consists of four 4 x 1 multiplexers each having four data inputs, 0 through 3, and two selection inputs, S_1 and S_0.
- The diagram shows that the bits in the same significant position in each register are connected to the data inputs of one multiplexer to form one line of the bus.
- The two selection lines S_1 and S_0 are connected to the selection inputs of all four multiplexers.

Common Bus System for 4 registers

- The selection lines choose the four bits of one register and transfer them into the four-line common bus.
- When S_1S_0 = 00, the 0 data inputs of all four multiplexers are selected and applied to the outputs that form the bus.
- This causes the bus lines to receive the content of register A since the outputs of this register are connected to the 0 data inputs of the multiplexers.
- Table shows the register that is selected by the bus for each of the four possible binary values of

the selection lines.

- 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 x 1 since it multiplexes k data lines.

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

Common Bus System for 4 registers

- For example, a common bus for eight registers of 16 bits requires

  Multiplexers - 16 of (8 x 1)

  Select Lines - 3

- 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.
- The symbolic statement for a bus transfer may mention the bus or its presence may be implied in the statement. When the bus is

includes in the statement, the register transfer is symbolized as follows:

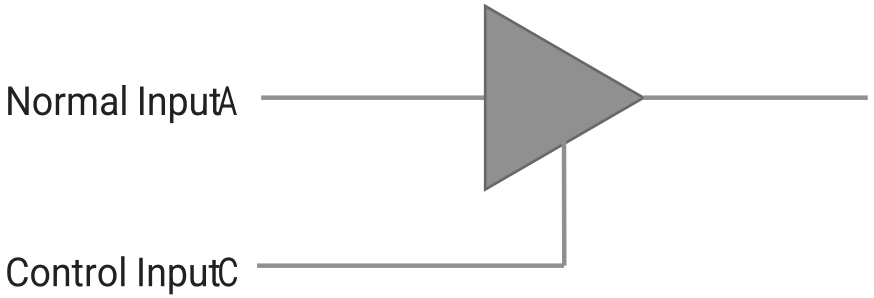
- The content of register C is placed on the bus, and the content of the bus is into register R 1 by activating its load control input

$$BUS \leftarrow C, \qquad R1 \leftarrow BUS$$

Tri-state Buffer (3 state Buffer)

- A three-state gate is a digital circuit that exhibits three states.
- Two of the states are signals equivalent to logic 1 and 0 as in a conventional gate.
- The third state is high impedance state which behaves like an open circuit, which means that the output is disconnected and does not have logic significance.
- The control input determines the output state. When the control input C is equal to 1, the output is enabled and the gate behaves like any conventional buffer, with the output equal to the normal input.

- When the control input C is 0, the output is disabled and the gate goes to a high-impedance state, regardless of the value in the normal input.

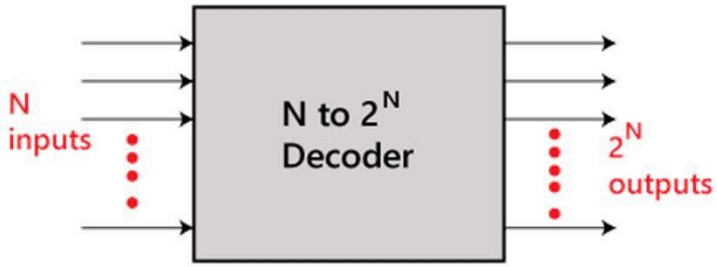


Output Y = A if C =1
High-impedance if C = 0

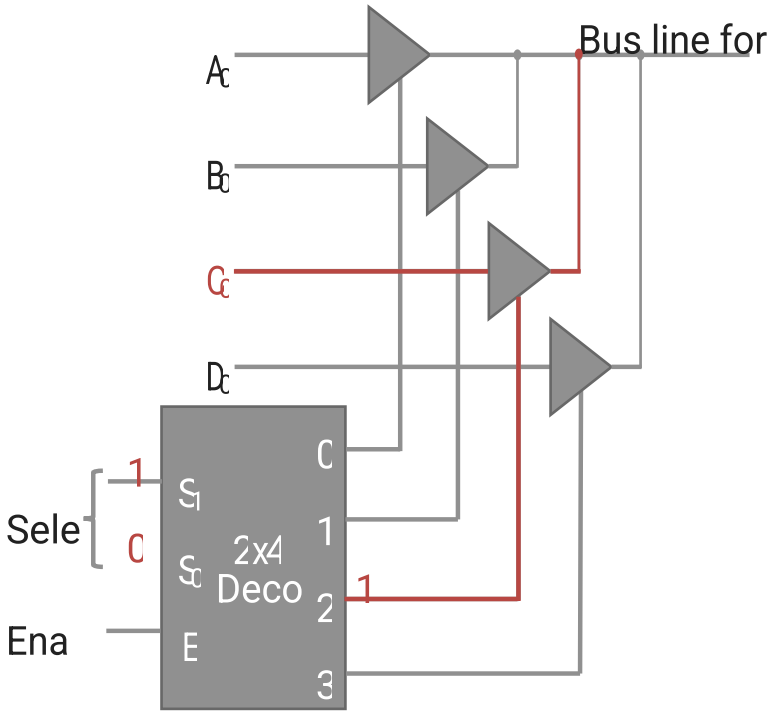
Decoder

- The combinational circuit that change the binary information into 2^N output lines is known as Decoders.
- The binary information is passed in the form of N input lines.

- The output lines define the 2^N-bit code for the binary information. In simple words, the Decoder performs the reverse operation of the Encoder.
- At a time, only one input line is activated for simplicity. The produced 2^N-bit output code is equivalent to the binary information.



Common Bus System using Decoder and Tri-state Buffer



Common Bus System using Decoder and Tri-state Buffer

- The construction of a bus system with three-state buffers is demonstrated in previous figure.
- The outputs of four buffers are connected together to form a single bus line.
- The control inputs to the buffers determine which of the four normal inputs will communicate with the bus line.
- The connected buffers must be controlled so that only one three -state buffer has access to the bus line while all other buffers are maintained in a high impedance state.

- One way to ensure that no more than one control input is active at any given time is to use a decoder, as shown in the figure: Bus line with three state-buffers.
- When the enable input of the decoder is 0, all of its four outputs are 0, and the bus line is in a high-impedance state because all four buffers are disabled.
- When the enable input is active, one of the three-state buffers will be active, depending on the binary value in the select inputs of the decoder.

Memory Transfer

- The transfer of information from a memory unit to the user end

is called a Read operation.

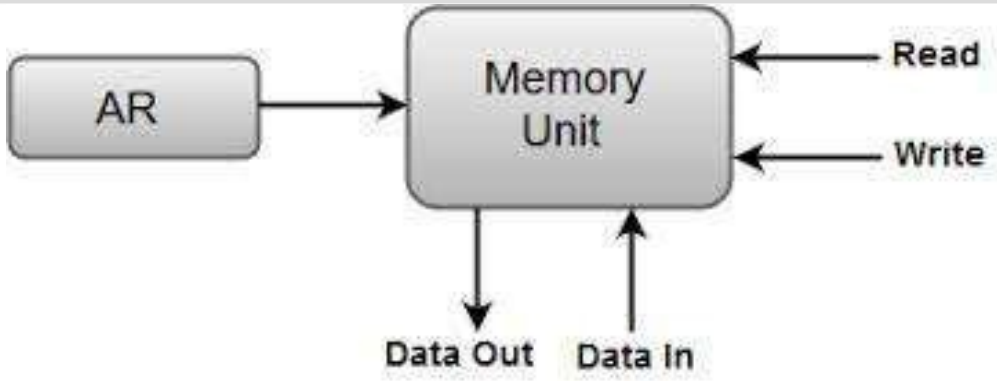
- The transfer of new information to be stored in the memory is called a Write operation.
- A memory word is designated by the letter M.
- We must specify the address of memory word while writing the memory transfer operations.
- The address register is designated by AR and the data register by DR.
- Thus, a read operation can be stated as:

Read: DR ← M [AR]

- The Read statement causes a transfer of information into the data register (DR) from the memory word (M) selected by the address register (AR). • And the corresponding write operation can be stated as:

  Write: M [AR] ← R1

- The Write statement causes a transfer of information from register R1 into the memory word (M) selected by address register (AR).



Arithmetic Micro-operations

Section - 5 In general, the Arithmetic Micro-operations deals with the operations performed on numeric data stored in the registers.

 The basic Arithmetic Micro-operations are classified in the following categories:

- Addition
- Subtraction
- Increment
- Decrement

- Shift
- Some additional Arithmetic Micro-operations are classified as:
- Add with carry
- Subtract with borrow
- Transfer/Load, etc.

Arithmetic Microoperations

- Arithmetic microoperations perform arithmetic operations on numeric data stored in registers.

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)

Add Microoperation

$R3 \leftarrow R1 + R2$

Subtract Microoperation

$R2 \leftarrow \overline{R2} + 1$	R2
$R3 \leftarrow R1 + \overline{R2} + 1$	R1, R2
$R1 \leftarrow R1 + 1$	R1
$R1 \leftarrow R1 - 1$	R1

$R3 \leftarrow R1 - R2$

$R3 \leftarrow R1 + \overline{R2} + 1$

2's complement the contents of R2

(negate) plus R1 the 2's complement of

(subtraction)

Increment the content of by one

Decrement the content of by one

- The arithmetic operations of multiply and divide are not listed in Table These two operations are valid arithmetic operations but are not included in the basic set of microoperations.
- The only place where these operations can be considered as

microoperations is in a digital system, where they are implemented by means of a combinational circuit.

- In such a case, the signals that perform these operations propagate through gates, and the result of the operation can be transferred into a destination register by a clock pulse as soon as the output signal propagates through the combinational circuit.
- In most computers, the multiplication operation is implemented with a sequence of add and shift microoperations.
- Division is implemented with a sequence of subtract and shift

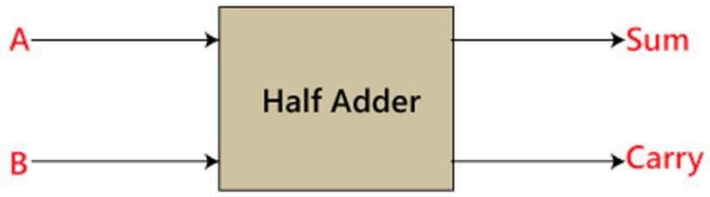
microoperations.

HALF ADDER

- A half adder is a digital logic circuit that performs binary addition of two single-bit binary numbers. It has two inputs, A and B, and two outputs, SUM and CARRY.
- The SUM output is the least significant bit (LSB) of the result, while the CARRY output is the most significant bit (MSB) of the result, indicating whether there was a carry-over from the addition of the two inputs.
- The half adder can be implemented using basic gates such as

XOR and AND gates.

- The Half-Adder is a basic building block of adding two numbers as two inputs and produce out two outputs. The adder is used to perform OR operation of two single bit binary numbers.
- The augent and addent bits are two input states, and 'carry' and 'sum 'are two output states of the half adder.



- 'A' and' B' are the input states, and 'sum' and 'carry' are the output

states.

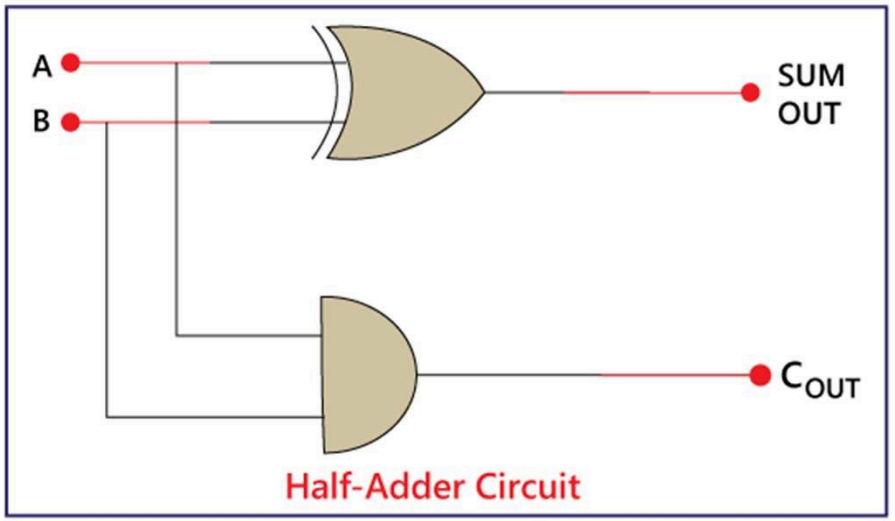
- The carry output is 0 in case where both the inputs are not 1.
- The least significant bit of the sum is defined by the 'sum' bit.
- The SOP form of the sum and carry are as follows:

- Sum = x'y+xy'
- Carry = xy
- n the block diagram, we have seen that it contains two inputs and two outputs. The augent and addent

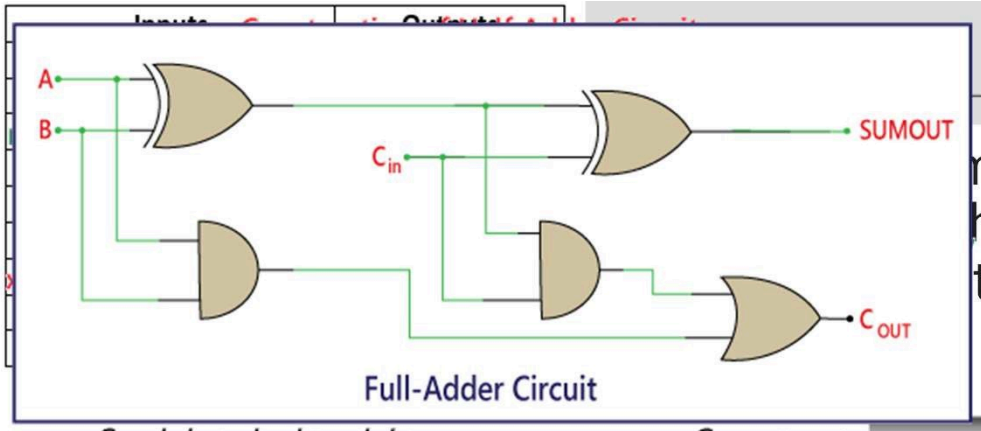
Inputs		Outputs	
A	B	Sum	Carry
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

bits are the input states, and carry and sum are the output states of the half adder. The half adder is designed with the help of the following two logic gates:

- 2-input AND Gate.
- 2-input Exclusive-OR Gate or Ex-OR Gate



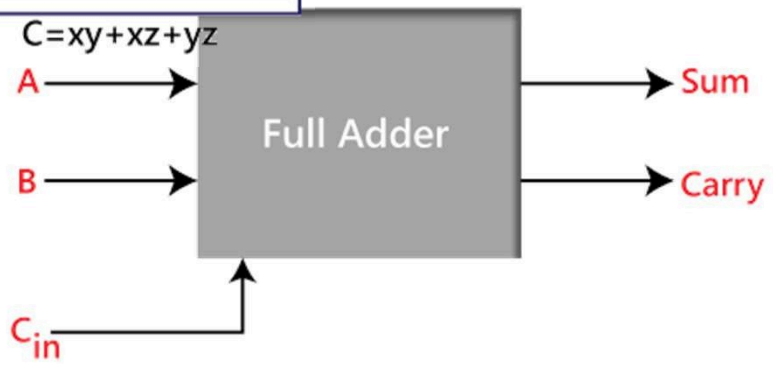
Half-Adder Circuit



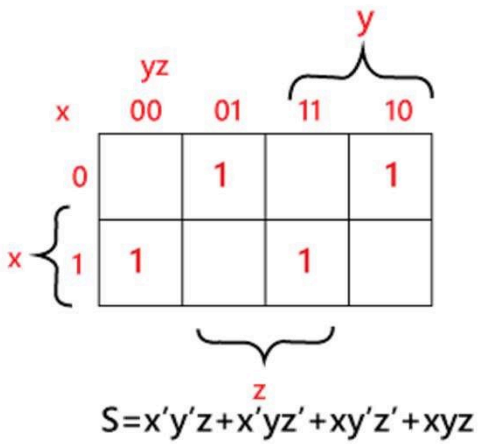
mbers. To overcome thi me this
hree 1-bit binary numbe umbers
t states i.e., sum and c m and

$S=x'y'z+x'yz'+xy'z'+xyz$

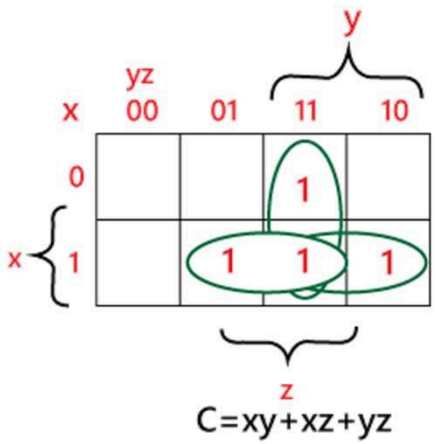
$C=xy+xz+yz$



- 'A' and' B' are the input variables. These variables represent the two significant bits which are going to be added
- 'Cin' is the third input which represents the carry. From the previous lower significant position, the carry bit is fetched.
- The 'Sum' and 'Carry' are the output variables that define the



S=x′y′z+x′yz′+xy′z′+xyz



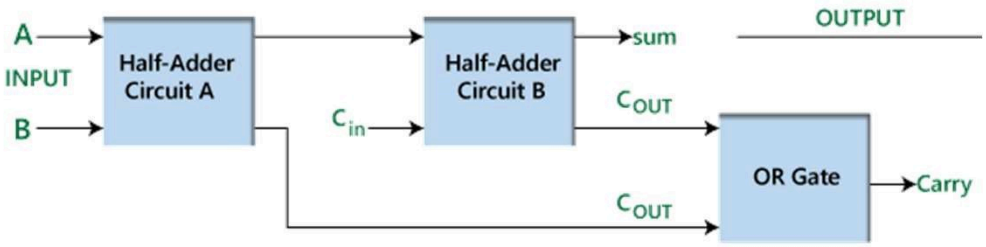
C=xy+xz+yz

Inputs			Outputs	
A	B	C_{in}	Sum	Carry
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

output values.

- The eight rows under the input variable designate all possible combinations of 0 and 1 that can occur in these variables.

Construction of Half Adder Circuit:



- In the above circuit, there are two half adder circuits that are combined using the OR gate.
- The first half adder has two single-bit binary inputs A and B. As we know that, the half adder produces two outputs, i.e., Sum and Carry.
- The 'Sum' output of the first adder will be the first input of the second half adder, and the 'Carry' output of the first adder will be the second input of the second half adder. The second half adder will again provide 'Sum' and 'Carry'.
- The final outcome of the Full adder circuit is the 'Sum' bit. In order to find the final output of the 'Carry', we provide the 'Carry'

output of the first and the second adder into the OR gate. The outcome of the OR gate will be the final carry out of the full adder circuit.

- The MSB is represented by the final 'Carry' bit.

Sum:

Perform the XOR operation of input A and B.

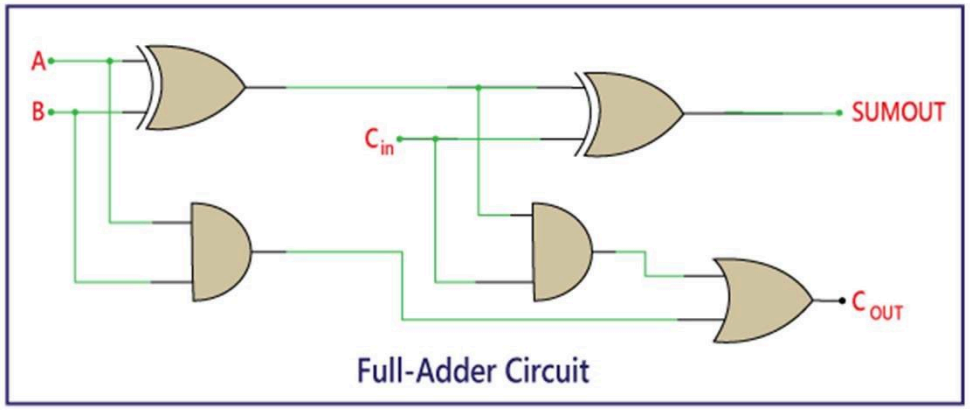
Perform the XOR operation of the outcome with carry. So, the sum is (A XOR B) XOR Cin which is also represented as:

$(A \oplus B) \oplus Cin$

Carry:

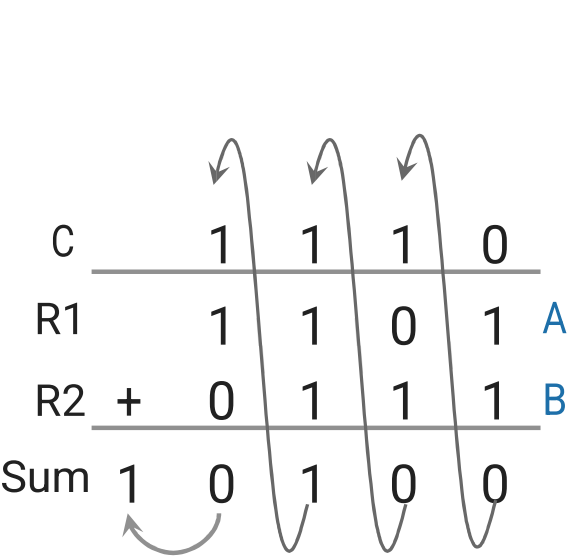
Perform the 'AND' operation of input A and B.

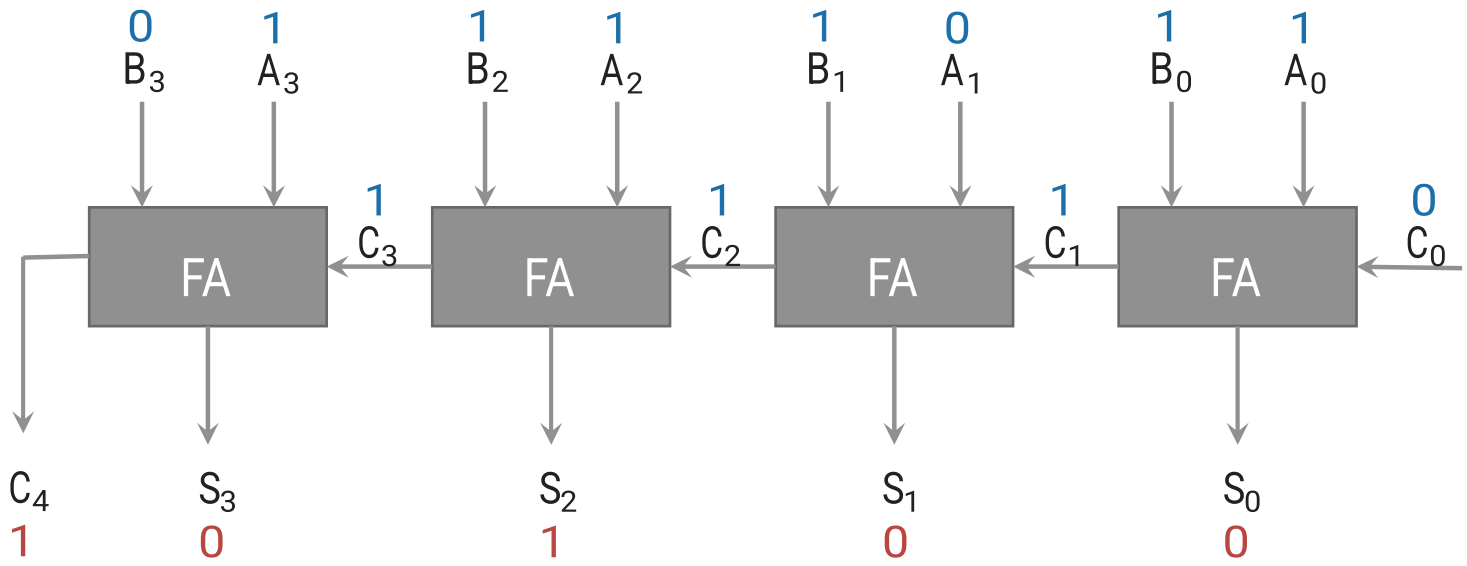
Perform the 'XOR' operation of input A and B.
Perform the 'OR' operations of both the outputs that come from the previous two steps. So the 'Carry' can be represented as:
$A.B + (A \oplus B)$



Full-Adder Circuit

4-bit Binary Adder





- The digital circuit that generates the arithmetic sum of two binary numbers of any length is called a binary adder.

- Example

4-bit binary adder

- The binary adder is constructed with full-adder circuits connected in cascade, with the output carry from one full-adder connected to the input carry of the next full-adder.
- The figure shows the interconnections of four full-adders (FA) to provide a 4-bit binary adder.
- The augends bits of A and the addend bits of B are designated by subscript numbers from right to left, with subscript 0 denoting the low-order bit.
- The carries are connected in a chain through the full-adders.

- The input carry to the binary adder is C0 and the output carry is C4.
- The S outputs of the full-adders generate the required sum bits.
- An n-bit binary adder requires n full-adders.
- The output carry from each full-adder is connected to the input carry of the next-high-order fulladder.
- The n data bits for the A inputs come from one register (such as R1), and the n data bits for the B inputs come from another register (such as R2). The sum can be transferred to a third register or to one of the source registers (R1 or R2), replacing its previous content.

r’s complement

- Generally, there are two types of complement of Binary number: 1’s complement and 2’s complement.
- To get 1’s complement of a binary number, simply invert the given number. For example, 1’s complement of binary number 110010 is 001101.
- To get 2’s complement of binary number is 1’s complement of given number plus 1 to the least significant bit (LSB). For example 2’s complement of binary number 10010 is (01101) + 1 = 01110.
- Uses of 1’s Complement Binary Numbers:

- There are various uses of 1’s complement of Binary numbers
- mainly in signed Binary number representation and various arithmetic operations for Binary numbers, e.g., additions, subtractions, etc.
- he signed bit representation is a way to indicate the sign of a binary number using a single bit. This single bit, often called the "sign bit," is positioned at the leftmost side of the binary number. It determines whether the number is positive or negative.
- In this representation:
- The sign bit of 0 indicates a positive number.
- The sign bit of 1 indicates a negative number.

- Let's illustrate this with an example using a 4-bit signed bit representation:
- Positive Number (5): To represent the positive number 5, we set the sign bit to 0 and then write the binary representation of the number itself: 0101.
- Negative Number (-3): To represent the negative number -3, we set the sign bit to 1 and then write the binary representation of the positive version of the number (in this case, the positive version of 3): 1100

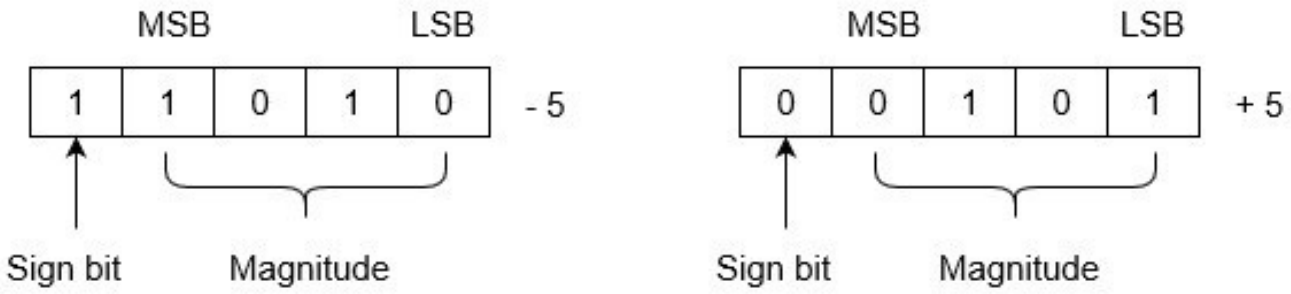
- Here's the breakdown:
- Positive 5: 0101
  - Sign bit: 0 (positive)
  - Magnitude: 0101 (binary for 5)
- Negative -3: 1100

- Sign bit: 1 (negative)
- Magnitude: 100 (binary for positive 3)

- 1’s Complementation in Signed Binary number Representation:
- 1’s complement binary numbers are very useful in Signed number representation. Positive numbers are simply

represented as Binary number .

- There is nothing to do for positive binary number.
- But in case of negative binary number representation, we represent in 1's complement. If the number is negative then it is represented using 1's complement. First represent the number with positive sign and then take 1's complement of that number.



Negative number in binary

- a simple approach that adds an extra bit (i.e., sign-bit) to detect the sign of a number. 11 indicates a -ve number, and 00 indicates a +ve number or vice versa (depending on the architecture of the computer).
- For example, if 7 = (0111)7=$(0111)_2$ then -7 = (1111)−7=$(1111)_2$.
- A drawback is that the adders, in the underlying hardware of the

computer, need to determine the sign-bit of an operation's result. Due to this, and other disadvantages, the sign-bit representation of negative numbers is now obsolete.

- In this representation, the left-most bit is considered to be the sign-bit (without adding an extra bit), where 11 is a -ve and 00 is a +ve. This reduces the range of positive numbers that can be

represented (using n bits) from $2^n - 1$ to $2^{n-1} - 1$ Despite this drawback, it is now the standard way of representing negative binary numbers.

- To convert a positive number into a negative number, using the two's complement representation, invert all of the bits of the number and add 1.

- For example:-
- 13:- 0000 1101 ⟶ 1111 0010 ⟶ 1111 0011

Subtraction using 2's complement

- In the first step, find the 2's complement of the subtrahend.
- Add the complement number with the minuend.
- If we get the carry by adding both the numbers, then we discard this carry and the result is positive else take 2's complement of the result which will be negative.

 Example 1: 10101 - 00111

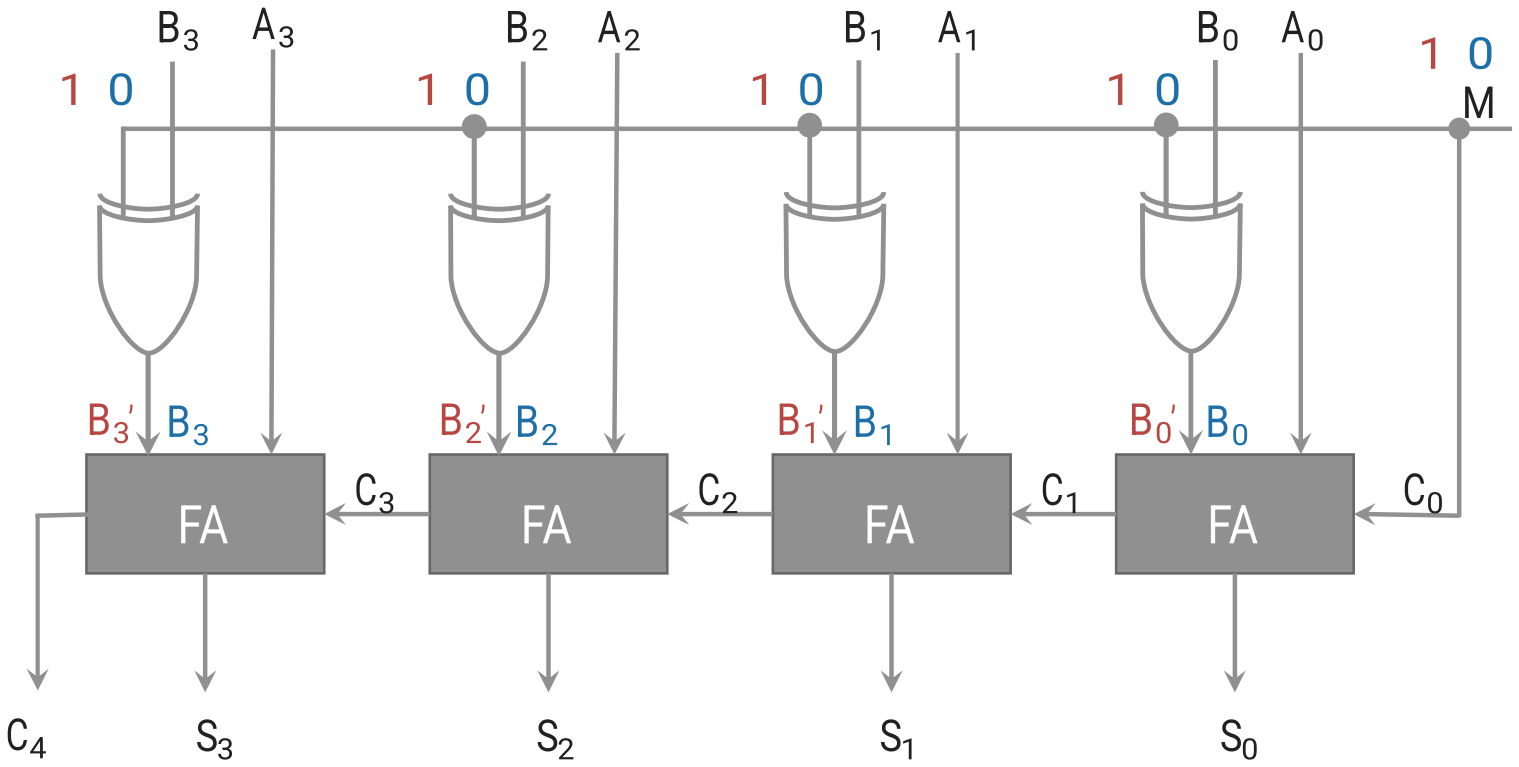
 We take 2's complement of subtrahend 00111, which is 11001. Now, sum them. So, 10101+11001 =1 01110.

 In the above result, we get the carry bit 1. So we discard this

carry bit and remaining is the final result and a positive number.

4-bit Binary Adder-Subtractor an when

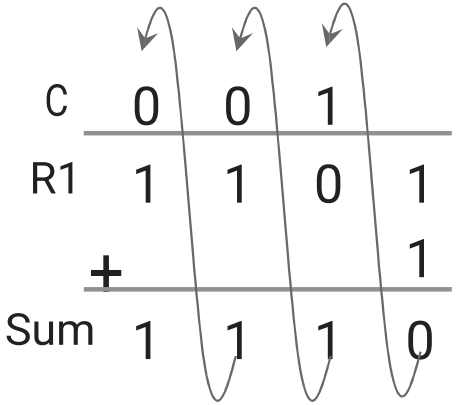
when M = 0 the circuit is Adder M = 1

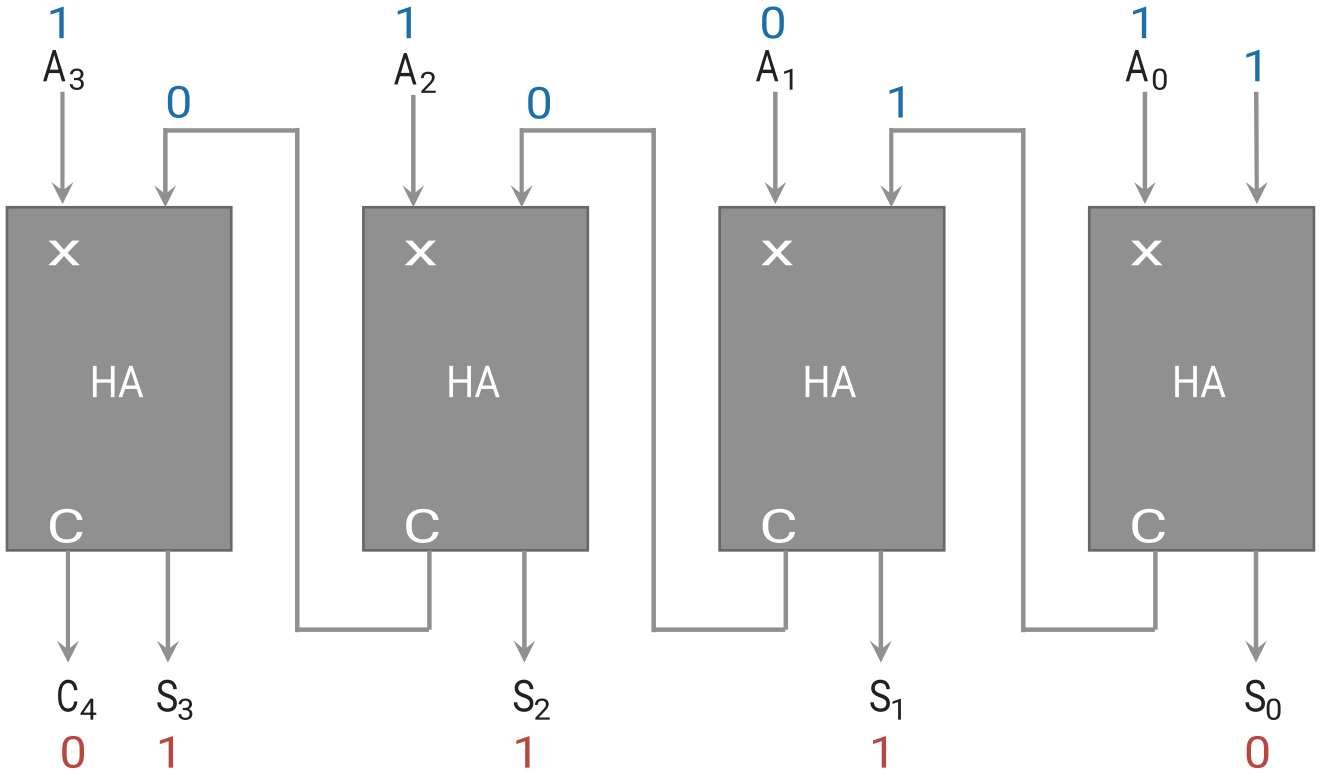


the circuit becomes a Subtractor When M = 0, we have $C_0 = 0 \& B \oplus 0 = B$ When M = 1, we have $C_0 = 1 \& B \oplus 1 = B'$

4-bit Binary Incrementer

- The increment microoperation adds one to a number in a register.





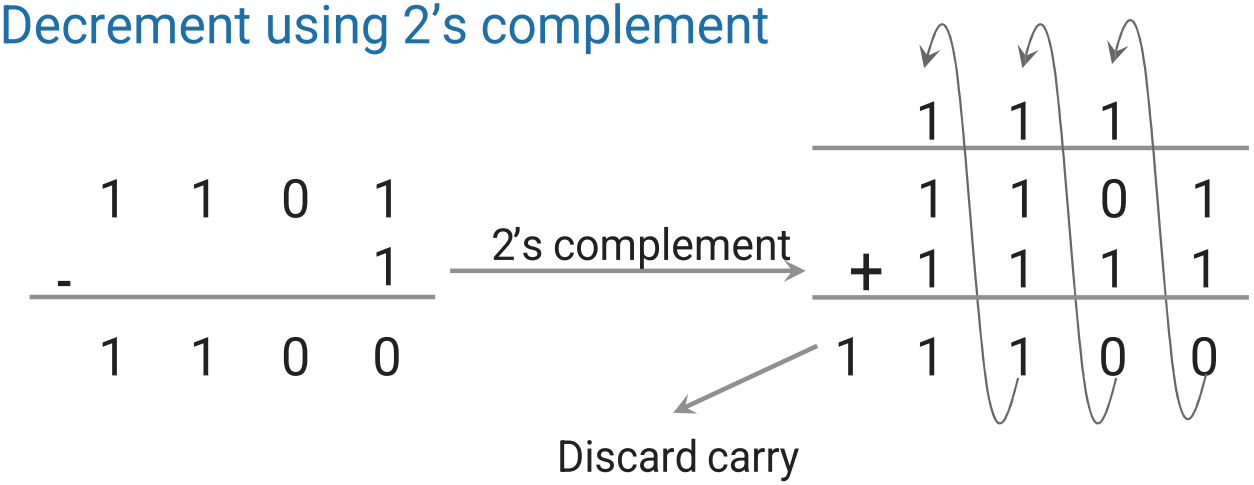
4-bit Arithmetic Circuit

- The arithmetic micro operations can be implemented in one composite arithmetic circuit.
- The basic component of an arithmetic circuit is the parallel adder.
- By controlling the data inputs to the adder, it is possible to obtain different types of arithmetic operations.
- The output of binary adder is calculated from arithmetic sum.

4-bit Arithmetic Circuit

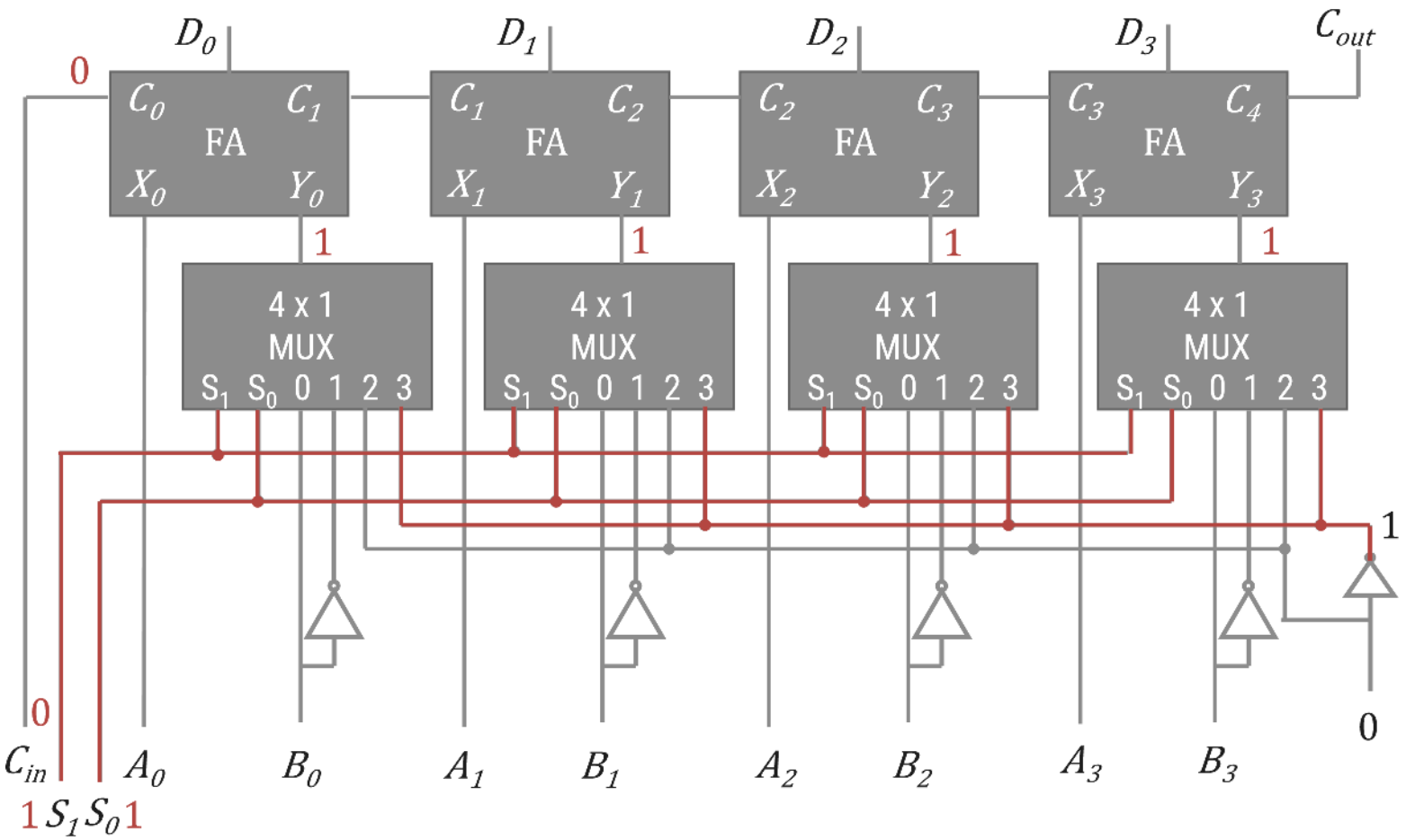
$$D = A + Y + C_{in}$$

Decrement using 2's complement



$$Decrement = A + 1111 + 0$$

4-bit Arithmetic Circuit



4-bit Arithmetic Circuit

Hardware implementation consists of:

- 4 full-adder circuits that constitute the 4-bit adder and four multiplexers for choosing different operations.
- There are two 4-bit inputs A and B.
- The four inputs from A go directly to the X inputs of the binary adder. Each of the four inputs from B is connected to the data inputs of the multiplexers. The multiplexer's data inputs also receive the complement of B.
- The other two data inputs are connected to logic-0 and logic-1.
  - Logic-0 is a fixed voltage value (0 volts for TTL integrated circuits)

4-bit Arithmetic Circuit

- Logic-1 signal can be generated through an inverter whose input is 0.

- The four multiplexers are controlled by two selection inputs, S_1 and S_0.
- The input carry C_{in} goes to the carry input of the FA in the least significant position. The other carries are connected from one stage to the next.
- 4-bit output $D_0 \ldots D_3$
- When $S_1S_0 = 00$
  - If $C_{in} = 0$ then $D = A + B$; Add

4-bit Arithmetic Circuit

- If C_{in} = 1 then D = A + B + 1; Add with carry

- When S_1S_0 = 01
  - If C_{in} = 0 then D = A + B; Subtract with borrow
  - If C_{in} = 1 then D = A + B + 1; A + 2's complement of B i.e. A - B

- When S_1S_0 = 10
  - Input B is neglected and all 0's are inserted to Y inputs

    D = A + 0 + C_{in}
    - If C_{in} = 0 then D = A; Transfer A
    - If C_{in} = 1 then D = A + 1; Increment A

4-bit Arithmetic Circuit

- When $S_1S_0 = 11$
  - Input B is neglected and all 1's are inserted to Y inputs

    $D = A - 1 + C_{in}$
    - If $C_{in} = 0$ then $D = A - 1$; 2's compliment
    - If $C_{in} = 1$ then $D = A$; Transfer A
- Arithmetic Circuit Function

S_1	S_0	C_{in}	Y	$D = A + Y + C_{in}$	Microoperation
0	0	0	B	$D = A + B$	Add
0	0	1	B	$D = A + B +$	Add with carry

4-bit Arithmetic Circuit

				1	
0	1	0	B’	D = A + B’	Subtract with borrow
0	1	1	B’	D = A + B’ + 1	Subtract
1	0	0	0	D = A	Transfer
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

Logic Micro-operations

Section - 6

Logic Microoperations

- Logic micro operations specify binary operations for strings of bits stored in registers.
- These operations consider each bit of the register separately and treat them as binary variables.
- It specifies a logic microoperation to be executed on the individual bits of the registers provided that the control variable

P = 1

- Example

$$P\colon\ R1 \leftarrow R1 \oplus R2$$

R1		1	0	1	0
R2	$\oplus$	1	1	0	0
R1 after $P = 1$		0	1	1	0

- The logic microoperations are seldom used in scientific computations, but they are very useful for bit manipulation of

binary data and for making logical decisions.

- Special symbols will be adopted for the logic microoperations OR, AND, and complement, to distinguish them from the corresponding symbols used to express Boolean functions.
- The symbol V will be used to denote an OR microoperation and the symbol ∧ to denote an AND microoperation. The complement microoperation is the same as the 1's complement and uses a bar on top of the symbol that denotes the register name.
- By using different symbols, it will be possible to differentiate

between a logic microoperation and a control (or Boolean) function.

Logic Microoperations

Boolean Function	Microoperation	Name
$F = 0$	$F \leftarrow 0$	Clear
$F = x\ y$	$F \leftarrow A \wedge B$	AND
$F =$	$F \leftarrow A \wedge$	

Boolean Function	Microoperation	Name
$F = (x + y)'$	$F \leftarrow \overline{A} \vee B$	NOR
$F = (x \oplus y)'$	$F \leftarrow \overline{A} \oplus B$	Exclusive-NOR

$x\ y$ '	B	
$F = x$	$F \leftarrow A$	Transfer A
$F = x\ 'y$	$F \leftarrow \bar{A} \wedge B$	
$F = y$	$F \leftarrow B$	Transfer B
$F = x\ y$	$F \leftarrow A \oplus B$	Exclusive-OR

$F = y$ '	$F \leftarrow B$	Complement B
$F = x + y$ '	$F \leftarrow A \vee B$	
$F = x$ '	$F \leftarrow \bar{A}$	Complement A
$F = x$ ' $+ y$	$F \leftarrow \bar{A} \vee B$	
$F = (x\ y)$'	$F \leftarrow \overline{A \wedge B}$	NAND

$F = x + y$	$F \leftarrow A \vee B$	OR

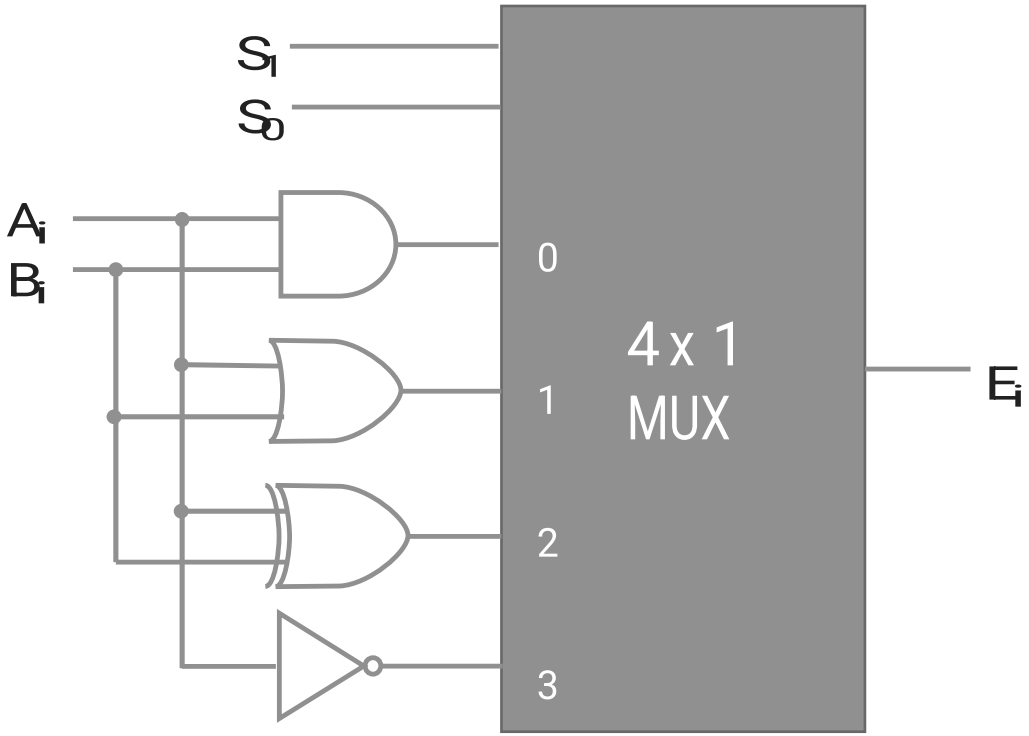
$F = 1$	$F \leftarrow all\ 1$s	Set to all 1's

- 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.

Hardware Implementation of Logic Circuit



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

- Logic microoperations are very useful for manipulating individual bits or a portion of a word stored in a register.
- They can be used to change bit values, delete a group of bits, or insert new bit values into a register.
- The following examples show how the bits of one register (designated by A) are manipulated by logic microoperations as a function of the bits of another register (designated by B).
- In a typical application, register A is a processor register and the bits of register B constitute a logic operand extracted from

memory and placed in register B .

Applications of Logic Microoperations

1.Selective Set Operation

- The selective-set operation sets to 1 the bits in register A where there are corresponding 1's in register B.
- It does not affect bit positions that have 0's in B.
- The OR microoperation can be used to selectively set bits of a register.

1 0 1 0 A before 1 1 0

Applications of Logic Microoperations

0 B (logic operand)

---

1 1 1 0 A after

2.Selective Complement Operation

- The selective-complement operation complements bits in register A wherethere are corresponding 1's in register B.
- It does not affect bit positions that have 0's in B.
- The exclusive - OR microoperation can be used to selectively set bits of a register.

Applications of Logic Microoperations

1 0 1 0 A before 1 1 0

0 B (logic operand)

---

0 1 1 0 A after

3.Selective Clear Operation

- The selective-clear operation clears to 0 the bits in register A only where there are corresponding 1's in register B.
- It does not affect bit positions that have 0's in B.

Applications of Logic Microoperations

- The corresponding logic microoperation is A ← A ∧ B'.

1	0	1	0	A before	1	1	0
0				B (logic operand)			
0	0	1	0	A after			

4.Mask Operation

- The mask operation is similar to the selective-clear operation

Applications of Logic Microoperations

except that the bits of register A are cleared only where there are corresponding 0's in register B.

- The mask operation is an AND microoperation.

1	0	1	0	A before	1	1	0
0				B (logic operand)			

Applications of Logic Microoperations

1 0 0 0 A after

5.Insert Operation

- The insert operation inserts a new value into a group of bits.
- This is done by first masking and then ORing them with required value.
- The mask operation is an AND microoperation and the insert operation is an OR microoperation.

Mask Insert

Applications of Logic Microoperations

A	01101010	A	00000101 0
B	00001111	B	10010000
A	00001010	A	10011010

6.Clear Operation

- The clear operation compares the words in register A and register B and produces an all 0's result if the two numbers are equal.

Applications of Logic Microoperations

- This operation is achieved by an exclusive-OR microoperation.

$$
\begin{array}{llllllll}
1 & 0 & 1 & 0 & A\,1 & 0 & B\,0 & 0 \\
1 & 0 & & & A \leftarrow A \oplus B & & & \\
0 & & & & & & & \\
\hline
 & 0 & & & & & &
\end{array}
$$

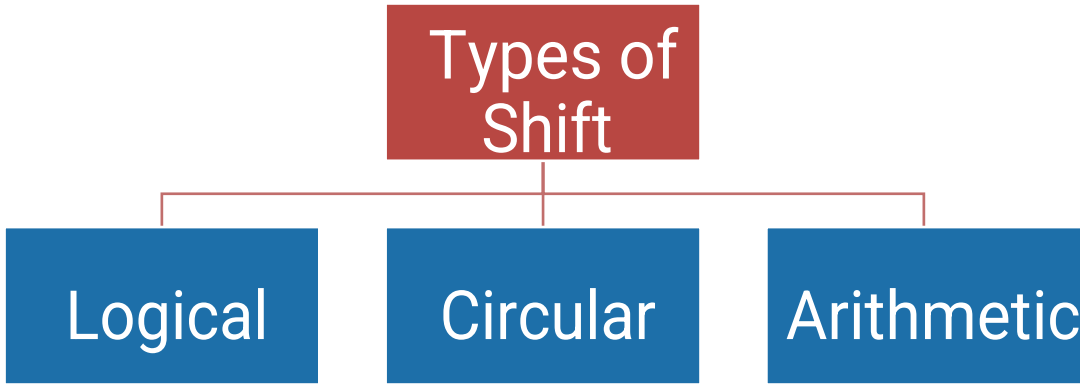
Shift Micro-operations

Section - 7

Shift Microoperations

- Shift microoperations are used for serial transfer of data.
- Used in conjunction with arithmetic, logic and other data processing operations.
- The content of the register can be shifted to the left or the right.

- The first flip-flop receives its binary information from the serial input.
- The information transferred through the serial input determines the type of shift.

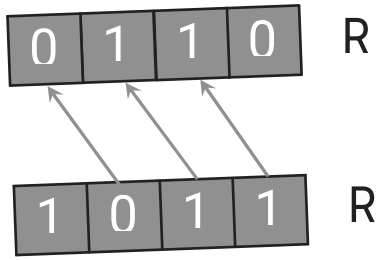


1.Logical Shift

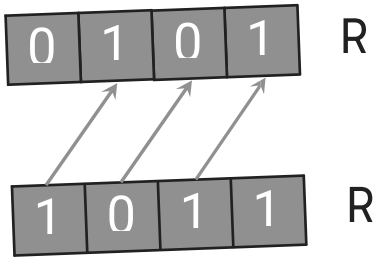
Types of Shift

- A logical shift is one that transfers 0 through the serial input.

shl - logical shift left



shr - logical shift right



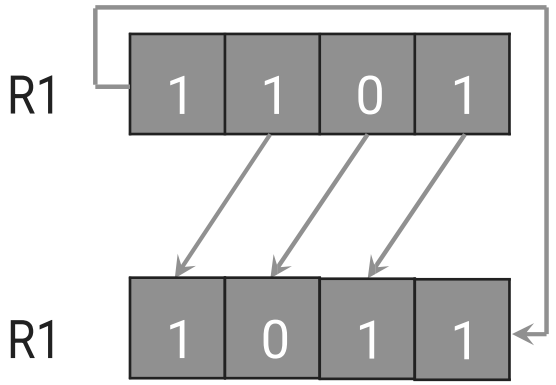
Types of Shift

2.Circular Shift

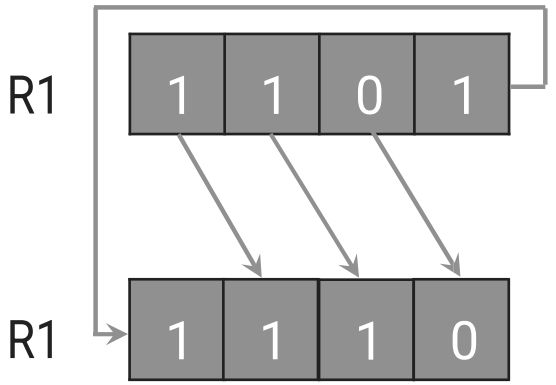
- A circular shift (also known as a rotate operation) circulates the bits of the register around the two ends without loss of information.
- This is accomplished by connecting the serial output of the shift register to its serial input.

Types of Shift

cil - circular shift left



cir - circular shift right



3.Arithmetic Shift

 An arithmetic shift is a micro-operation that shifts a signed binary number to the left or right.

Types of Shift

- An arithmetic shift-left multiplies a signed binary number by 2.
- An arithmetic shift-right divides the number by 2.

ashl - arithmetic shift leftashr - arithmetic shift right

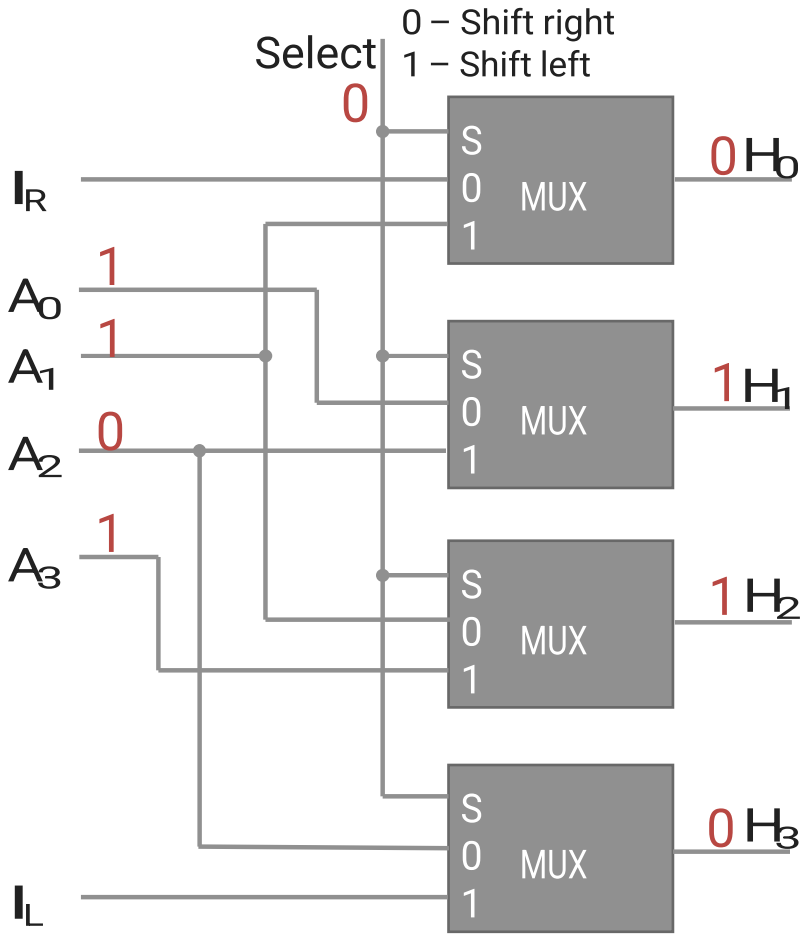
The arithmetic shift-left inserts a 0 into R_0, and shifts all other bits to the left. The initial bit of R_{n-1} is lost and replaced by the bit from R_{n-2}. A sign reversal occurs if the bit in R_{n-1} changes in value after the shift. This happens if the multiplication by 2 causes an overflow. An overflow occurs after an arithmetic shift left if initially, before the shift, R_{n-1} is not equal to R_{n-2}. An overflow flip-flop V_s can be used to detect an arithmetic shift-left overflow.

$$V_s = R_{n-1} \oplus R_{n-2}$$

If $V_s = 0$, there is no overflow, but if $V_s = 1$, there is an overflow and a sign reversal after the shift. V_s must be transferred into the overflow flip-flop with the same clock pulse that shifts the register.

significant bit. The arithmetic shift-right leaves the sign bit unchanged and shifts the number (including the sign bit) to the right. Thus R_{n-1} remains the same, R_{n-2} receives the bit from R_{n-1}, and so on for the other bits in the register. The bit in R_0 is lost.

4 - bit Combinational Circuit Shifter



S	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

4 - bit Combinational Circuit Shifter

- The 4-bit shifter has four data inputs, A_0 through A_3 and four data outputs, H_0 through H_3.
- There are two serial inputs, one for shift left (I_L) and the other for shift right (I_L).
- When the selection input S = 0, the input data are shifted right (down in the diagram).
- When S = 1, the input data are shifted left (up in the diagram).
- The two serial inputs can be controlled by another multiplexer to provide the three possible types of shifts.

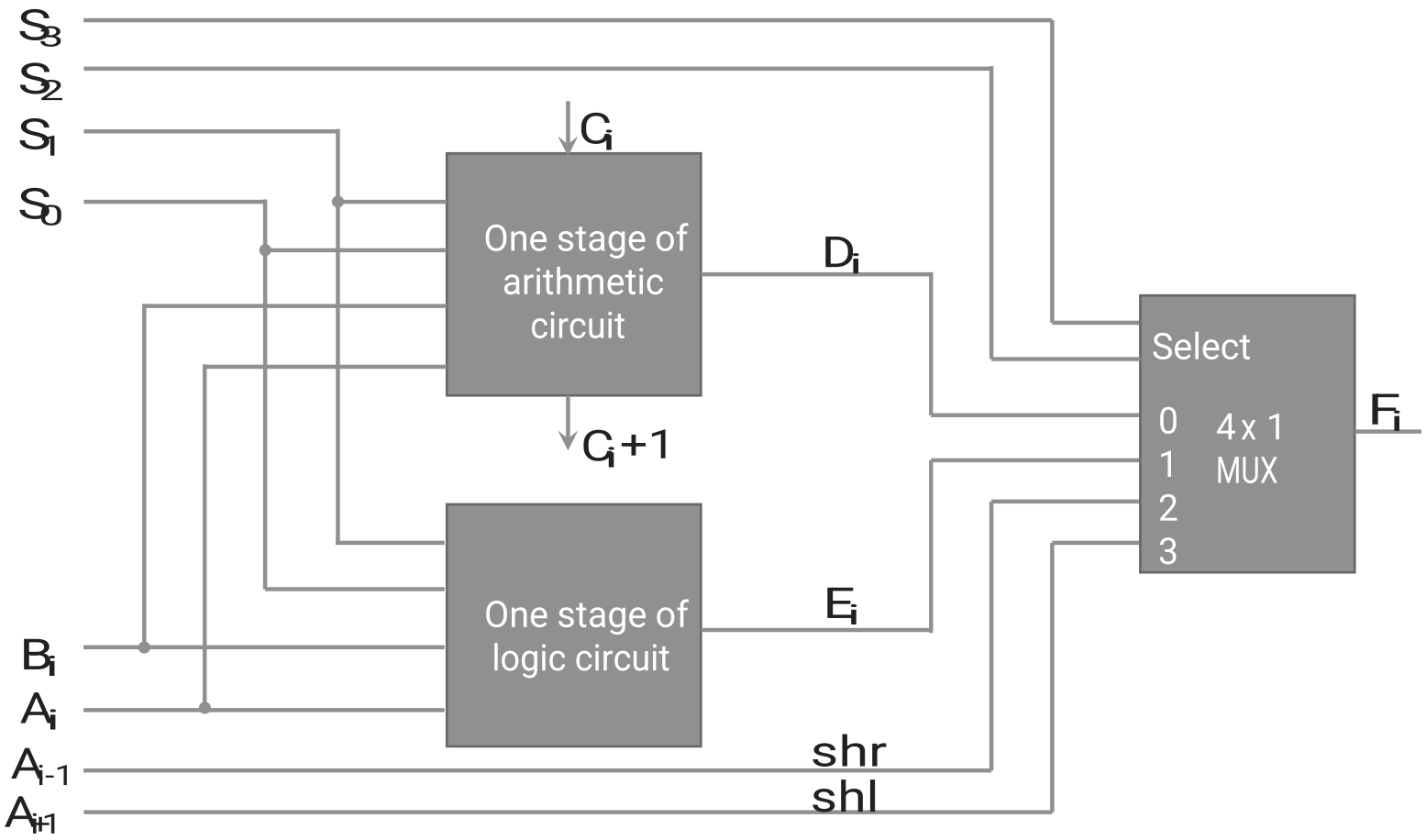
Arithmetic logical shift unit

Section - 8 Instead of having individual registers performing the micro operations directly, computer systems employ a number of storage registers connected to a common operational unit called an arithmetic logic unit, abbreviated ALU.

- To perform a microoperation, the contents of specified registers are placed in the inputs of the common ALU.
- The ALU performs an operation and the result of the operation is then transferred to a destination register.

- The arithmetic, logic, and shift circuits introduced in previous sections can be combined into one ALU with common selection variables.

4 - bit Arithmetic Logic Shift Unit



4 - bit Arithmetic Logic Shift Unit

S_3	S_2	S_1	S_0	C_{in}	Operation	Function
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

S_3	S_2	S_1	S_0	C_{in}	Operatio
0	0	1	1	1	F = A
0	1	0	0	x	F = A B
0	1	0	1	x	F = A

0	0	0	1	1	F = A +B + 1	Add with carry
0	0	1	0	0	F = A +B’	Subtract with borrow
0	0	1	0	1	F = A +B’ + 1	Subtraction
0	0	1	1	0	F = A – 1	Decrement

					B
0	1	1	0	x	F = A ⊕B
0	1	1	1	x	F = A’
1	0	x	x	x	F = shrA
1	1	x	x	x	F = shlA