Unit 2
Microoperations

Combinational and sequential circuits can be used to create simple digital systems. These are the low-level building blocks of a digital computer. The operations on the data in registers are called microoperations. Examples of micro-operations are

- Shift
- Load
- Clear
- Increment

Alternatively we can say that an elementary operation performed during one clock pulse on the information stored in one or more registers is called micro-operation. The result of the operation may replace the previous binary information of the resister or may be transferred to another resister. Register transfer language can be used to describe the (sequence of) micro-operations.

Microoperation types

The microoperations most often encountered in digital computers are classified into 4 categories:

1. Register transfer microoperations
2. Arithmetic microoperations
3. Logic microoperations
4. Shift microoperations

1. Resister 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 ←R1" denotes a transfer of the content of the R1 into resister R2.

Control Function

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 ← R1 i.e. if (P = 1) then (R2← 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 ← R5, MAR ← IR

csitnepal

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

Summary of typical arithmetic microoperations

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

Binary Adder

To implement the add microoperation with hardware, we need the resisters that hold the data and the digital component that performs the arithmetic addition. The digital circuit that generates the arithmetic sum of two binary numbers of any lengths is called **Binary adder**. The binary adder is constructed with the full-adder circuit connected in cascade, with the output carry from one full-adder connected to the input carry of the next full-adder.

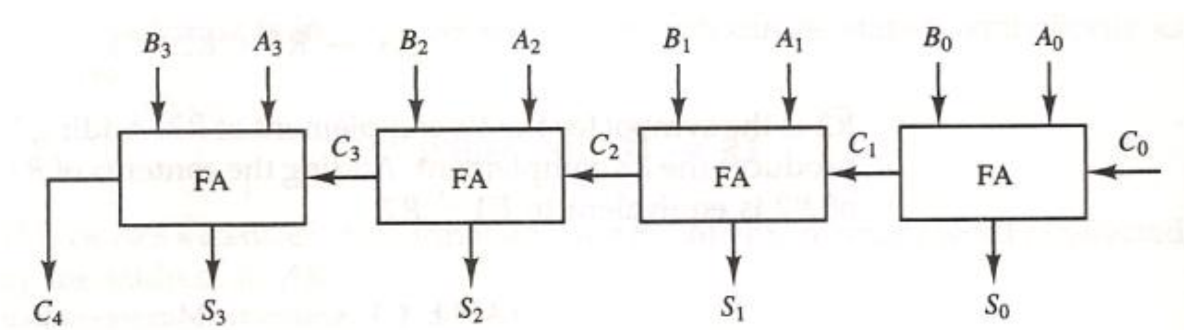


Fig.: 4-bit binary adder

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-full-adder. Inputs A and B come from two registers R1 and R2.

csitnepal

Binary Subtractor

The subtraction A – B can be done by taking the 2's complement of B and adding to A. It means if we use the inverters to make 1's complement of B (connecting each B_i to an inverter) and then add 1 to the least significant bit (by setting carry C_0 to 1) of binary adder, then we can make a binary subtractor.

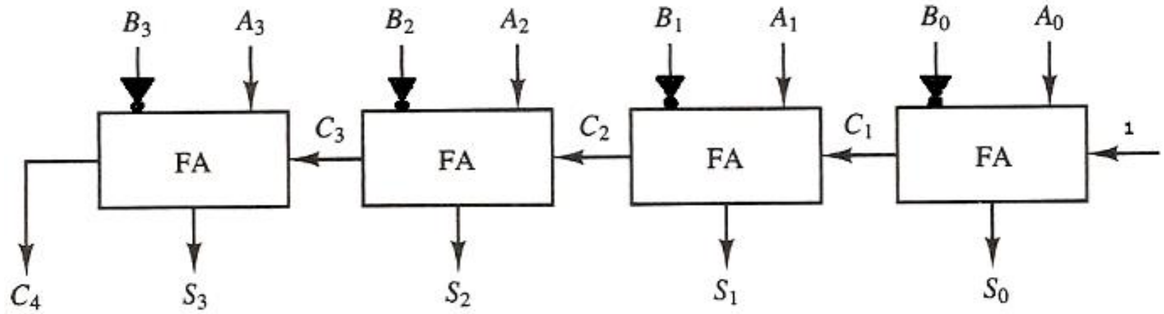


fig.: 4-bit binary subtractor

Binary Adder-Subtractor

__Question__: __*How binary adder and subtractor can be accommodated into a single circuit? explain.*__

The addition and subtraction operations can be combined into one common circuit by including an exclusive-OR gate with each full-adder.

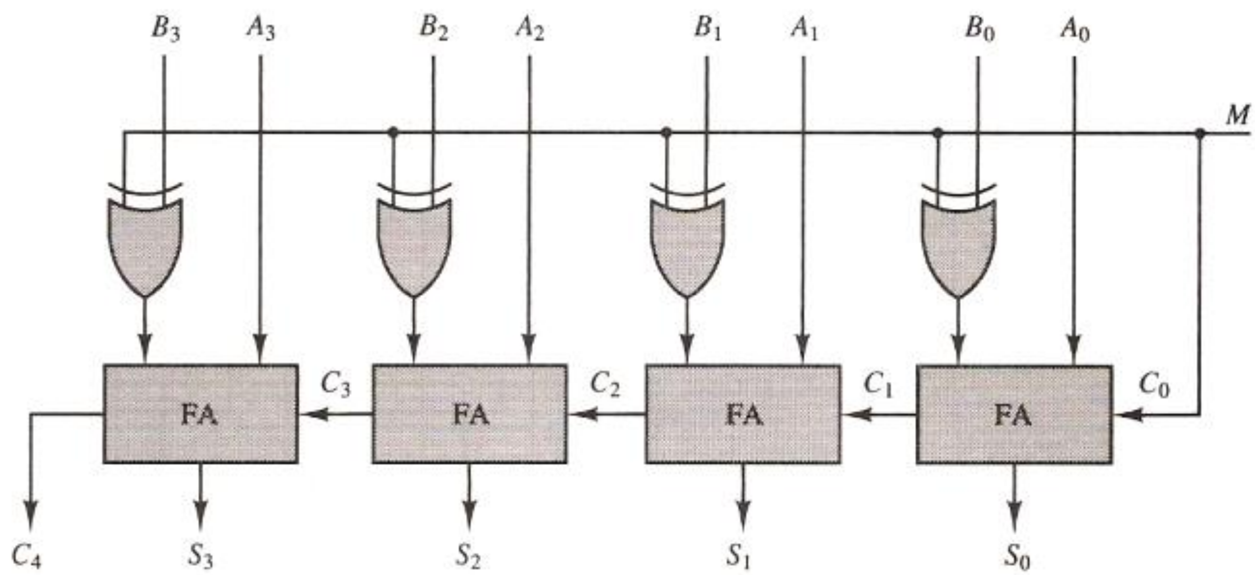


Fig.: 4-bit adder-subtractor

The mode input M controls the operation the operation. When M=0, the circuit is an adder and when M=1 the circuit becomes a subtractor. Each exclusive-OR gate receives input M and one of the inputs of B.

- When M=0: $B \oplus M = B \oplus 0 = B$, i.e. full-adders receive the values of B, input carry is B and circuit performs A+B.
- When M=1: $B \oplus M = B \oplus 1 = B'$ and $C_0 = 1$, i.e. B inputs are all complemented and 1 is added through the input carry. The circuit performs A + (2's complement of B).

csitnepal

Binary Incrementer

The increment microoperation adds one to a number in a register. For example, if a 4-bit register has a binary value 0110, it will go to 0111 after it is incremented. Increment microoperation can be done with a combinational circuit (half-adders connected in cascade) independent of a particular register.

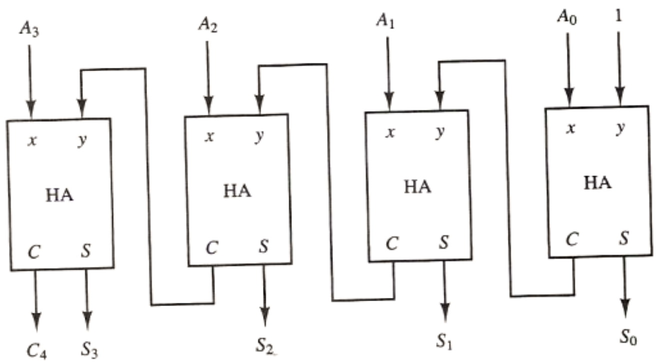


Fig.: 4-bit binary Incrementer

Arithmetic Circuit

The arithmetic microoperations can be implemented in one composite arithmetic circuit. By controlling the data inputs to the adder (basic component of an arithmetic circuit), it is possible to obtain different types of arithmetic operations.

In the circuit below contains:

- 4 full-adders
- 4 multiplexers (controlled by selection inputs S0 and S1)
- two 4-bit inputs A and B and a 4-bit output D
- Input carry c_{in} goes to the carry input of the full-adder.

Output of the binary adder is calculated from the arithmetic sum: $D = A + Y + c_{in}$.

By controlling the value of Y with the two selection inputs S1 & S0 and making c_{in}= 0 or 1, it is possible to generate the 8 arithmetic microoperations listed in the table below:

Select S_1	S_0	C_{in}	Input Y	Output $D = A + Y + C_{in}$	Microoperation
0	0	0	B	$D = A + B$	Add
0	0	1	B	$D = A + B + 1$	Add with carry
0	1	0	$\overline{B}$	$D = A + \overline{B}$	Subtract with borrow
0	1	1	$\overline{B}$	$D = A + \overline{B} + 1$	Subtract
1	0	0	0	$D = A$	Transfer A
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

csitnepal

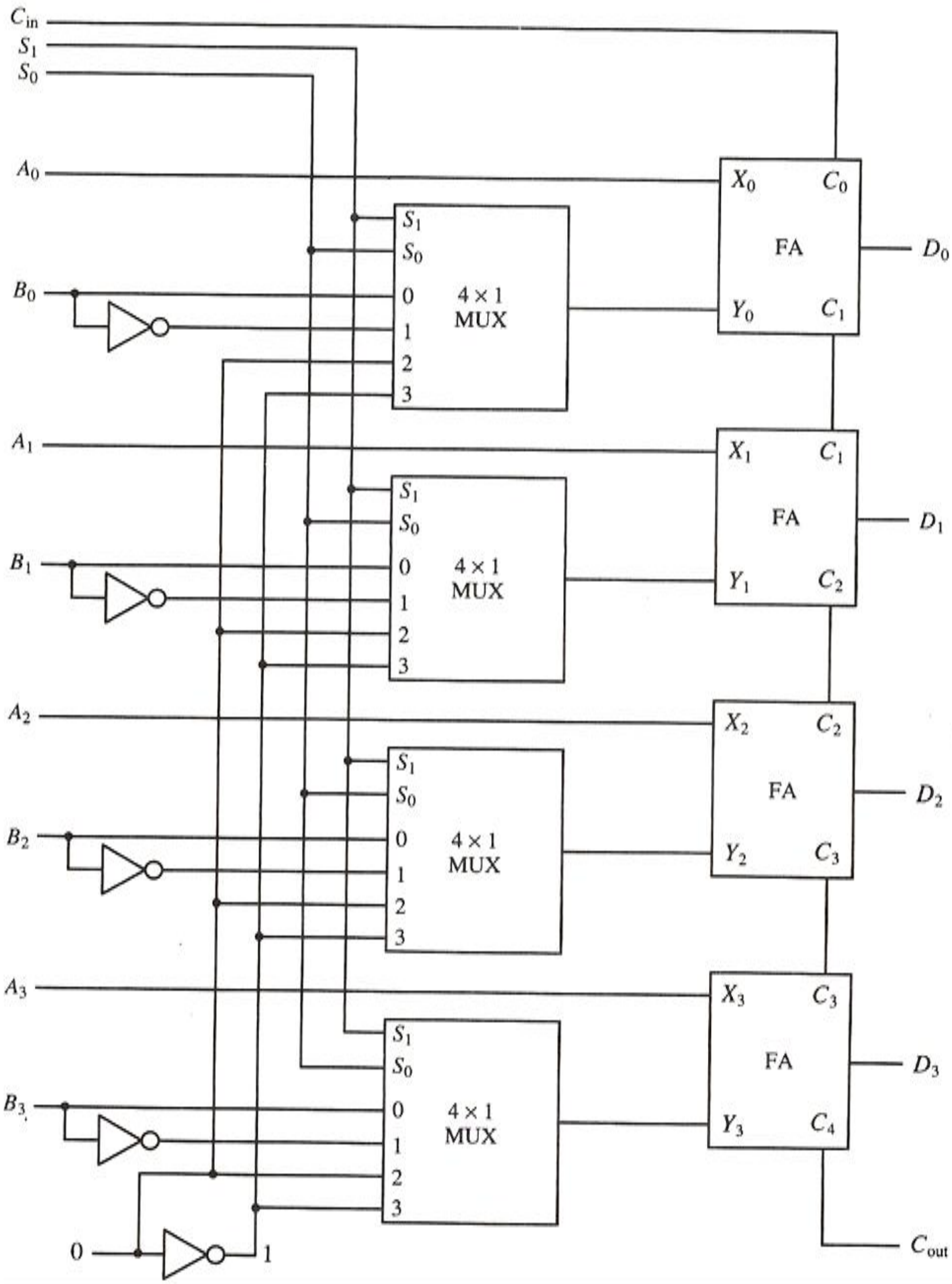


Fig: 4-bit arithmetic circuit

csitnepal

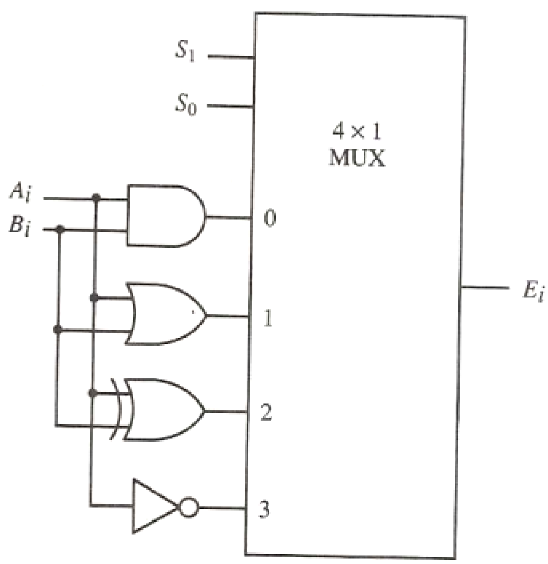
3. Logic microoperations

Question: *What do you mean by Logic microoperations? Explain with its applications.*
Question: *How Logic microoperations can be implemented with hardware?*
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, in principle, 16 different logic functions that can be defined over two binary input variables. However, most systems only implement four of these

- AND (^), OR (˅), XOR (⊕), Complement/NOT

The others can be created from combination of these four functions.

Hardware implementation
Hardware implementation of logic microoperations requires that logic gates be inserted be each bit or pair of bits in the resisters to perform the required logic operation.



(a) Logic diagram

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	$E = \overline{A}$	Complement

(b) Function table

Applications of Logic Microoperations

Logic microoperations can be used to manipulate individual bits or a portion of a word in a register. Consider the data in a register A. Bits of register B will be used to modify the contents of A.

– Selective-set	A ← A + B
– Selective-complement	A ← A ⊕ B
– Selective-clear	A ← A • B’
– Mask (Delete)	A ← A • B
– Clear	A ← A ⊕ B
– Insert	A ← (A • B) + C
– Compare	A ← A ⊕ B

csitnepal

Selective-set

In a selective set operation, the bit pattern in B is used to *set* certain bits in A.

$$\begin{array}{lll} 1\,1\,0\,0 & A_t & \\ 1\,0\,1\,0 & B & \\ \hline 1\,1\,1\,0 & A_{t+1} & (A \leftarrow A + B) \end{array}$$

Bits in resister A are set to 1 when there are corresponding 1's in resister B. It does not affect the bit positions that have 0's in B.

Selective-complement

In a selective complement operation, the bit pattern in B is used to *complement* certain bits in A.

$$\begin{array}{lll} 1\,1\,0\,0 & A_t & \\ 1\,0\,1\,0 & B & \\ \hline 0\,1\,1\,0 & A_{t+1} & (A \leftarrow A \oplus B) \end{array}$$

If a bit in B is 1, corresponding position in A get complemented from its original value, otherwise it is unchanged.

Selective-clear

In a selective clear operation, the bit pattern in B is used to *clear* certain bits in A.

$$\begin{array}{lll} 1\,1\,0\,0 & A_t & \\ 1\,0\,1\,0 & B & \\ \hline 0\,1\,0\,0 & A_{t+1} & (A \leftarrow A \bullet B') \end{array}$$

If a bit in B is 1, corresponding position in A is set to 0, otherwise it is unchanged.

Mask Operation

In a mask operation, the bit pattern in B is used to *clear* certain bits in A.

$$\begin{array}{lll} 1\,1\,0\,0 & A_t & \\ 1\,0\,1\,0 & B & \\ \hline 1\,0\,0\,0 & A_{t+1} & (A \leftarrow A \bullet B) \end{array}$$

If a bit in B is 0, corresponding position in A is set to 0, otherwise it is unchanged. This is achieved logically ANDing the corresponding bits of A and B.

Clear Operation

In clear operation, if the bits in the same position in A and B same, that bit in A is cleared (putting 0 there), otherwise same bit in A is set(putting 1 there). This operation is achieved by exclusive-OR microoperation.

$$\begin{array}{lll} 1\,1\,0\,0 & A_t & \\ 1\,0\,1\,0 & B & \\ \hline 0\,1\,1\,0 & A_{t+1} & (A \leftarrow A \oplus B) \end{array}$$

Insert Operation

An insert operation is used to introduce a specific bit pattern into A register, leaving the other bit positions unchanged.

csitnepal

This is done as

- A **mask (ANDing)** operation to clear the desired bit positions, followed by
- An **OR** operation to introduce the new bits into the desired positions
- Example
  - » Suppose you want to introduce 1010 into the low order four bits of A:

1101 1000 1011 0001	A (Original)
1101 1000 1011 1010	A (Desired)

1101 1000 1011 0001	A (Original)
1111 1111 1111 0000	B (Mask)

1101 1000 1011 0000	A (Intermediate)
0000 0000 0000 1010	Added bits

1101 1000 1011 1010	A (Desired)

4. Shift microoperations

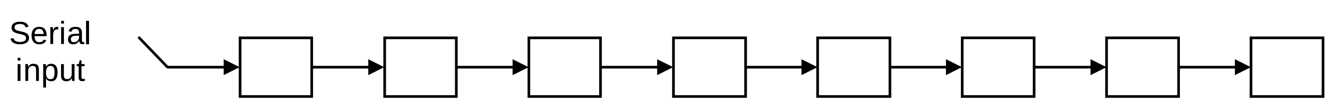
<u>Question</u>: <u>*What do you mean by shift microoperations*? Explain its types.</u>

<u>Question</u>: <u>*Is there a possibility of Overflow during arithmetic shift? If yes, how it can be detected?*</u>

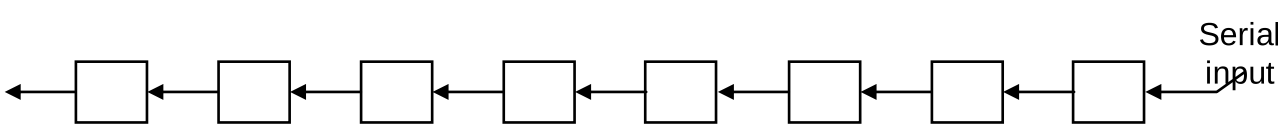
Shift microoperations are used for serial transfer of data. They are also used in conjunction with arithmetic, logic and other data processing operations. The contents of a resister can be shifted left or right. There are three types of shifts

1. Logical shift
2. Circular shift
3. Arithmetic shift

Right Shift Operation



Left shift operation



1. Logical shift

A logical shift is one that transfers 0 through the serial input. In a Register Transfer Language, the following notation is used

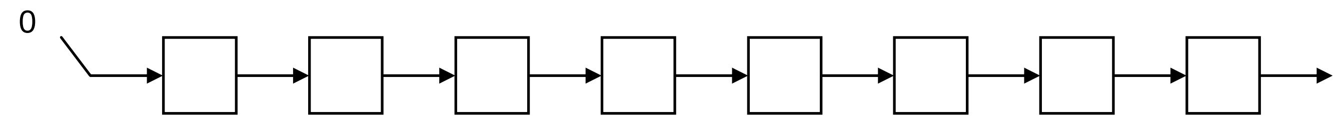
- *shl* for a logical shift left
- *shr* for a logical shift right

Examples:

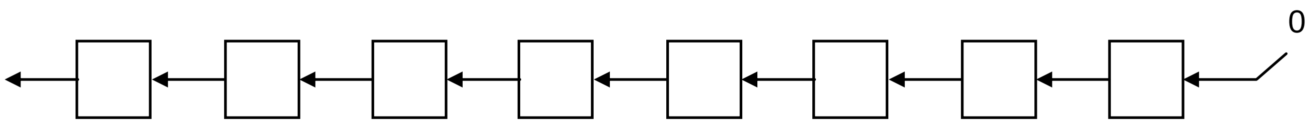
R2 ← *shr* R2

csitnepal

R3 ← *shl* R3



Logical right shift (shr)

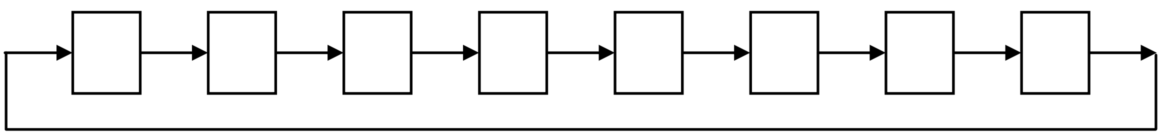


Logical left shift (shl)

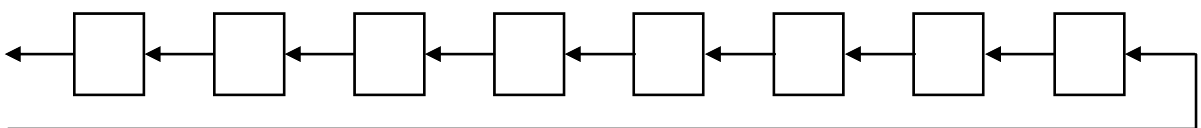
2. Circular Shift (rotate operation)

Circular-shift circulates the bits of the resister around the two ends without the loss of information.

Right circular shift operation



Left circular shift operation:



In a RTL, the following notation is used

- *cil* for a circular shift left
- *cir* for a circular shift right
- Examples:

  R2 ← *cir* R2

  R3 ← *cil* R3

3. Arithmetic shift

An arithmetic shift is meant for signed binary numbers (integer). An arithmetic left shift multiplies a signed number by 2 and an arithmetic right shift divides a signed number by 2. Arithmetic shifts must leave the sign bit unchanged because the sign of the number remains the same when it is multiplied or divided by 2. The left most bit in a resister holds a sign bit and remaining hold the number. Negative numbers are in 2's complement form.

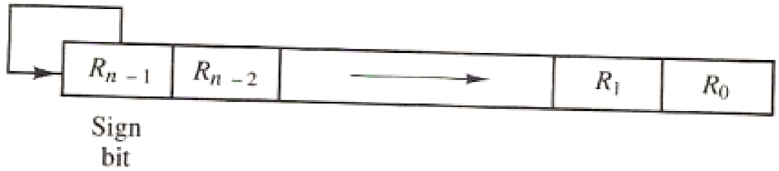
In a Resister Transfer Language, the following notation is used

- *ashl* for an arithmetic shift left
- *ashr* for an arithmetic shift right
- Examples:
  - » R2 ← *ashr* R2
  - » R3 ← *ashl* R3

csitnepal

Arithmetic shift-right

Arithmetic shift-right leaves the sign bit unchanged and shifts the number (including a sign bit) to the right. Thus R_{n-1} remains same; R_{n-2} receives input from R_{n-1} and so on.



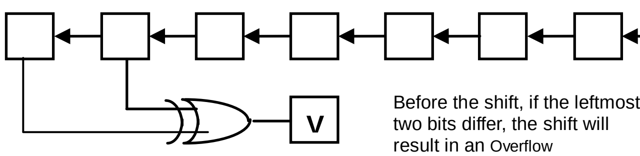
Arithmetic shift-left

Arithmetic shift-left inserts a 0 into R_0 and shifts all other bits to left. Initial bit of R_{n-1} is lost and replaced by the bit from R_{n-2}.

Overflow case during arithmetic shift-left:

If a bit in R_{n-1} changes in value after the shift, sign reversal occurs in the result. This happens if the multiplication by 2 causes an overflow.

Thus, left arithmetic shift operation must be checked for the overflow: an overflow occurs after an arithmetic shift-left if before shift $R_{n-1} \neq R_{n-2}$.



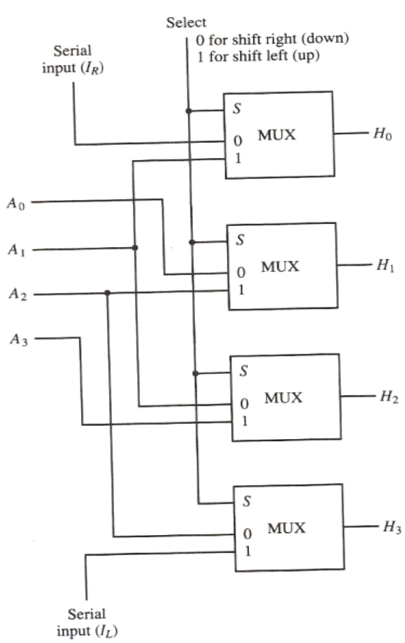
An overflow flip-flop V can be used to detect an arithmetic shift-left overflow.

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

If V = 0, there is no overflow but if V = 1, overflow is detected.

Hardware implementation of shift microoperations

A combinational circuit shifter can be constructed with multiplexers as shown below:



Function table

Select	Output			
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

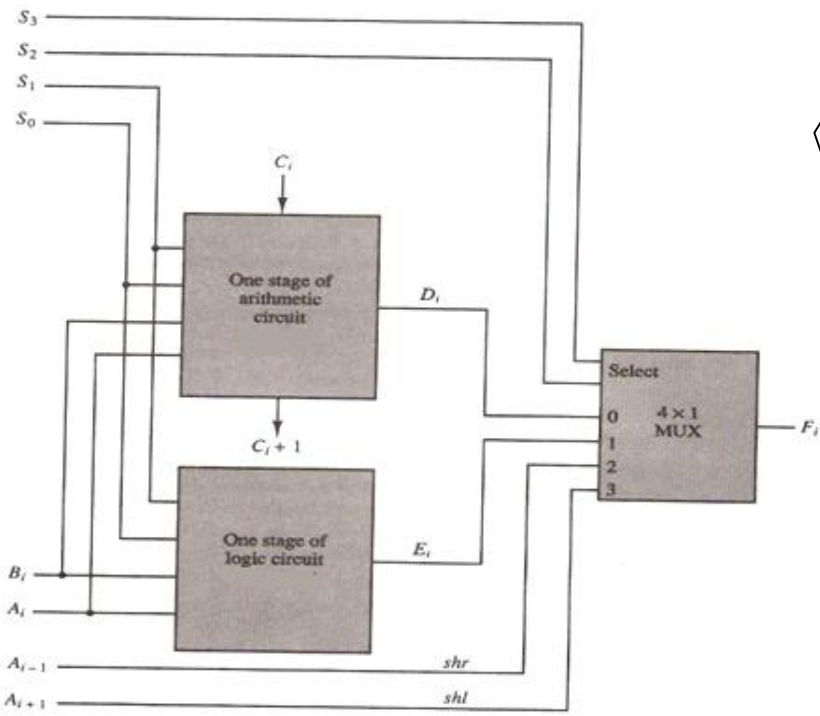
- It has 4 data inputs A_0 through A_3 and 4 data outputs H_0 through H_3.
- There are two serial inputs, one for shift-left (I_L) and other for shift-right (I_R).
- When S = 0: input data are shifted right (down in fig).
- When S = 1: input data are shifted left (up in fig).

Fig: 4-bit combinational circuit shifter

csitnepal

Arithmetic Logic Shift Unit

This is a common operational unit called arithmetic logic unit (ALU). To perform a microoperation, the contents of specified registers are placed in the inputs of the common ALU. The ALU performs the operation and transfer result to destination resister.



- A particular microoperation is selected with inputs s1 and s0.
- A 4x1 MUX at the output chooses between an arithmetic output in D_i and logic output E_i.
- Other two inputs to the MUX receive inputs A_{i-1} for right-shift operation and A_{i+1} for left-shift operation.
- The diagram shows just one typical stage. The circuit must be repeated n times for an n-bit ALU.

Fig: One stage of arithmetic logic shift unit

This circuit provides 8 arithmetic operations, 4 logic operations and 2 shift operations. Each operation is selected with five variables s_3, s2, s_1, s_0 and c_{in}. The input carry c_{in} is used for arithmetic operations only. Table below lists the 14 operations of the ALU.

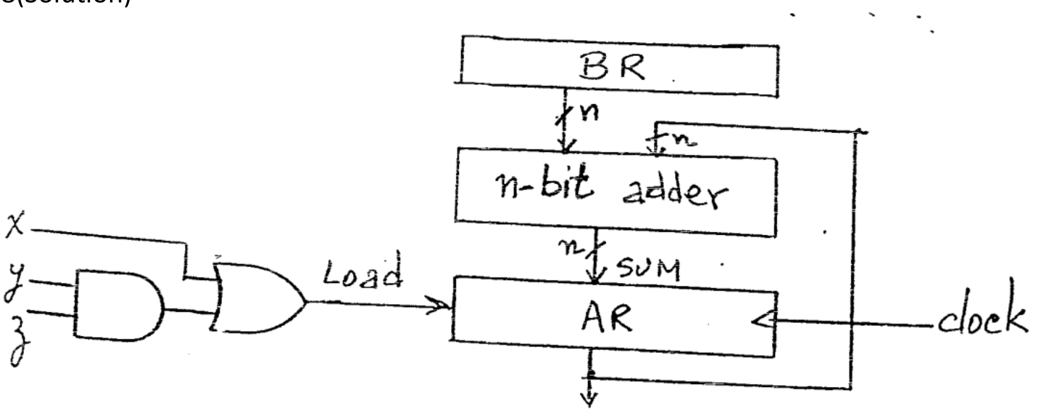
Operation select						
S_3	S_2	S_1	S_0	C_{in}	Operation	Function
0	0	0	0	0	$F = A$	Transfer A
0	0	0	0	1	$F = A + 1$	Increment A
0	0	0	1	0	$F = A + B$	Addition
0	0	0	1	1	$F = A + B + 1$	Add with carry
0	0	1	0	0	$F = A + \overline{B}$	Subtract with borrow
0	0	1	0	1	$F = A + \overline{B} + 1$	Subtraction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

Table: Function table for Arithmetic logic shift unit

csitnepal

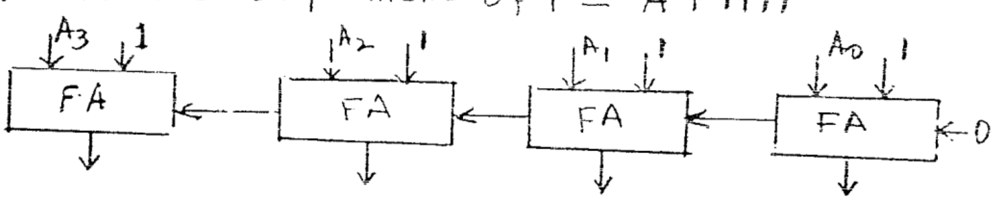
EXERCISES: Textbook chapter 4 ➔ 4.8, 4.13, 4.17, 4.18, 4.19, 4.21

4.8(Solution)

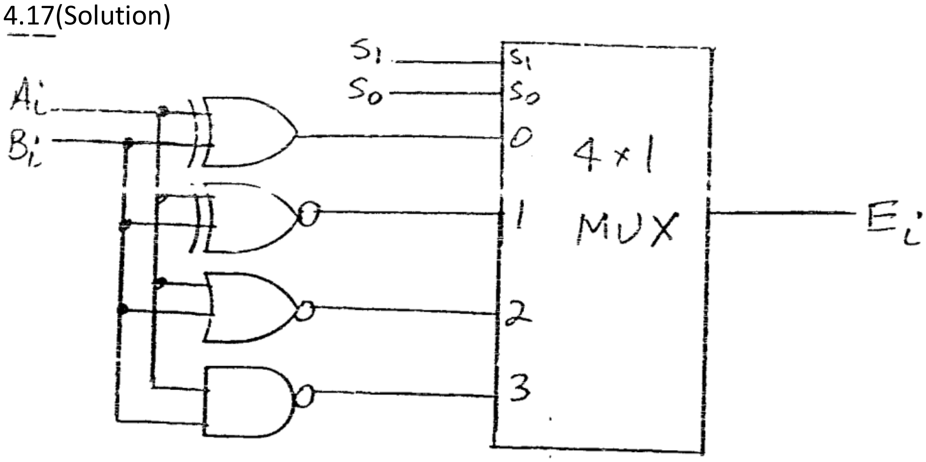


4.13(Solution)

$A-1 = A + 2\text{'s complement of } 1 = A + 1111$



4.17(Solution)



4.18(Solution)

(a)

	A =	11011001
	B =	10110100 ⊕
A←A⊕B		01101101

A =	11011001	
B =	11111101	(OR)
	11111101	A←A∨B

4.19(do it yourself)
4.21(do it too)

csitnepal