Date:
Page: 1

"Register Transfer and Microoperations"

Register Transfer Language :- Digital systems are composed of modules that are constructed from digital components, such as registers, decoders, arithmetic elements and control logic. The modules are interconnected with common data & control path to form a digital computer system.

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 micro-operation performed on the binary information stored in the register.
3. The control that initiates the sequence of microoperation.

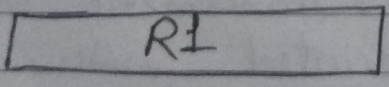
It is more convenient to adopt a suitable symbolic language to describe the sequence of transfers between registers and the various arithmetic and logic microoperation associated with the transfers rather than using explaining every operation in words.

The operations performed on the data stored in registers are called Micro-operations.

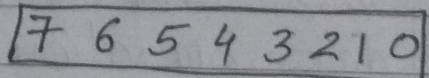
The Register Transfer Language is the symbolic representation of notations used to specify the sequence of micro-operations. or a convenient tool for describing the internal organization of digital computers. or The language, which is basically used to express the transfer of data among the registers, is called Register Transfer Language (RTL)

Register Transfer:- Normally computer registers are designated by capital letters, sometimes followed by numerals to denote the register function. For example, the address memory register is designated by (MAR), in which it is function is to hold an address for the memory unit. other examples are Program Counter (PC), Instruction Register (IR), and Processor Register (R1)

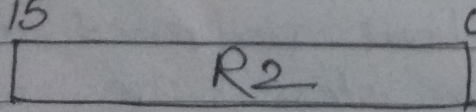
The n-bit of the register are numbered in sequence from 0 in the rightmost position and increasing the numbers towards the left.



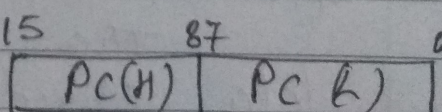
(a) Register R



(b) Showing individual bits



(c) Numbering of bits



(d) Divided into two parts (two byte registers)

Block Diagram of Register

1. fig (a) shows the common way to represent a registers (a rectangular box with the name of the register inside).
2. fig (b) shows how the individual register bits are distinguished.
3. The numbering of bits in a 16-bit register can be marked on top of the box as shown in (c)
4. A 16-bit register is partitioned into two parts in (d). bits 0 to 7 are assigned the symbol L (low byte), while bits 8 to 15 are assigned the symbol H (high byte). and the name of the register is PC (Program Counter). The symbol PC(0-7) or PC(L) refers to the low order byte and PC(8-15) or PC(H) to the high orders byte.

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

$$R2 \leftarrow R1$$

The statement 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. By definition, the content of the source register R1 does not change after the transfer.

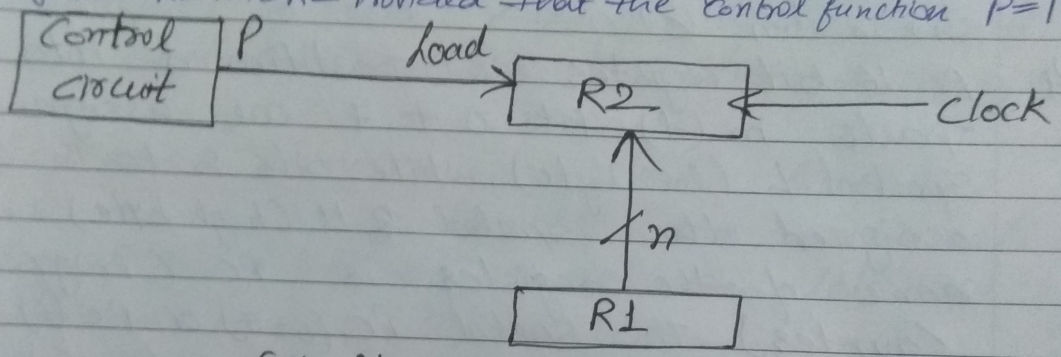
If there is a predetermined control condition like

If (P=1) then (R2 ← R1)

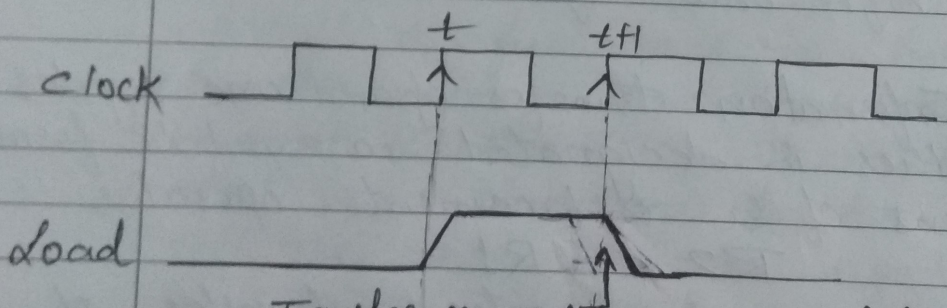
then we can write the statement as

P: R2 ← R1

where P is control signal usually a control function which is Boolean variable that is equal to 1 or 0. This statement denotes a transfer of the content of register R1 into R2 provided that the control function P=1.



(a) Block diagram



(b) Timing diagram

Transfer from R1 to R2 when P=1

The n outputs of Register R1 are connected to the n inputs of register R2. The letter n will be used to indicate any number of bits for the register.

It is assumed that all transfer occur during a clock edge transition

Even though the control condition such as P becomes active just after time t, the actual transfer does not occur until the register is triggered by the next positive transition of the clock at time t+1.

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

The statement

T: R2 ← R1, R1 ← R2

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

some basic symbols for register transfer are

symbol	Description	Examples
• Letters & Numerals	Denotes a Register	MAR, R2
• Parentheses ( )	Denotes a part of a Register by specifying the range of bits or by giving a symbol name to a portion of a register.	R2(0-7), R2(L) or R2(8-15), R2(H)
• Comma ,	separates two Micro-operation that are executed at the same time	R2 ← R1, R1 ← R2
• Arrow ←	Denotes transfer of information & the transfer direction	R2 ← R1
• Colon :	Denotes transfer of information under a pre-determined control condition.	P: R2 ← R1

Date
Page 6

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.

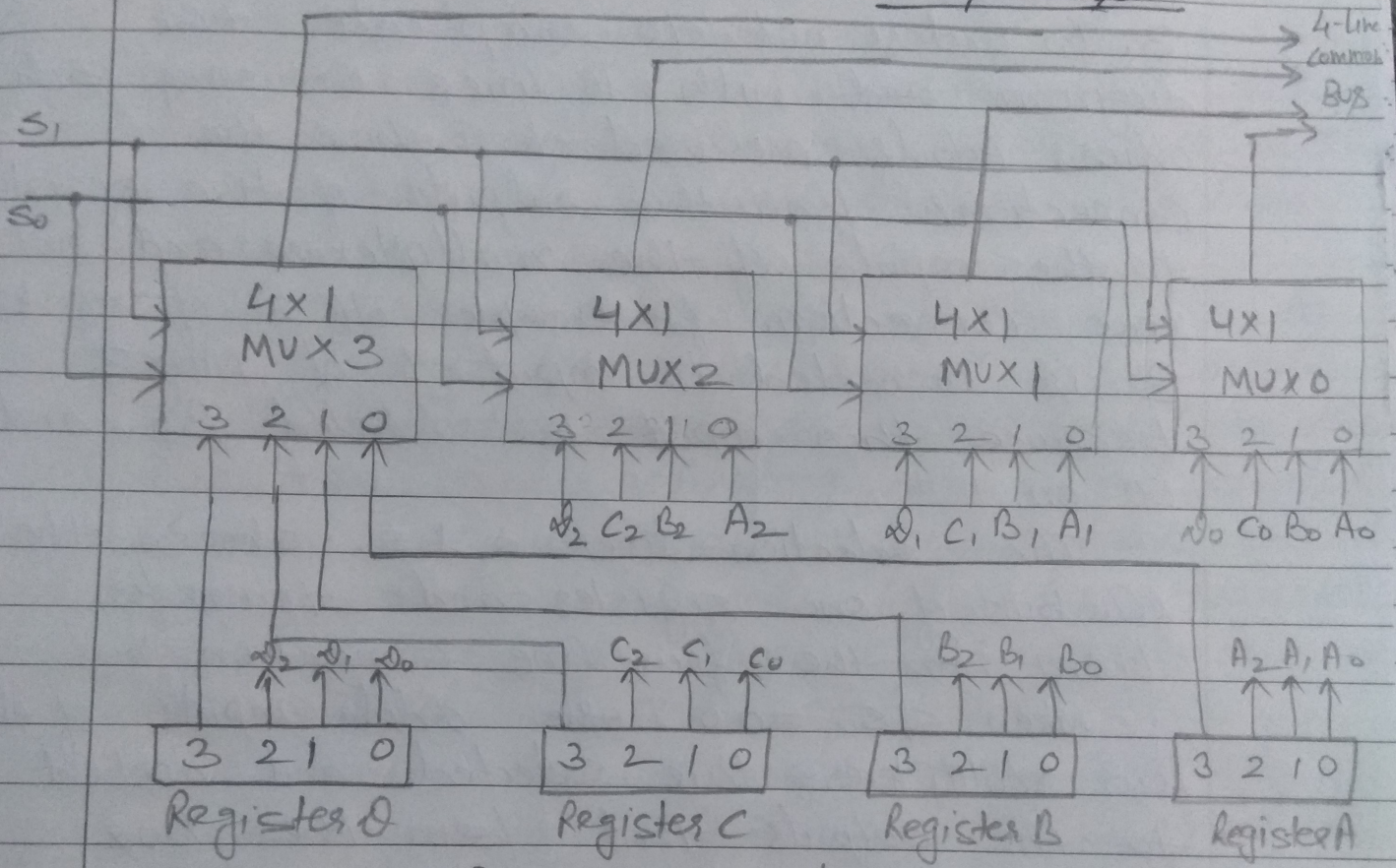
A more efficient scheme for transferring information between registers in a multiple register configuration is a "Common bus system."

A Bus is a structure which consists of a set of common lines, one for each bit of a register, through which binary information is transferred one at a time (between two registers, Memory and a register, Register and a Memory etc). Control signals determine which register is selected by the bus during each particular register transfer.

A common bus system can be constructed

using either Multiplexers or a Three-state Bus Buffers.

1. Constructing A common Bus System using Multiplexers :-



Bus system for four Registers.

A common bus system can be Constructed using multiplexers, in which these multiplexers select the source registers whose information are placed on the bus. The above figure shows a bus system for four registers. The bus consists of four 4-1 Mux, each having four data inputs 0-3 & two selection inputs, S_1 & S_0.

Bits in the same significant position

in each registers are connected to the data inputs of one multiplexes to form one line of the bus.

Each registers has four ~~data inputs, 0 through 3~~ bits, numbered 0 through 3. In orders not to complicate the diagram ~~and~~ with 16 lines crossing each others, labels are used to show the connections from the outputs of the registers to the inputs of the multiplexess and not line connection. for example, o/p 2 of register B is connected to input 1 of MUX 3 because this input is labeled B_3, and so on.

The selection lines S_1 & S_0 choose the four bits of one registers and transfer them into the four line connection bus.

When $S_1 S_0 = 0\,0$, the 0 data inputs of all four multiplexers are selected and applied to the outputs that form the bus. table shows the registers that is selected by the bus for each of the four possible binary values of the selection lines

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

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 registers). The size of each multiplexer must be KX1 (no. of registers). 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 i/p lines & three selection lines.

To transfer information from a bus to one of many destination register, the inputs of all destination registeres are connected to the bus lines and activating the load control of the particular destination register squired.

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

for example, to transfer the content of register A to Register B the symbolic statement can be written a follows.

The symbolic statement using a bus symbol.

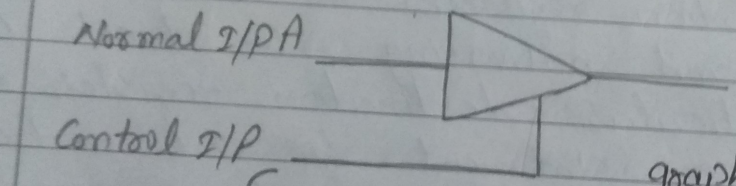
BUS ← A , B ← BUS

The content of register A is placed on the bus & the content of the bus is loaded into Register B by activating its load control input. If the bus is known to exist in the system, it may be convenient just to show the direct transfer.

B ← A

2. Constructing a Common Bus System using Three-State Bus Buffers :-

A three state gate is a digital circuit that exhibits three state. Two of the three state are the normal logic state 0 & 1, while the third state is a high impedance state. The high impedance state behaves like an open circuit (i.e. the output is disconnected & does not have logic significance). The most gate commonly used in the design of a bus system are the buffer gate.

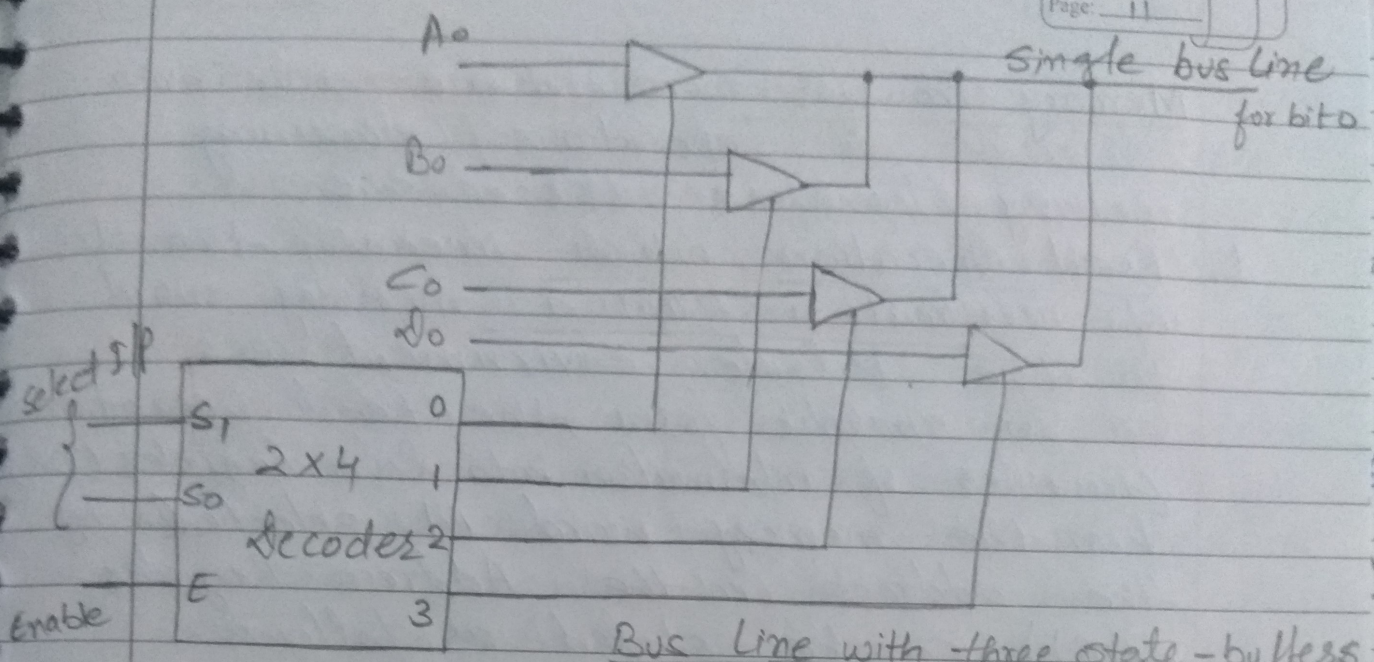


graphic symbol of a three state buffer

Control Input C	Normal Input A	Gate output X
0	0	High Impedance
0	1	High Impedance
1	0	0
1	1	1

truth table of a three state buffer

A bus system can be constructed with three-state gates instead of multiplexers. Figure shows a single bus line using three-state buffers. A single bus line is formed by connecting together the outputs of the four buffers.



Bus Line with three state buffers

The Control inputs to the buffers determine which of the four normal inputs will communicate with the bus line.

To ensure that no more than one control input is active at any given time, a 2x4 decoder is used. 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. The above circuit can replace one multiplexer.

Date:
Page: 12

Memory Transfer:- There are two main operations concerning memory transfers, these are:

1. **Read Operation**, which means transfer of information from a memory word to the outside environment.

In symbolic form, the Read Operation (transfer of information into Data Register (DR) from the memory word M selected by the address in the Address Register (AR)) can be stated as follows:

$$DR \leftarrow M[AR]$$

where AR stands for Address Register, from which the memory receives the address
DR stands for Data Register that receives data from the specified memory word.
M stands for the memory word.

2. **Write Operation**, which means transfer of information from the outside environment into a specified memory word.

In symbolic form, the write operation (transfer of information from Data Register (DR) into the memory word M selected by the address in the Address Register (AR)) can be stated as follows:

$$M[AR] \leftarrow DR$$

or

$$M[AR] \leftarrow R1$$

Date
Page 13

Microoperations: — The operations on the data in registers are called microoperations. also 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) micro-operations.

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

(i) Register transfer microoperation
(ii) Arithmetic microoperation
(iii) Logic microoperation
(iv) Shift microoperation

(i) **Register transfer microoperations:** Registers are designated by capital letters, sometimes followed by numbers (e.g. A, R1, 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 Register transfer micro-operation does not change the information content when the binary information moves from the source register to the destination while, the other three types of microoperations change the information content during the transfer."

The statement "R2 ← R1" denotes a transfer of the content of the R1 into register R2.

often action need to only occur if a certain condition is true. In digital systems, this is often done via a control signal, called 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

(ii) Arithmetic micro-operation:— Arithmetic micro-operations perform arithmetic operations on numeric data stored in registers. the basic arithmetic microoperations are addition, subtraction, Increment and decrement. The additional arithmetic microoperations are Add with carry, Subtract with borrow & transfer/load.

Symbolic designation	Description
• R3 ← R1+R2	Content of R1 Plus R2 transferred to R3
• R3 ← R1−R2 R3 ← R1+$\overline{R2}$+1	Content of R1 minus R2 transferred to R3 or R1 Plus the 2's Complement of R2 (subtraction)

• $R2 \leftarrow \overline{R2}$	Complement the content of R2 (1's comp)
• $R2 \leftarrow \overline{R2} + 1$	2's complement of the contents of R2 (negate)
• $R1 \leftarrow R1 + 1$	Increment the content of R1 by 1
$R1 \leftarrow R1 - 1$	Decrement the content of R1 by 1

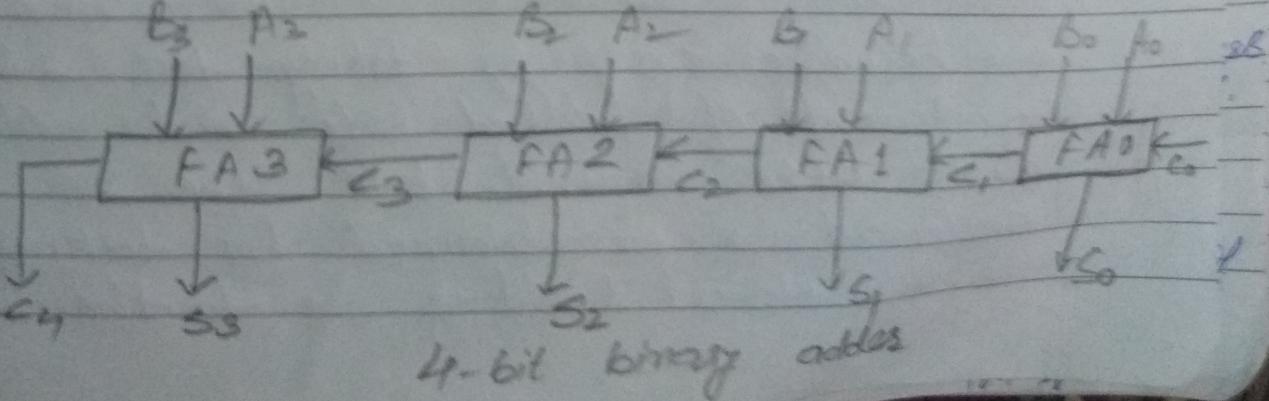
The arithmetic multiply and divide operations are not include in the table since in most computers, the multiplication operation is implemented with a sequence of add and shift microoperations, while the division operation is implemented with a sequence of subtract & shift micro-operations.

Hardware implementation of the Arithmetic M.O.

- **Binary Adder:-** To implement the add microoperation with hardware,

$$R3 \leftarrow R1 + R2$$

we need the registers that hold the data & the digital component that perform the arithmetic addition. The digital circuit that generates the arithmetic sum of two binary numbers of any length is called a binary adder.



4-bit binary adder

figure shows the 4-bit binary adder. This adder is constructed with 4-full adders connected in cascade, with the output carry from one full adder connected to the input carry of the next full adder. The corresponding augend bits of A and the addend bits of B are the two inputs to the successive full adders. The input carry to the binary adder is C_0 and the output carry is C_4. The S output of the full adder generate the required sum bits. An n-bit binary adder require n full adderss.

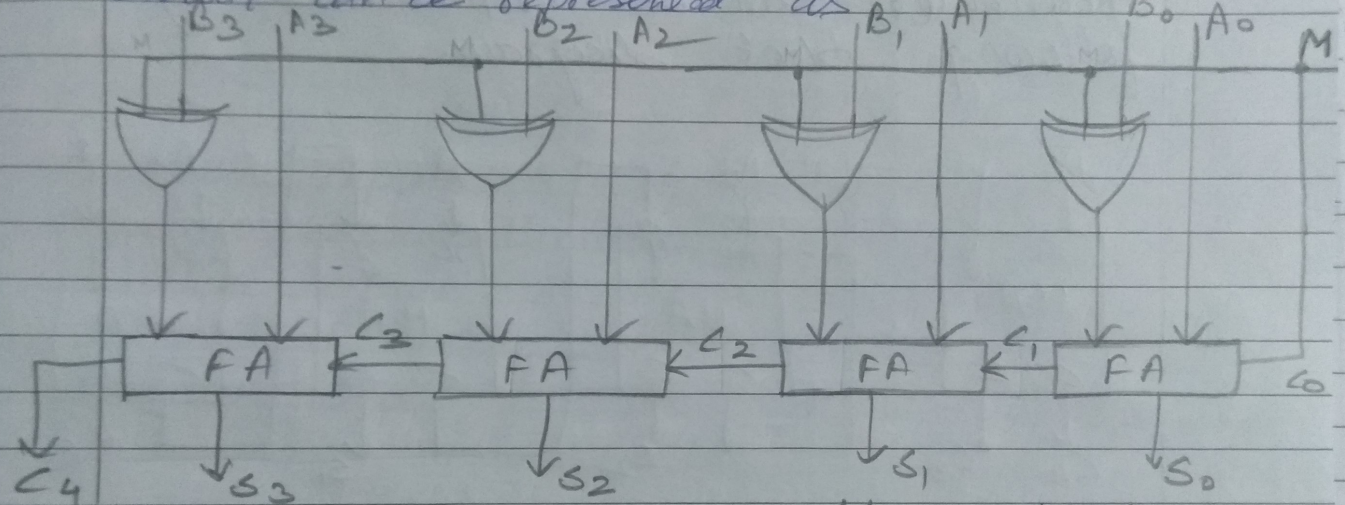
for example, to implement the arithmetic micro-operation R3 ← R1 + R2 using the 4-bit binary adder, we will have the 4 data bits for the A inputs come from one register (such as R1), & the 4-data bits for the B inputs come from another register (such as R2). The sum can be transferred to a third register (R3) or to one of the source register (R1 or R2) replacing its previous content.

- **Binary Adder - Subtractor:** — The subtraction micro-operation can be done easily by taking the 2's complement of addend bits and adding it to the augend bits. The 2's complement can be obtained by taking the 1's complement & adding one to the least

significant pair of bits. The 1's complement can be implemented with inverters, and one can be added to the sum through the input carry.

The Arithmetic micro-operation like addition and subtraction can be combined into one ~~com~~ common circuit by including an exclusive-OR gate with each full adder.

The block diagram for a 4-bit adder-subtractor circuit can be represented as



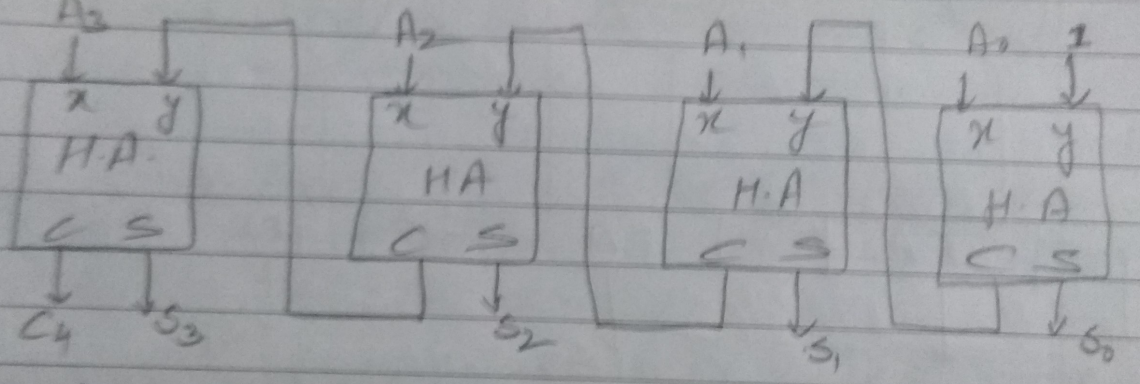
4-bit adder-subtractor

The mode input M controls the operation. When the mode input (M) is at low logic, i.e. '0' the circuit act as an adder and when M=1 the circuit becomes a subtractor. The ~~exct~~ exclusive-OR gate connected in series receives input M and one of the input B. When M=0, we have $B \oplus 0 = B$. The full adder receive the value of B, the input carry is 0 and the circuit performs A plus B. When M=1, we have $B \oplus 1 = \overline{B}$ and $C_0 = 1$. The B inputs are complemented, and a 1 is added through the input carry. the circuit performs the operation A plus the 2's complement of B.

transfer

- **Binary Incrementer:** The increment micro-operation adds one binary value to the value of binary variables stored in a register. For instance, a 4-bit register has a binary value 0110, when incremented by one the value becomes 0111.

The increment micro-operation is best implemented by a 4-bit combinational circuit incrementer, can be represented by following block diagram.



- A logic-1 is applied to one of the input of least significant half adder, & the other input is connected to the least significant bit of the number to be incremented.
- The output carry from the half adder is connected to one of the inputs of the next higher-order half adder.
- The binary incrementer circuit receives the four bits from A_0 through A_3 adds one to it, & generates the incremented output in S_0 through S_3.
- The output carry

Date:
Page: 19

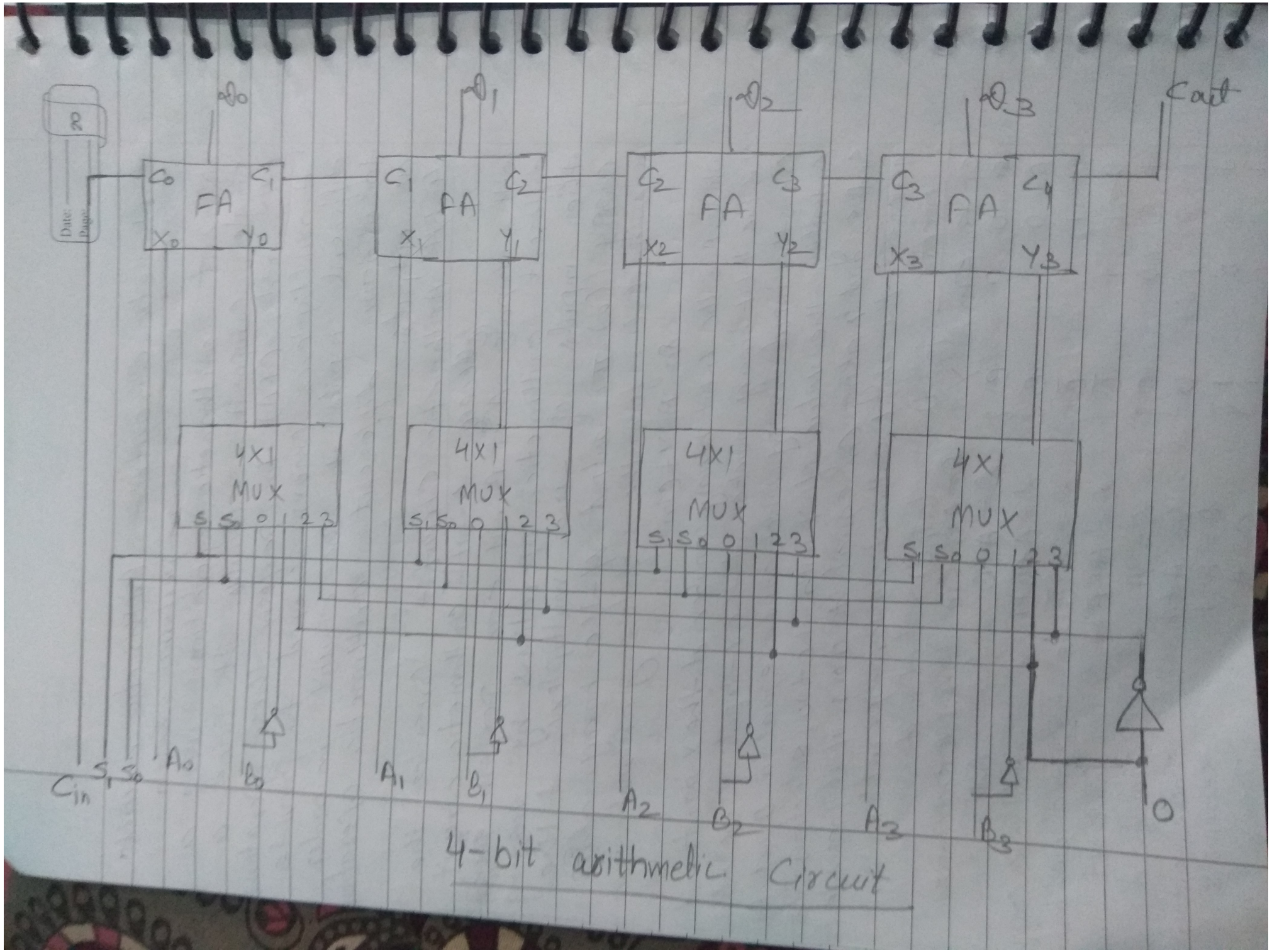
- **Arithmetic Circuits:-** Arithmetic circuits can perform seven different arithmetic operations using a single composite circuit. It uses a full adder (FA) to perform these operations. A multiplexer (MUX) is used to provide different inputs to the circuit in order to obtain different arithmetic operations as outputs.

4-bit Arithmetic Circuit:- Consider the 4-bit arithmetic circuit with inputs A & B. It can perform seven different arithmetic operations by varying the inputs of the multiplexer & the carry (C_0). The output of the binary adder is calculated from the following arithmetic sum.

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

where A is the 4-bit binary number at the X-inputs and Y is the 4-bit binary number at the Y-inputs of the adder. C_{in} is the input carry, which can be equal to 0 or 1. By controlling the value of Y with the two selection i/p's S_1 & S_0 and making C_{in} equal to 0 or 1, it is possible to generate the eight arithmetic micro-operations.

Select			(Input)	(output)	
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	$D = A+B+1$	Add with carry
0	1	0	$\bar{B}$	$D = A+\bar{B}$	Subtract with borrow
0	1	1	$\bar{B}$	$D = A+\bar{B}+1$	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



4-bit arithmetic Circuit

- When $S_1 S_0 = 00$, the value of B is applied to the Y inputs of the adder. Now If $C_{in} = 0$, then $D = A + B$, & If $C_{in} = 1$ then output $D = A + B + 1$.
- When $S_1 S_0 = 01$, the complement of B is applied to the Y inputs of the adder.
  if $C_{in} = 1$ then $D = A + \overline{B} + 1$ (or $A - B$)
  if $C_{in} = 0$ then $D = A + \overline{B}$ (or $A - B - 1$)
- When $S_1 S_0 = 10$, the inputs from B are neglected & instead, all 0's are inserted into the Y inputs. The O/P become $D = A + \overline{0} + C_{in}$ or $D = A + C_{in}$
  if $C_{in} = 0$ then $D = A$ (direct transfer from I/P A to o/p D)
  If $C_{in} = 1$ then $D = A + 1$ (value of A is incremented by 1.
- When $S_1 S_0 = 11$, all 1's are inserted into the Y inputs of the adder. The output become
  $D = A + 1111 + C_{in}$ (a number with all 1's is equal to the 2's complement of 1) (the 2's complement of binary 0001 is 1111). So $D = A +$ 2's complement of $1 = A - 1$. $\boxed{D = A - 1}$.

  When $C_{in} = 0$ $D = A - 1$, decrement operation
  When $C_{in} = 1$, then $D = A - 1 + 1 = A$ which causes a direct transfer from input A to output D.
  Note: The microoperation $D = A$ is generated twice, so there are only seven different micro-operations in the arithmetic circuit.

(iii) Logic Micro-operations:— Logic Micro-operations specify binary operations performed for strings of bits in registers. These operations consider each bit of the register separately and treat them as binary variables.

example. $R_1 \leftarrow R_1 \oplus R_2$

R_1 1010 (Content of R1)

R_2 ⊕ 1100 (Content of R2)

R_1 after P=1 0110 (Content of R_1 after P=1)

where P is a Control function.

These are 16 different logic operations with 2 binary variables.

x y	f_0	f_1	f_2	f_3	f_4	f_5	f_6	f_7	f_8	f_9	f_{10}	f_{11}	f_{12}	f_{13}	f_{14}	f_{15}
0 0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
0 1	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
1 0	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
1 1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1

Truth table for 16 functions of two variables

The 16 different logic microoperations can be determined from the above truth tables. In this table, each of the 16 columns f_0 through f_{15} represents a truth table of one possible Boolean function for the two variable x & y.

The 16 Boolean functions of two variables x & y are expressed in algebric form in the first column of the table. The logic Boolean function in the first column

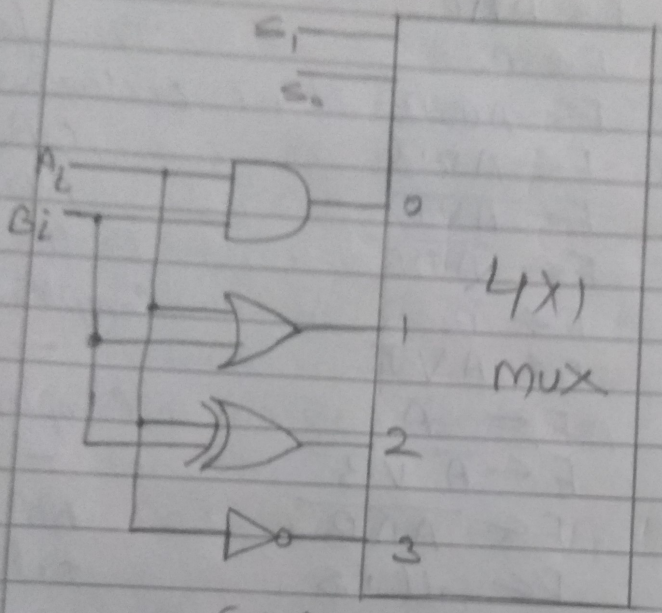
Sixteen Logic Microoperation

Boolean function	Microoperation	Name
$F_0 = 0$	$F \leftarrow 0$	Clear
$F_1 = xy$	$F \leftarrow A \wedge B$	AND
$F_2 = xy'$	$F \leftarrow A \wedge B'$	—
$F_3 = x$	$F \leftarrow A$	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
$F_{15} = 1$	$F \leftarrow$ all 1's	set to all 1's

Of the table represents a relationship between two binary variables x & y. The logic microoperation in the second column represent a relationship b/w the binary content of two registers A & B. each of bit of the register is treated as a binary variable. & the microoperation is performed on the string of bits stored in the registers.

Hardware Implementation:- The hardware implementation of logic microoperations required that logic gates be inserted for each bit or pair of bits in the register to perform the required logic function.

- Although there are 16 logic microoperations, most computers use only four - AND, OR, X-OR & complement, from which all others can be derived.



(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) Functional table

One stage of logic circuit

The diagram shows one typical stage with subscript i. For a logic circuit with n bits, the diagram must be repeated n times for $i = 0, 1, 2 \ldots n-1$.

Some Applications:— logic micro-operations 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 value into a register.

- Consider the data in a register A. In

another register, B, is bit data that will be used to modify the contents of A.
The different applications are

★ selective-set	$A \leftarrow A + B$
★ selective-Complement	$A \leftarrow A \oplus B$
★ selective Clear	$A \leftarrow A \cdot \overline{B}$
★ Mask	$A \leftarrow A \cdot B$
★ clear	$A \leftarrow A \oplus B$
★ Insert	$A \leftarrow (A \cdot B) + C$

Selective set:- • In a selective set operation, the bit pattern in B is used to set certain bits in A.
• If a bit in B is set to 1, that same position in A gets set to 1, otherwise that bit in A keeps its previous value.
• OR microoperation can be used to selectively set bits of a register.

```
  1  0    1  0    A (before)
  1  1    0  0    B (logic operand)
-------------------
  1  1    1  0    A after (A ← A+B)
```

Selective Complement :- In a selective complement operation, the bit pattern in B is used to complement certain bits in A
• if a bit in B is set to 1, that same position in A gets complemented from its original value, otherwise it is unchanged.
• The exclusive-OR microoperation can be used to selectively complement bits of a register.

Date: Page: 26

```
1 0 1 0    A (before)
1 1 0 0    B
-----------
0 1 1 0    A (after)   (A ← A ⊕ B)
```

- The selective clear:— In a selective clear operation, the bit pattern in B is used to clear certain bits in A.

```
1 1 0 0    A (before)
1 0 1 0    B
-----------
0 1 0 0    A (after)   (A ← A · B')
```

- This operation clears to 0 the bits in A only where there are corresponding 1's in B, otherwise it is unchanged.
- The boolean function performed on the individual bits is AB'

- Mask operation:— The mask operation is similar to the selective clear operation except that the bit of A are cleared only where there are corresponding 0's in B.

```
1 1 0 0    A (Before)
1 0 1 0    B
-----------
1 0 0 0    A (after)   (A ← A · B)
```

- If a bit in B is set to 0, that same position in A gets set to 0, otherwise it is unchanged.
- The AND micro operation can be used to mask or delete the bits. The mask operation is more convenient to use than the selective clear operation because most computers

provide an AND insertion.

- Clear operation:- In a clear operation, if the bits in the same position in A and B are the same, they are cleared in A, otherwise they are set in A.

```
1 1 0 0   A (Before)
1 0 1 0   B
---------
0 1 1 0   A (After)      A ← A ⊕ B
```

- It compares words in A & B and produces all 0's result if the two numbers are equal. This operation is achieved by X-OR microoperation.

- Insert operation:- The insert operation inserts a new value into a group of bits. This is done as.
  - A mask operation to clear the desired bit position, followed by
  - An OR operation to introduce the new bits into the desired positions.

Ex. ⇒ suppose that an A register contains 8 bits, 0110 1010, to replace the 4 leftmost bits by the value 1001, we first mask the four unwanted bits.

```
0 1 1 0 1 0 1 0   A (Before)
0 0 0 0 1 1 1 1   B (mask)
---------------
0 0 0 0 1 0 1 0   A (after masking)
```

Then insert the new value:

```
0 0 0 0 1 0 1 0   A (before)
1 0 0 1 0 0 0 0   B (insert)
---------------
1 0 0 1 1 0 1 0   A (after insertion)
```

Date:
Page: 25

(iv) Shift micro-operation :- Shift micro-operations are used for serial transfer of data or information. They are also used in conjunction with arithmetic micro-operation, logic micro-operation, & other data-processing operations.

The contents of a register can be shifted to the left or the right. At the same time that the bits are shifted, the first flip-flop receives its binary information from the serial input.

These are three type of shifts:

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

1. Logical Shift:- • A logical shift is one that transfer 0 through the serial input.

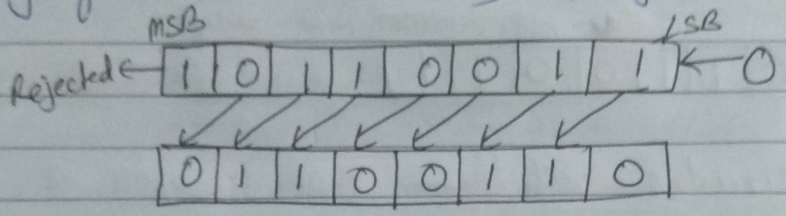
• We use the symbol shl for logical shift left and shr for shift right.

• Ex- R1 ← shl R1
R2 ← shr R2

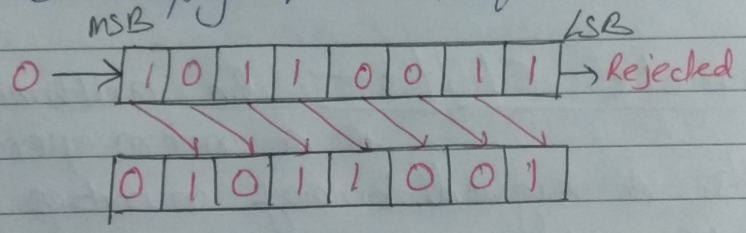
are two microoperation that specify a 1-bit shift to the left of the content of register R1 & a 1-bit shift to the right of the content of register R2. The register symbol must be the same on both sides of the arrow.

Date:
Page: 29

(a) Logical shift left :- In this shift one position moves each bit to the left one by one. The empty least significant bit (LSB) is filled with zero (i.e. serial input) and the most significant bit (MSB) is rejected.



(b) Right Logical shift :- In this one position moves each bit to the right one by one & the least significant bit (LSB) is rejected and the empty MSB is filled with zero.

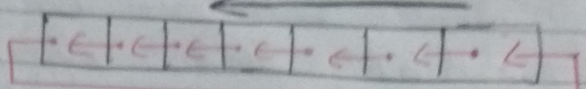


2. Circular shift:- The circular shift (also known as a rotate operation) circulates the bits of the register around the two ends without loss of information. This is accomplished by connecting the serial output of the shift register to its serial input. The symbol cil and cir are used for circular shift left & circular shift right respectively.

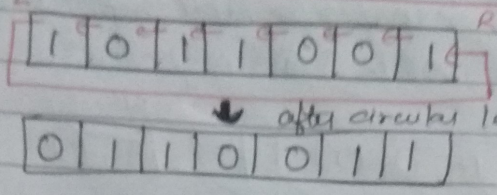
Ex- R1 ← cil R1
R2 ← cir R2

Date:
Page: 36

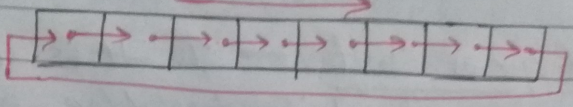
(a) Left circular shift:—



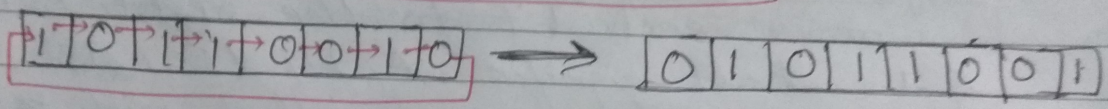
Ex



(b) Right circular shift:—



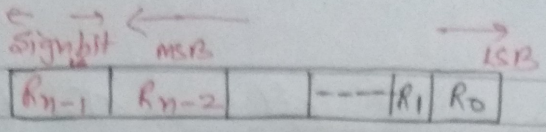
Ex



3. Arithmetic Shift:— An arithmetic shift is a micro-operation that shifts a signed binary number to the left or right.

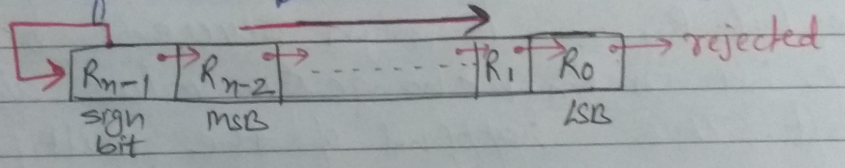
- An arithmetic shift left multiplies a signed binary number by 2.
- An arithmetic shift-right divides the number by 2.

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. In a register the leftmost bit holds the sign bit, & the remaining bits hold the number. The sign bit is 0 for positive & 1 for negative. negative no. are in 2's Complement form.

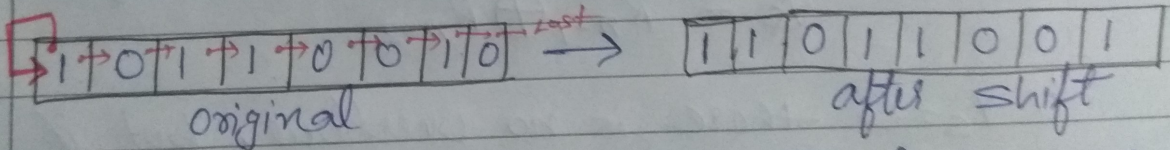


this is a n-bit registes. bit R_{n-1} in the leftmost position holds the sign bit, R_{n-2} is the most significant bit of the number & R_0 is the least significant bit.

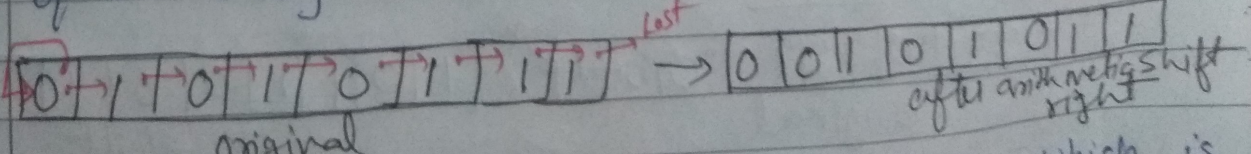
(a) Arithmetic shift right :— In this one position moves each bit to the right one by one and the least significant bit is rejected and the MSB is filled with the value of the sign bit & the



sign bit remains the same.
[bit R_{n-1} (is the sign bit) remains the same, R_{n-2} receives the bit from R_{n-1}, & so on for other bits in the registes. The bit in R_0 is lost].

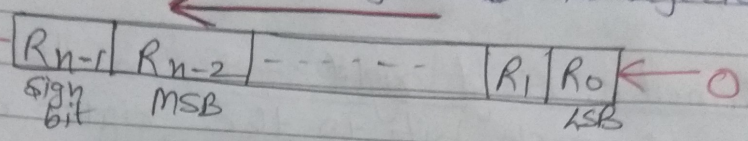
Ex 

The original number was −78 & the new number is −39, which is the result of dividing −78 by 2. (negative number are in 2's complement form) (first find the 2's complement of the bits).

Ex 

the original no was 87 & the new no is 43, which is the result of dividing 87 by 2. (43.5 ~ 43)

(b) Arithmetic shift left :- In this one position moves each bit to the left one by one. The empty least significant bit (LSB) is filled with zero & the initial bit (sign bit) is rejected.



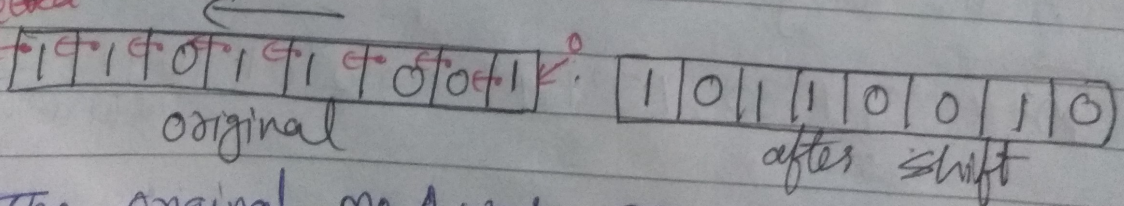
The arithmetic shift left inserts a 0 into R_0, and shifts all other bits to the left. The initial bit of R_{n-1} is lost & replaced by the bit R_{n-2}. A sign reversal occurs if the bit in R_{n-1} changes in value after the shift. This happens if the multiplication by 2 cause an overflow. An overflow 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.

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

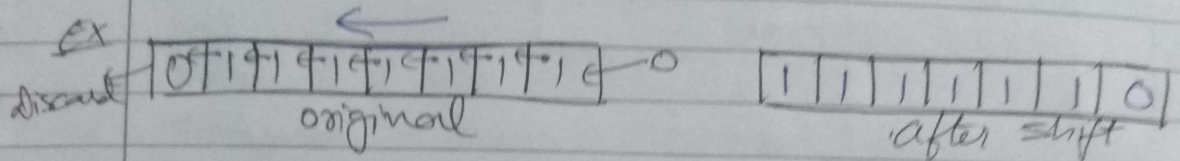
If $V_S = 0$ there is no overflow, but if $V_S = 1$, there is an overflow & a sign reversal after the shift. V_S must be transferred into the overflow flip-flop with the same clock pulse that shift the register.

Ex.



The original no. was -39 & the new no. is -78, which is the result of multiplying -39 by 2

The operation is valid because no overflow occured.



The original no was 127 & the new number is -2. here the result is not valid because an overflow has occured. The expected answer is $127 \times 2 = 254$, can not be represented by an 8-bit sign pattern.

Hardware Implementation:-

take an example $A = A_3 A_2 A_1 A_0$ → discarded

for logical shift right = shr $0 \rightarrow A_3 \rightarrow A_2 \rightarrow A_1$

here 0 is an initial input for shift right

can be written as I_R

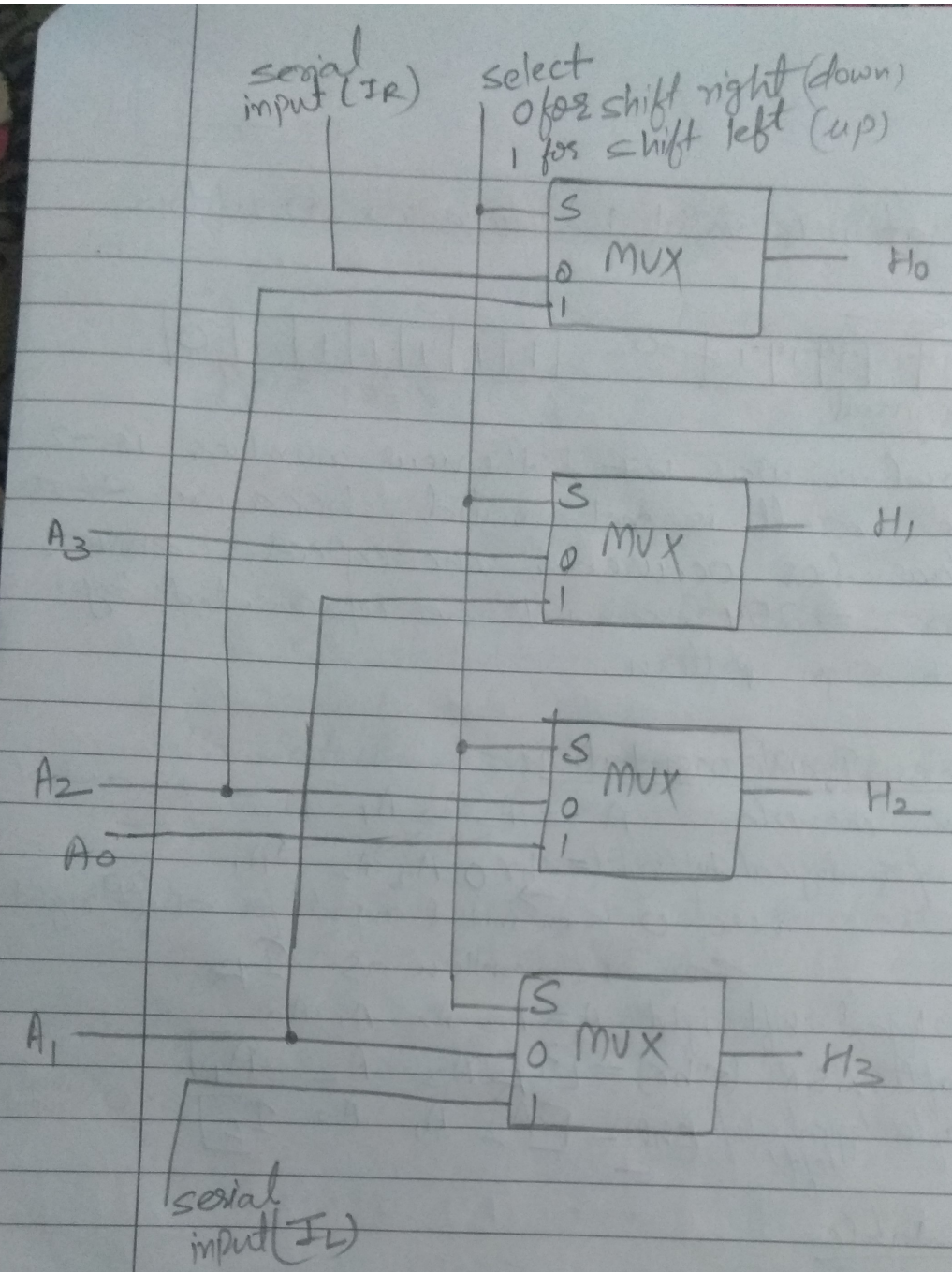
so for logical shift right $A = A_3\ A_2\ A_1\ A_0$

after L.S.R (shr) $= [I_R\ A_3\ A_2\ A_1]$

after logical shift left (shl) $= [A_2\ A_1\ A_0\ I_L] \leftarrow 0$ (initial input)

Functional table

Select	output			
S	H_0	H_1	H_2	H_3
shr 0	I_R	A_3	A_2	A_1
shl 1	A_2	A_1	A_0	I_L



A combinational circuit shifter can be constructed with multiplexers. The 4-bit shifter has four data inputs, A_0 through A_3, & four data o/p H_0 through H_3. There are two serial inputs, one for shift left (I_L) & the other for shift right (I_R).

The two serial inputs can be controlled by another multiplexer to provide the three possible types of shifts.