Chapter 3
Microoperations
LH-4

Prepared By: Rolisha Sthapit

CONTENTS

Arithmetic Microoperations, Logic Microoperations, Shift Microoperations, Arithmetic Logic Shift Unit.

Computer Organization vs Architecture

- **Computer Architecture** refers to those attributes of a system that have a direct impact on the logical execution of a program. Examples:

- the instruction set

- the number of bits used to represent various data types

- I/O mechanisms

- memory addressing techniques

- **Computer Organization** refers to the operational units and their interconnections that

realize the architectural specifications. Examples are things that are transparent to the programmer:

- control signals

- interfaces between computer and peripherals

- the memory technology being used.

- So, for example, the fact that a multiply instruction is available is a computer architecture issue. How that multiply is implemented is a computer organization issue.
- **Architecture** is those attributes visible to the programmer - Instruction set, number of bits used for data representation, I/O mechanisms, addressing techniques. e.g. Is there a multiply instruction?
- **Organization** is how features are implemented - Control signals, interfaces, memory technology. e.g. Is there a hardware multiply unit or is it done by repeated addition?
- **Computer architecture** is concerned with the structure and behavior of computer system as seen by the user.
- **Computer organization** is concerned with the way the hardware components operate and the way they are connected together to form a computer system.

Register and Register Transfer Language (RTL)

Register:

- Register is the storage device, inside CPU, of data on which microoperations are performed.
- The operations executed on data stored in registers are called microoperations. A microoperation is an elementary operation performed on the information stored in one or more registers. The result of the operation may replace the previous binary information of a register or may be transferred to another register. Examples of microoperations are shift, count, clear and load.
- The internal hardware organization of a digital computer is best defined by specifying:

  - The set of registers it contains and their function.

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

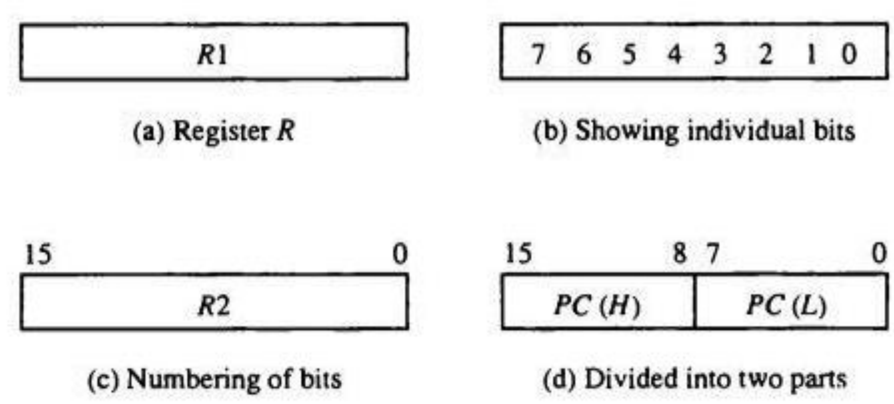
  - The control that initiates the sequence of microoperations.

- The language, which is basically used to express the transfer of data among the registers, is called **Register Transfer Language (RTL).** It is the symbolic notation used to describe the microoperation transfers among registers. In such transfer, one of the source or destination should be register (not necessarily both).

Register Transfer

- Computer registers are designated by capital letters.
- For example, the register that holds an address for the memory unit is usually called a memory address register and is designated by the name MAR. Other designations for registers are PC (for program counter), IR (for instruction register, and R1 (for processor register).
- The most common way to represent a register is by a rectangular box with the name of the register inside. Fig (a).
- The individual flip-flops in an n-bit register are numbered in sequence from 0 through n-1, starting from 0 in the rightmost position and increasing the numbers toward the left. For e.g. 8-bit register numbered: Fig (b).
- The numbering of bits in a 8-bit register can be marked on top of the box. Fig (c).

- A 16-bit register is partitioned into two parts in (d). Bits 0 through 7 are assigned the symbol L (for low byte) and bits 8 through 15 are assigned the symbol H (for high byte). The name of the 16 bit register is PC. The symbol PC(0—7) or PC(L) refers to the low order byte and PC(8—15) or PC(H) to the high order byte.
- Information transfer from one register to another is designated in symbolic form by means of a replacement operator. $R2 \leftarrow R1$

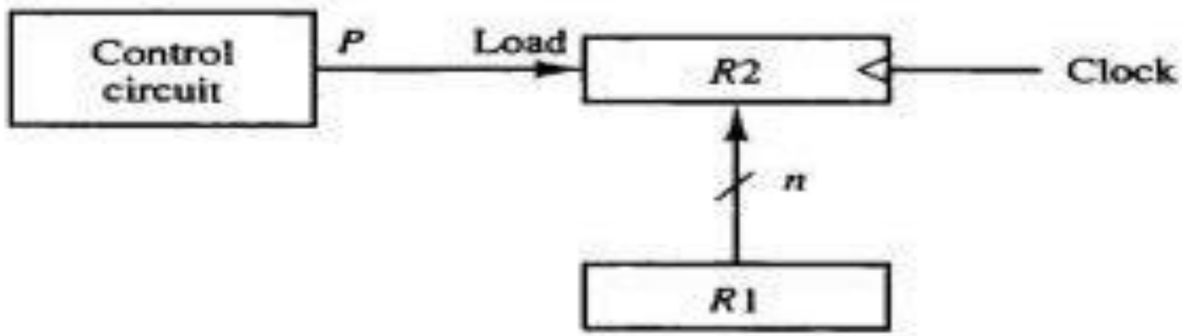


Control function

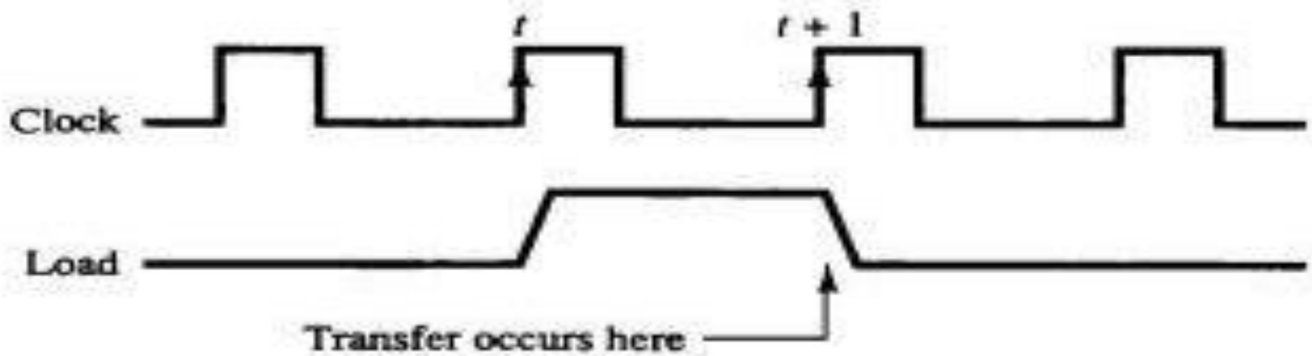
If there is predetermined control condition like If (P=1) then ($R2 \leftarrow R1$), then we can write the statement as

$$P: R2 \leftarrow R1$$

where P is control signal usually a control function which is Boolean variable that is equal to 1 or 0.



(a) Block diagram



(b) Timing diagram

- 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 transfers 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 $T: R2 \leftarrow R1, R1 \leftarrow R2$ denotes an operation that exchanges the contents of two registers during one common clock pulse provided that T = 1.

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

Fig: Basic Symbols for Register Transfers

→ For example, RTL of fetch cycle can be written as:

T1: MAR←PC

T2: MBR←[MAR]

T3: IR←MBR

T4: unspecified; PC←PC+1

The notation (T1, T2, T3, T4) represents successive time units. All three units are of equal duration. A time unit is defined by regularly spaced clock pulses. The operations performed within this single unit of time are called microoperations. A single time unit can contain one or more microoperations. Since each microoperation can specifies the transfer of data into or out of a register, such type is called **RTL**.

Bus and Memory Transfer

Bus

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

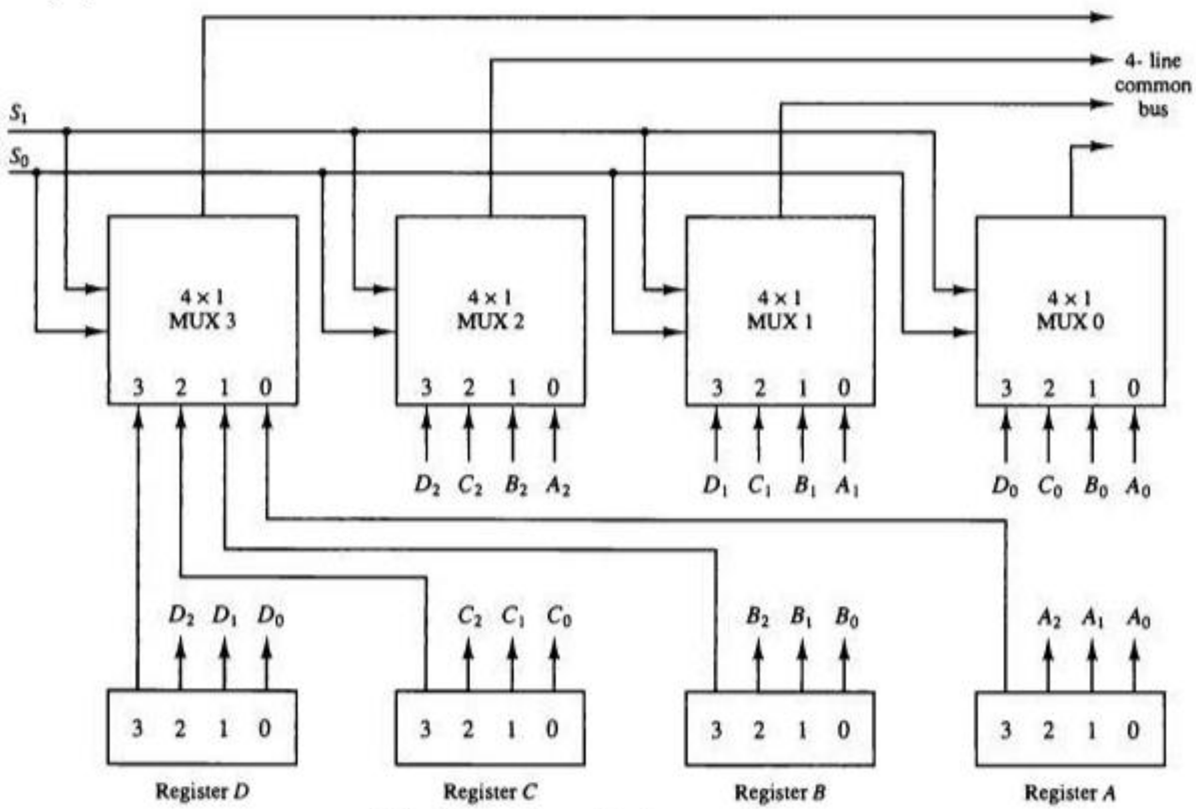


Fig: Bus system for four registers.

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

Fig: Function table for Bus

- 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.
- For example, a common bus for eight registers of 16 bits each requires 16 multiplexers, one for each line in the bus. Each multiplexer must have eight data input lines and three selection lines to multiplex one significant bit in the eight registers.
- The transfer of information from a bus into one of many destination registers can be accomplished by connecting the bus lines to the inputs of all destination registers and activating the load control of the particular destination register selected.

Memory Transfer

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

$$\text{Read: } DR \leftarrow M[AR]$$ `

This causes a transfer of information into DR from the memory word M selected by the address in AR.

- The write operation

- The write operation transfers the content of data register to a memory word M selected by memory address. Assume that the input data are in register R_1 and the address in AR, the write operation can be stated as:

$$\text{Write: } M[AR] \leftarrow R_1$$

This causes a transfer of information from R_1 into the memory word M selected by the address in R.

Microoperations and its types

- The operations on the data in registers are called microoperations.
- Alternatively we can say that an elementary operation performed during one clock pulse on the information stored in one or more registers is called 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 (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 microoperations

ii) Arithmetic microoperations

iii) Logic microoperations

iv) Shift microoperations

Arithmetic Microoperation

- The basic arithmetic microoperations are: addition, subtraction, increment and decrement. The additional arithmetic operations are with carry, subtract with borrow and transfer/load.

Summary of typical arithmetic microoperations

Symbolic Designation	Description
$R3 \leftarrow R1 + R2$	Contents of R1 plus R2 transferred to R3
$R3 \leftarrow R1 + R2' + 1$	Contents of R1 minus R2 transferred to R3
$R1 \leftarrow R1 + 1$	Increment the contents of R1 by one
$R1 \leftarrow R1 - 1$	Decrement the contents of R1 by one
$R3 \leftarrow R1 + R2 + 1$	Add with carry
$R3 \leftarrow R1 + R2'$	Subtract with borrow
$R1 \leftarrow R1' +1$	2's complement the contents of R1 (negate)

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.

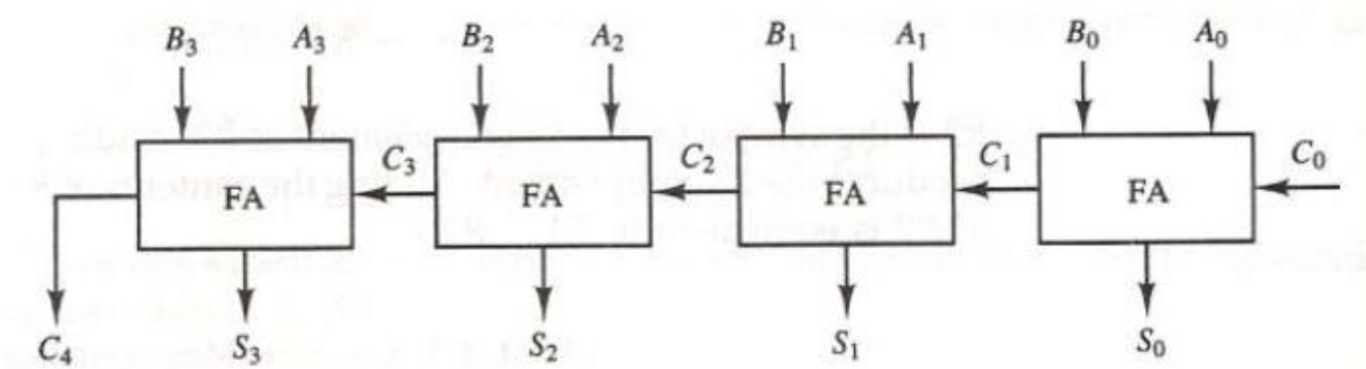


Fig.: 4-bit binary adder

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.

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 Bi to an inverter) and then add 1 to the least significant bit (by setting carry C0 to 1) of binary adder, then we can make a binary subtractor.

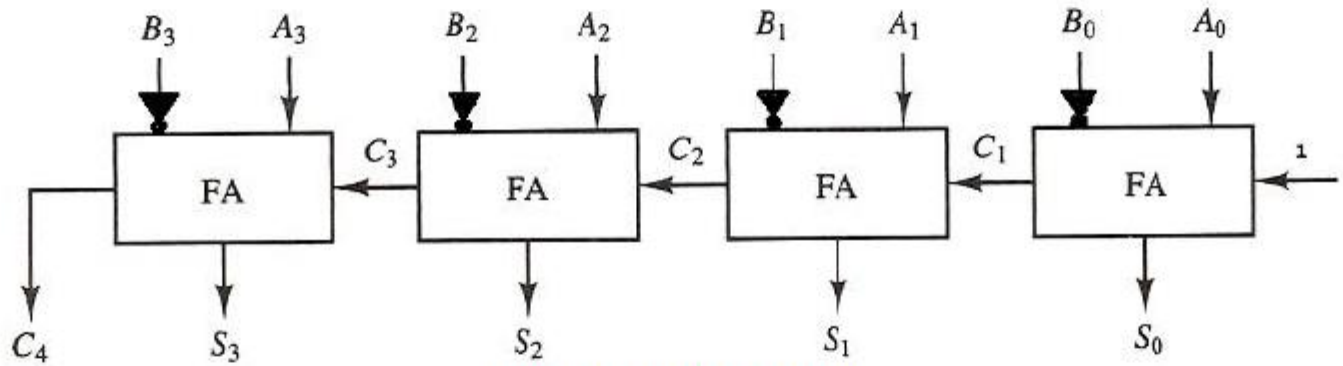


fig.: 4-bit binary subtractor

Binary Adder-Subtractor

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

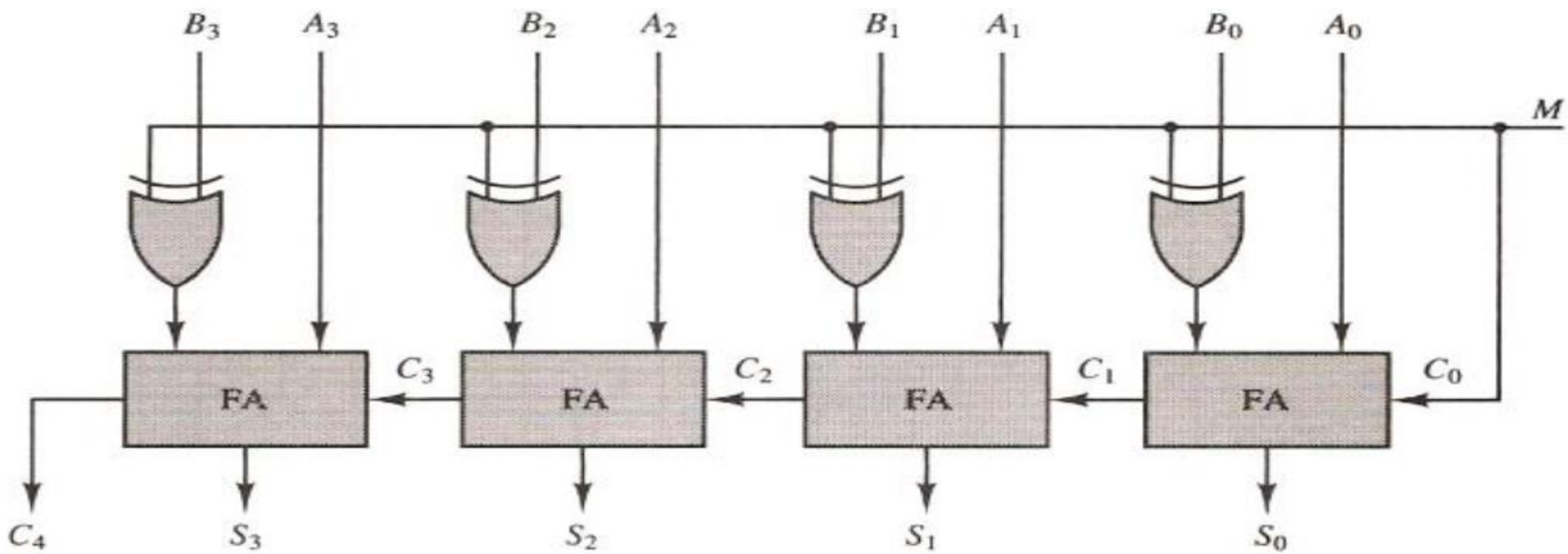


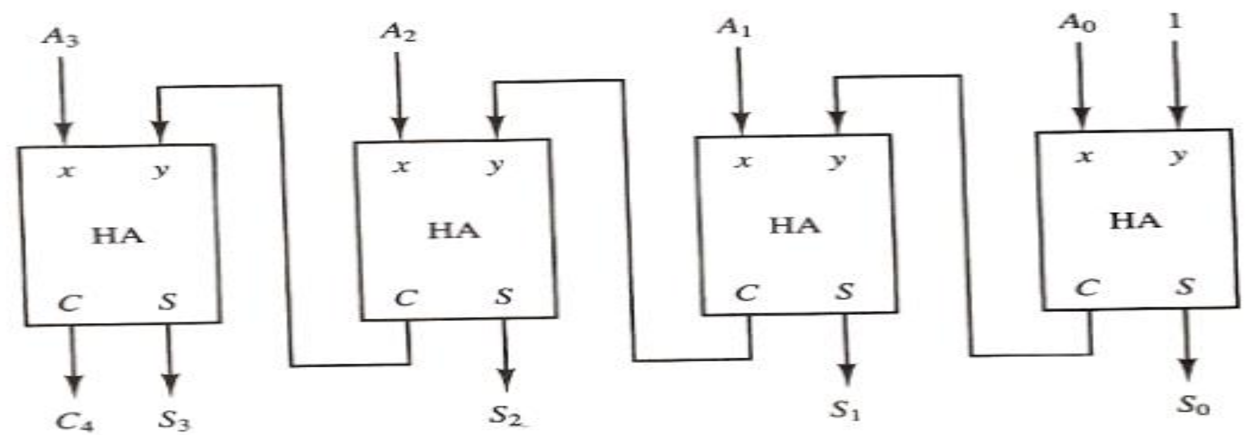
Fig.: 4-bit adder-subtractor

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).

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.



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			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	$\overline{B}$	$D = A + \overline{B}$	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

addition

When $S_1S_0 = 00$, the value of B is applied to the Y inputs of the adder. If $C_{in} = 0$, the output $D = A + B$. If $C_{in} = 1$, output $D = A + B + 1$. Both cases perform the add microoperation with or without adding the input carry.

subtraction

When $S_1S_0 = 01$, the complement of B is applied to the Y inputs of the adder. If $C_{in} = 1$, then $D = A + \overline{B} + 1$. This produces A plus the 2's complement of B, which is equivalent to a subtraction of $A - B$. When $C_{in} = 0$, then $D = A + \overline{B}$. This is equivalent to a subtract with borrow, that is, $A - B - 1$.

increment

When $S_1S_0 = 10$, the inputs from B are neglected, and instead, all 0's are inserted into the Y inputs. The output becomes $D = A + 0 + C_{in}$. This gives $D = A$ when $C_{in} = 0$ and $D = A + 1$ when $C_{in} = 1$. In the first case we have a direct transfer from input A to output D. In the second case, the value of A is incremented by 1.

decrement

When $S_1S_0 = 11$, all 1's are inserted into the Y inputs of the adder to produce the decrement operation $D = A - 1$ when $C_{in} = 0$. This is because a number with all 1's is equal to the 2's complement of 1 (the 2's complement of binary 0001 is 1111). Adding a number A to the 2's complement of 1 produces $F = A + 2\text{'s complement of } 1 = A - 1$. When $C_{in} = 1$, then $D = A - 1 + 1 = A$, which causes a direct transfer from input A to output D. Note that the microoperation $D = A$ is generated twice, so there are only seven distinct microoperations in the arithmetic circuit.

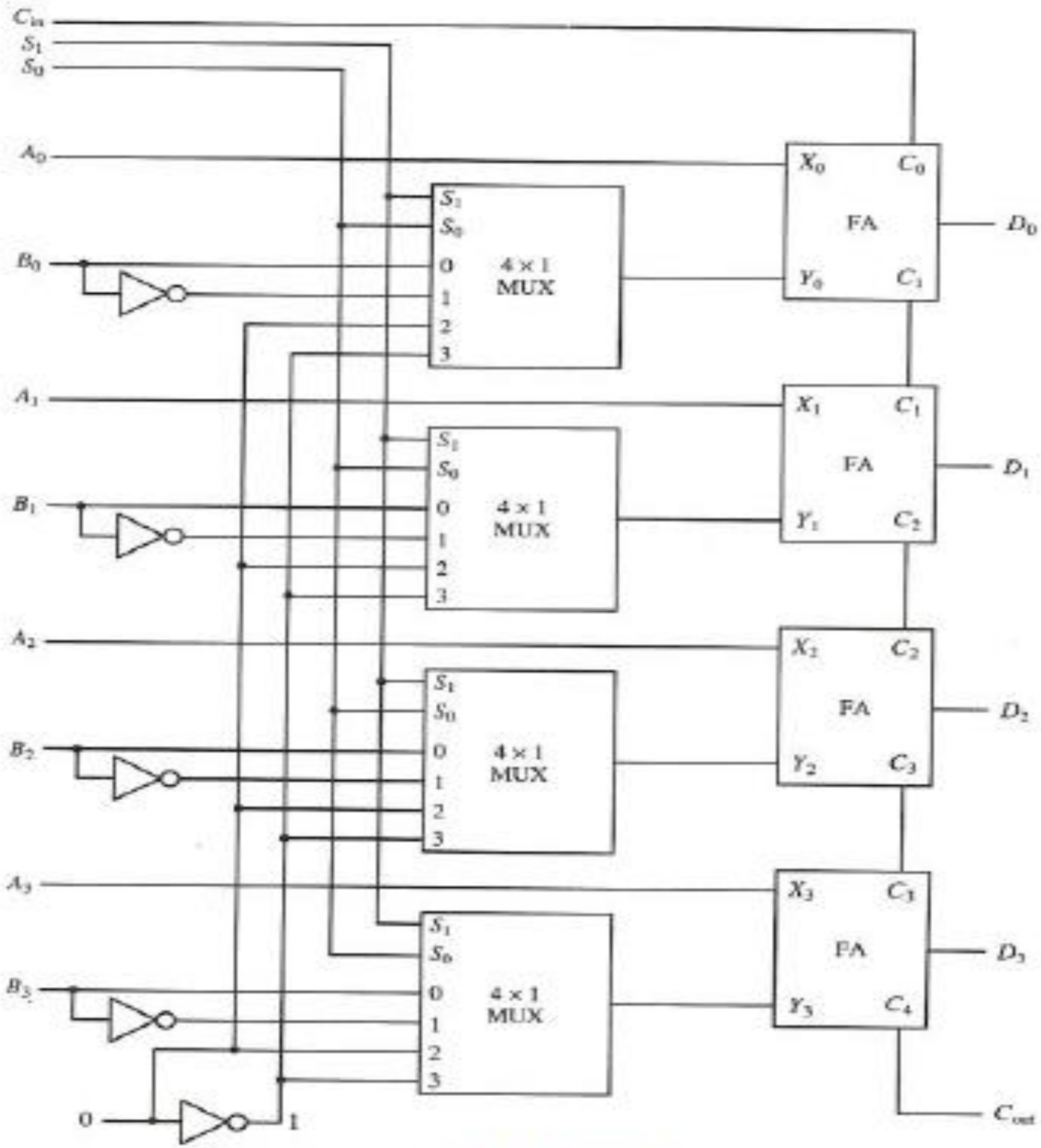


Fig: 4-bit arithmetic circuit

Logic Microoperations

- 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 ($\wedge$), OR ($\vee$), XOR ($\oplus$), Complement/NOT
- The others can be created from combination of these four functions.

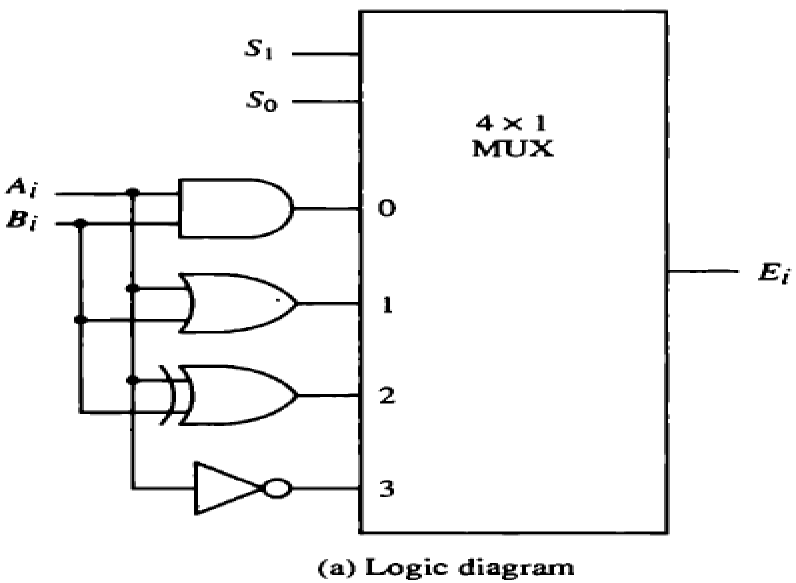
TABLE 4-6 Sixteen Logic Microoperations

Boolean function	Microoperation	Name
$F_0 = 0$	$F \leftarrow 0$	Clear
$F_1 = xy$	$F \leftarrow A \wedge B$	AND
$F_2 = xy'$	$F \leftarrow A \wedge \overline{B}$	
$F_3 = x$	$F \leftarrow A$	Transfer A
$F_4 = x'y$	$F \leftarrow \overline{A} \wedge B$	
$F_5 = y$	$F \leftarrow B$	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
$F_{10} = y'$	$F \leftarrow \overline{B}$	Complement B
$F_{11} = x + y'$	$F \leftarrow A \vee \overline{B}$	
$F_{12} = x'$	$F \leftarrow \overline{A}$	Complement A
$F_{13} = x' + y$	$F \leftarrow \overline{A} \vee B$	
$F_{14} = (xy)'$	$F \leftarrow \overline{A \wedge B}$	NAND
$F_{15} = 1$	$F \leftarrow$ all 1's	Set to all 1's

Hardware implementation

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

Figure 4-10 One stage of logic circuit.



(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.
- Individual bits of a registers are operated with other corresponding register bits.
- They can be used to change values, delete a group of bits or insert new bit values into a register.

Consider a data in register **A** . In other register, **B** is bit data that will be used to modify the contents of **A**.

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.

1 1 0 0	A_t	
1 0 1 0	B	
1 1 1 0	A_{t+1}	$(A \leftarrow A + B)$

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.

1 1 0 0	A_t	
1 0 1 0	B	
0 1 1 0	A_{t+1}	$(A \leftarrow A \oplus B)$

Selective-clear

In a selective clear operation, the bit pattern in B is used to clear certain bits in A. If a bit in B is set to 1, that same position in A gets set to 0, otherwise it is unchanged.

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

Mask Operation

In a mask operation, the bit pattern in B is used to clear certain bits in A. If a bit in B is set to 0, that same position in A gets set to 0, otherwise it is unchanged.

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

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 inA.

1 1 0 0	A_t	
1 0 1 0	B	
0 1 1 0	A_{t+1}	$(A \leftarrow A \oplus B)$

Insert Operation

An insert operation is used to introduce a specific bit pattern into A register, leaving the other bit positions unchanged. This is done as mask operation to clear the desired bit positions, followed by An OR operation to introduce the new bits into the desired positions

Example

» Suppose you wanted to introduce 1010 into the low order four bits of A:

1101 1000 1011 0001 (Original)

1101 1000 1011 1010 (Desired)

1101 1000 1011 0001	A (Original)
1111 1111 1111 0000	Mask
1101 1000 1011 0000	A
0000 0000 0000 1010	Added bits
1101 1000 1011 1010	A (Desired)

Shift Microoperation

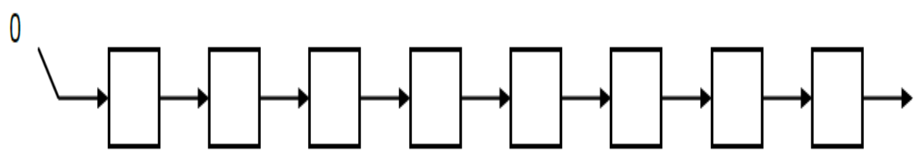
- The operation that changes the adjacent bit position of the binary values stored in the register is known as shift microoperation. They are used for serial transfer of data. The shift microoperations are classified into 3 types:

    1. Logical Shift
    2. Circular Shift
    3. Arithmetic Shift

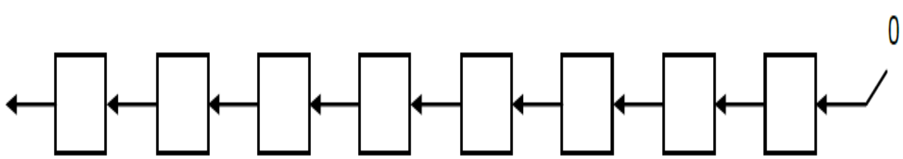
1. Logical Shift:

A logical shift is the one that transfers 0 through the serial input. In RTL, following notation is used

- shr for logical shift right ($R3 \leftarrow$ shr $R3$)
- shl for logical shift left ($R4 \leftarrow$ shl $R4$)



Logical right shift (shr)



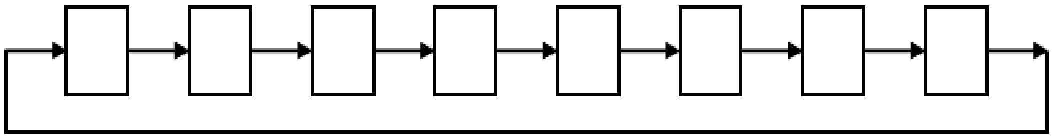
Logical left shift (shl)

Example
44 = 01000101
shr = 00100010
shl = 10001010

2. Circular Shift:

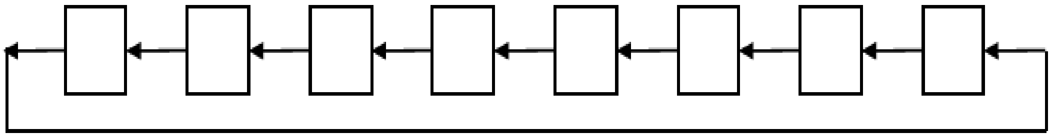
Circular shift circulates the bits of the register around the two ends without the loss of information. It is also known as rotate operation.

Right circular shift operation



Example
00101100
cir = 00010110
cil = 01011000

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 Operation

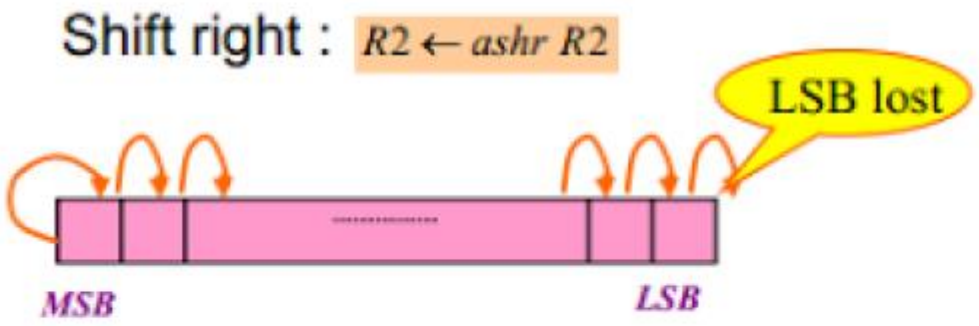
An arithmetic shift is meant for signed binary numbers (integer). An arithmetic left shift multiplies a signed number by two and an arithmetic right shift divides a signed number by two. The main distinction of an arithmetic shift is that it must keep the sign of the number the same as it performs the multiplication or division.

In a RTL, the following notation is used

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

a. **Arithmetic Shift Right:**

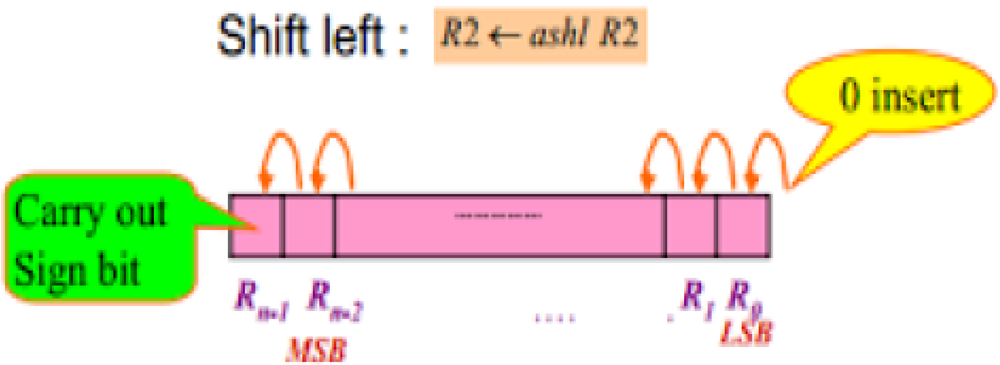
It leaves the signed bit unchanged and shifts the number including the signed bit to the right.



Eg: 10110010
ashr : 11011001

b. Arithmetic Shift Left:

It inserts a 0 into last bit and shifts all other bits to the left. The initial bit is lost.



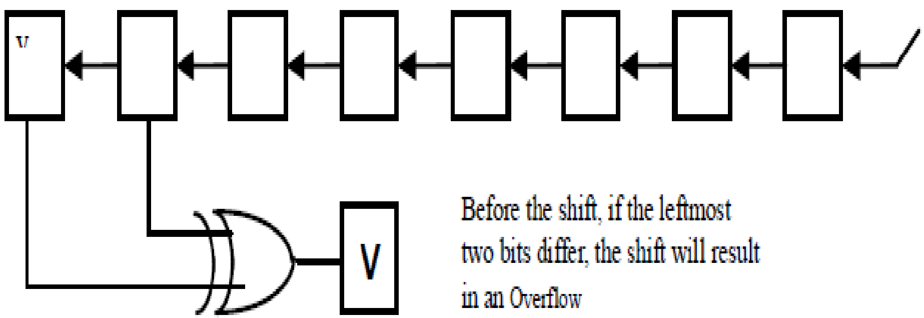
Eg: 10110100

ashl : 01101000

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 $R_{n-1} \neq R_{n-2}$.



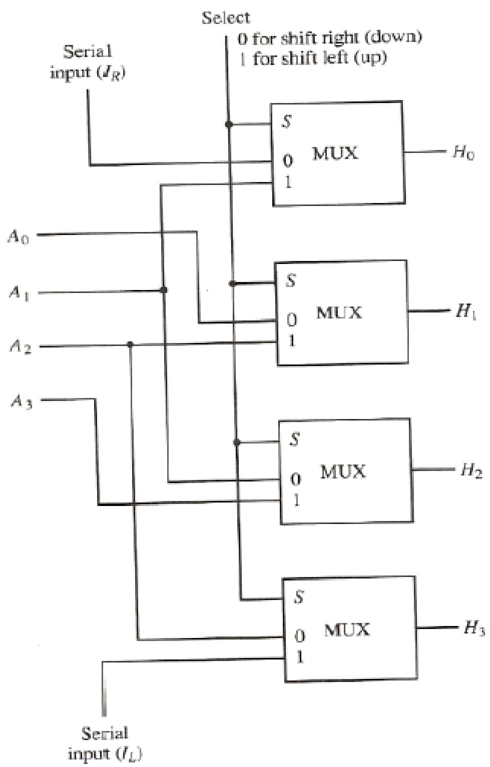
An overflow flip-flop V can be used to detect an arithmetic shift-left overflow.

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

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

Fig: 4-bit combinational circuit shifter

- 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).

Arithmetic Logic Shift Unit

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

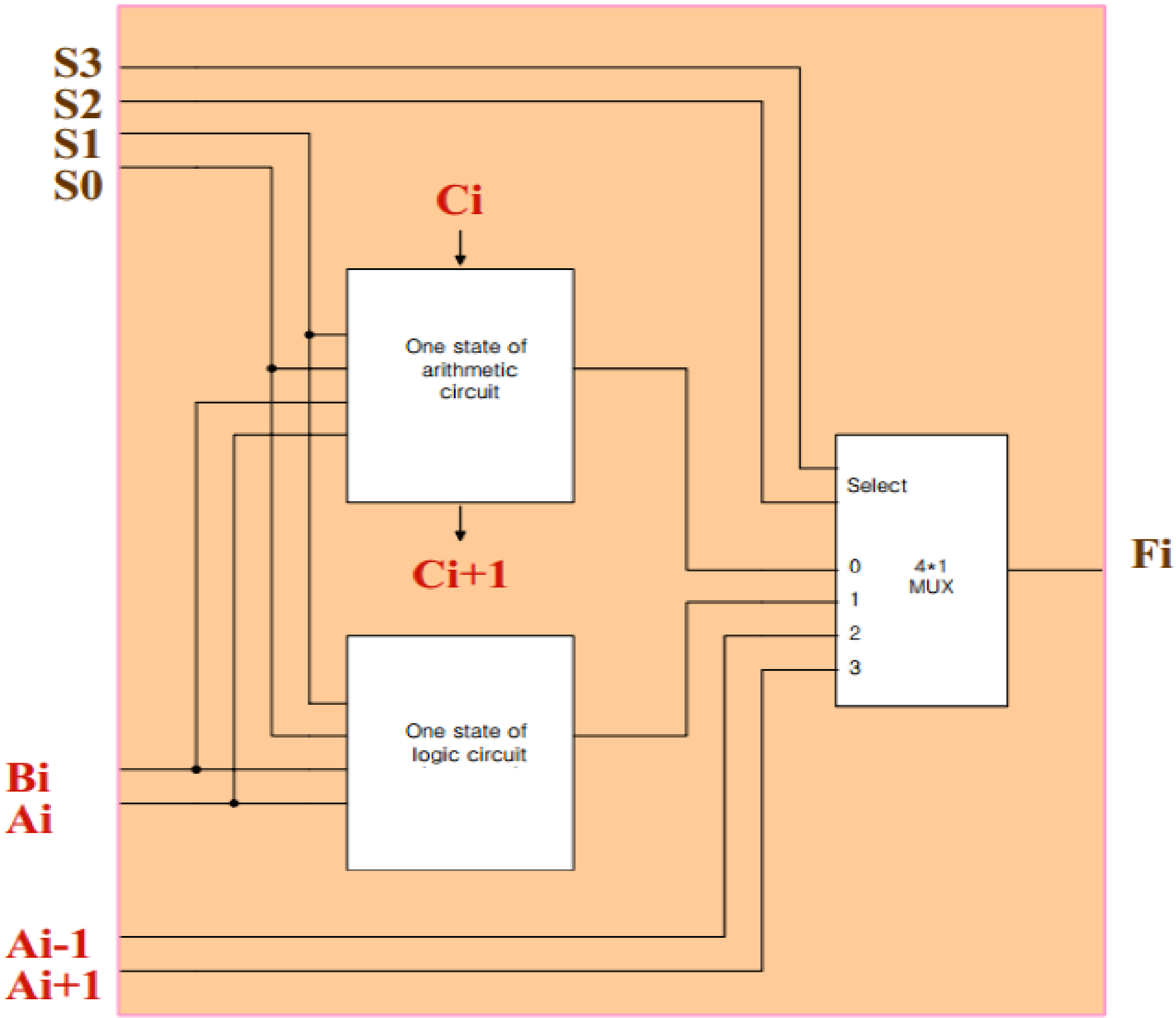


Fig: one stage of arithmetic logic shift unit

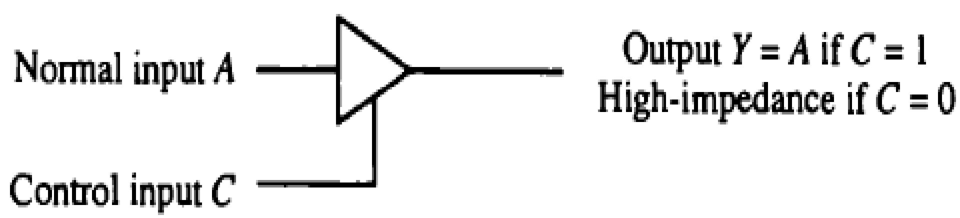
- Inputs A_i and B_i are applied to both the arithmetic and logic units. A particular microoperation is selected with inputs S_1 and S_0. A 4*1 MUX selects the final output. The two inputs of the MUX are received from the output of the arithmetic circuit and logic circuit. The other two is A_{i-1} for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

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

Three State Buffer

A three state gate is a digital circuit that exhibits three state. Two of the states are signed equivalent to logic 1 and 0 as in a conventional gate. The third state is a high impedance state. The high impedance state behaves likes an open circuit which means that the output is disconnected and does not have a logic significance.

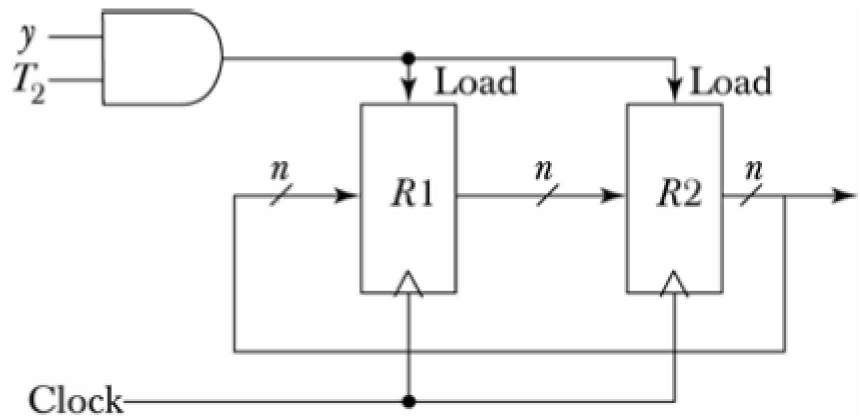
Three state gate may perform any conventional logic such as AND or NAND. However, the most commonly used in the design of bus system is the buffer gate,



Numerical

4-1. Show the block diagram of the hardware (similar to Fig. 4-2a) that implements the following register transfer statement:

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



4-3. Represent the following conditional control statement by two register transfer statements with control functions.

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

SOLUTION:

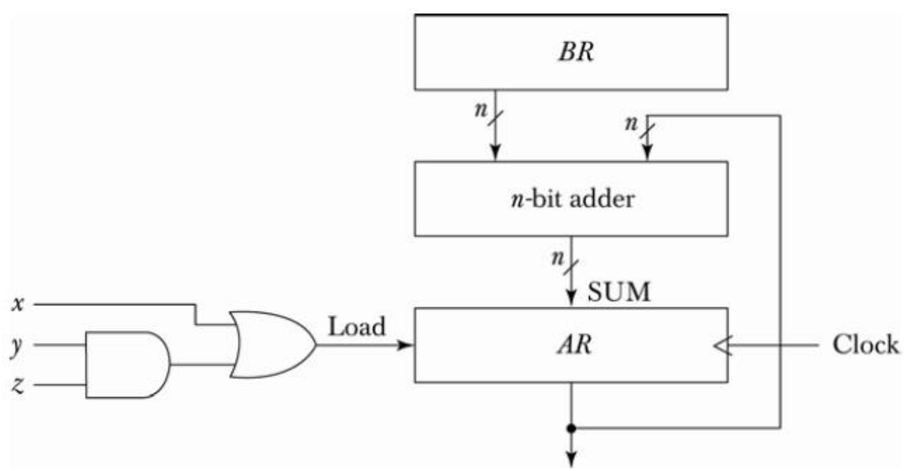
$P: R1 \leftarrow R2$

$P'Q: R1 \leftarrow R3$

4-8. Draw the block diagram for the hardware that implements the following statements:

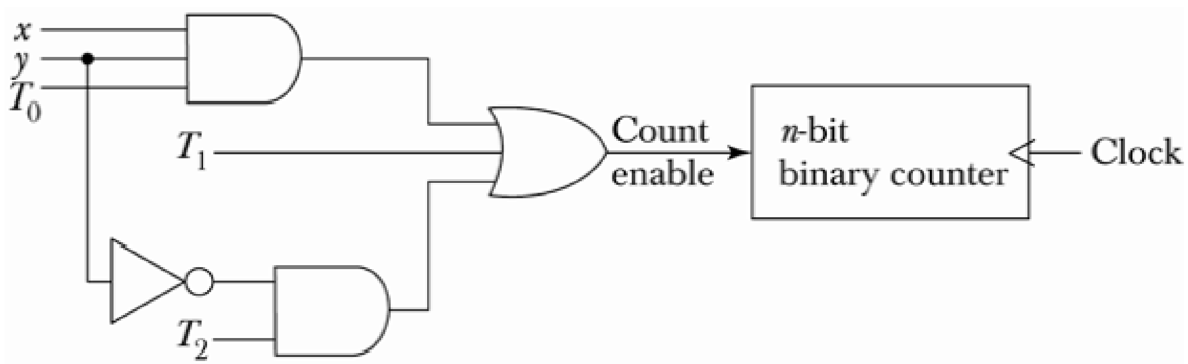
$$x + yz: \quad AR \leftarrow AR + BR$$

where AR and BR are two n-bit registers and x, y, and z are control variables. Include the logic gates for the control function. (Remember that the symbol + designates an OR operation in a control or Boolean function but that it represents an arithmetic plus in a microoperation.)



4-9. Show the hardware that implements the following statement. Include the logic gates for the control function and a block diagram for the binary counter with a count enable input.

$$xyT_0 + T_1 + y'T_2: \quad AR \leftarrow AR + 1$$



4-19. The 8-bit registers AR, BR, CR, and DR initially have the following values:

$$\begin{aligned} AR &= 11110010 \\ BR &= 11111111 \\ CR &= 10111001 \\ DR &= 11101010 \end{aligned}$$

Determine the 8-bit values in each register after the execution of the following sequence of microoperations.

$AR \leftarrow AR + BR$	Add BR to AR
$CR \leftarrow CR \wedge DR$, $BR \leftarrow BR + 1$	AND DR to CR, increment BR
$AR \leftarrow AR - CR$	Subtract CR from AR

(a) AR = 11110010
BR = <u>11111111(+)</u>
AR = 11110001 BR = 11111111 CR = 10111001 DR= 1110
1010

(b) CR = 10111001 BR = 1111 1111
<u>DR = 11101010</u>(AND) +1
CR = 10101000 BR = 0000 0000 AR = 1111 0001 DR = 11101010

(c) AR = 11110001 (−1)
CR = <u>10101000</u>
AR = 01001001; BR = 00000000; CR = 10101000; DR = 11101010

4-21. Starting from an initial value of $R = 11011101$, determine the sequence of binary values in R after a logical shift-left, followed by a circular shift-right, followed by a logical shift-right and a circular shift-left.

$R = 11011101$

Logical shift left:	10111010
Circular shift right:	01011101
Logical shift right:	00101110
Circular shift left:	01011100

4-17. Design a digital circuit that performs the four logic operations of exclusive-OR, exclusive-NOR, NOR, and NAND. Use two selection variables. Show the logic diagram of one typical stage.