Micro-operations and its types

Microoperation: Different modules are constructed from various digital components like registers, decoders, arithmetic elements, and control logic. They are interconnectedwith common data and control paths to form a digital computer system.
Microoperation is an elementary operation performed with the data stored in the registers. A microoperation is an elementary operation performed on the on the informationstored in one or more registers. The result of the operation may replace the previous binaryinformation of a register or transfer it to another register.

Types: There are basically four type of micro operations viz.

1. Register transfer microoperation: They are used to transfer binary information from one register to another
2. Arithmetic microoperations: They perform arithmetic operations on the numeric data stored in various registers.
3. Logic microoperations: They perform bit manipulation operation on nonnumeric data stored in various registers.
4. Shift microoperations: They perform shift operations on the data stored in various registers.

Specifying the Microoperation: It is possible to specify the sequence of microoperations in a computer by explaining every operation in words, but this procedure usually involves a lengthy descriptive explanation. It is more convenient to adopt a suitable symbolic representation to describe the sequence of transfers between registers and the various arithmetic and logic microoperations associated with the transfers. The symbolic notation used to describe the microoperation transfers among registers is called **Register Transfer Language (RTL)**. 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 other register.

1. **Register transfer microoperation & Register Transfer Language:** Here Computer registers are designated by capital letters (sometimes followed by numerals) to denote thefunction of the register. For example, the register that holds an address for the memory unit is called a memory address register and is designated by the name MAR. Other designations are PC (for program counter), IR (for instruction register, and R1 (for processor register).The individual flip-flops in an n-bit register are numbered in sequence from 0 through n, startingfrom 0 in the rightmost position and increasing the numbers toward the left.

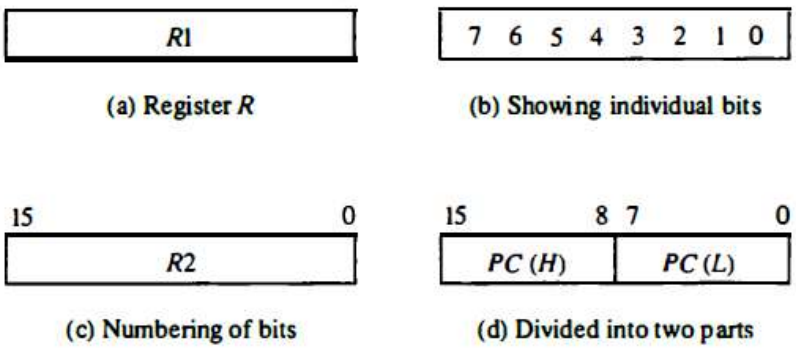
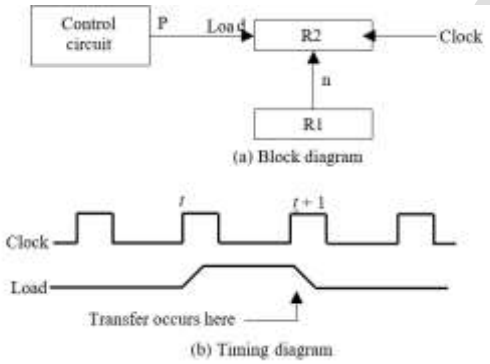


Figure 1: Representing Registers in RTL

Reference: M Morris Mano, Computer System Architecture, Third Edition, Pearson Education

Example of Register Transfer Microoperations

1. Information transfer from one register to another: $R2 \leftarrow R1$
   A statement that specifies a register transfer implies that circuits are available from the outputs of the source register to the inputs of the destination register and that the destination registerhas a parallel load capability.
2. Information transfer from one register to another based upon some preconditions:
   If (P = 1) then ($R2 \leftarrow R1$)
   where P is a control signal generated in the control section.
   It is sometimes convenient to separate the control variables from the register transfer operation by specifying a control function. A control function is a Boolean variable that is equal to 1 or 0. The control function is included in the statement as follows:-
   P: ($R2 \leftarrow R1$)
   It symbolizes the requirement that the transfer operation be executed by the hardware only if P = 1. Every statement written in a register transfer notation implies a hardware construction for implementing the transfer.



In RTL, Registers are denoted by capital letters, and numbers may follow the letters. Parentheses are used to denote a partof a register by specifying the range of bits or by giving a symbol name to a portion of a register. Thearrow denotes a transfer of information and the direction of transfer. A comma is used to separate twoor 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 pulseif T = 1. This simultaneous operation is possible with registers that have edge-triggered flip-flops.

Basic Symbols for Register Transfers

Symbol	Description	Examples
Letters (and numerals)	Denotes a Register	MAR, R2
Parenthesis ()	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

Reference: M Morris Mano, Computer System Architecture, Third Edition, Pearson Education

2. **Arithmetic Microoperation:** Basic arithmetic operations are addition, subtraction, increment, decrement and shift.

$R3 \leftarrow R1 + R2$	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

Figure 2: Various Arithmetic Microoperations (Source: Computer System Architecture by Morris Mano)

There are other types of microoperations also like multiply and divide but they are not included in basic set of microoperations. In most of the computers the multiplication operation is implemented with the sequence of add and shift microoperation whereas division implemented with a sequence of shift and subtract microoperation.
The add microoperation this is implemented using binary adder as illustrated in next figure

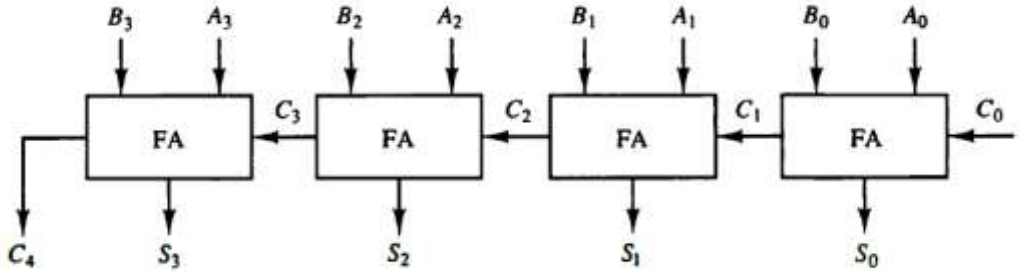


Figure 3: Implementing Add Microoperation

Implementing Add and Subtract microoperation together:

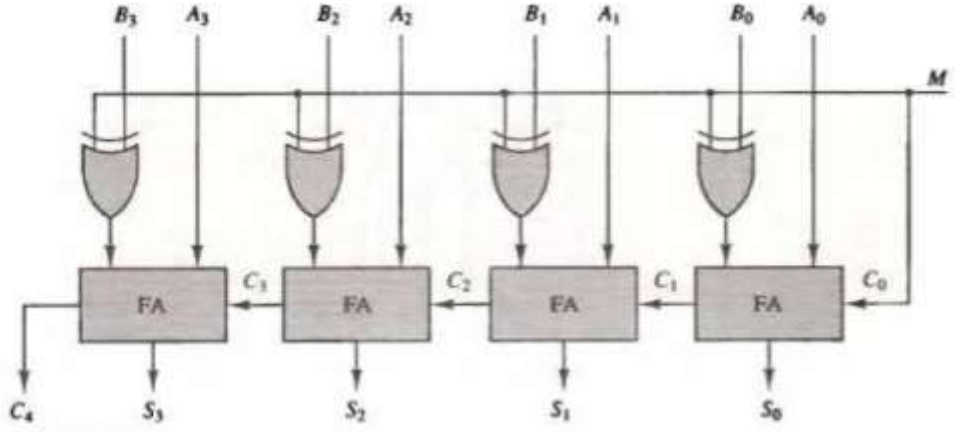


Figure 4: Integrated Add & Subtract Microoperation Implementation.

Reference: M Morris Mano, Computer System Architecture, Third Edition, Pearson Education

Binary Incrementor: This would add one to number in a register. It can be easily implemented using a binary counter. Every time the count enable is active the contents of the register will be increased by one with every clock transition.

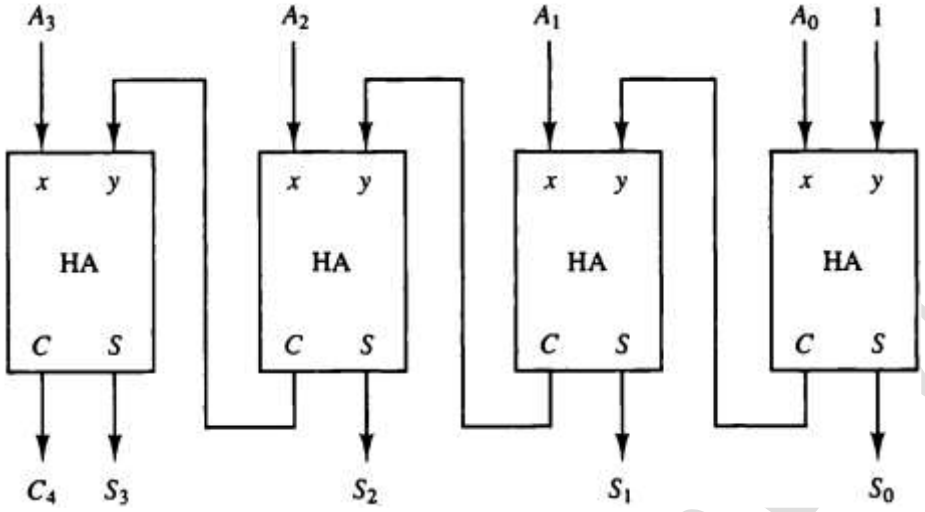


Figure 5: Binary Incrementor to implement Increment microoperation

Arithmetic Circuit:

Block diagram of arithmetic circuit to implement various arithmetic microoperation is drawn in next figure. It has four 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 and a 4-bit output D. The four inputs from A go directly to the X inputs of the binary adder. Each of the for inputs from B are 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 ad logic -1. Logic-0 is fixed voltage value (0 volts for TTL integrated circuits) and the 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. 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 binary adder. C_{in} is the input carry, which can be equal to 0 or 1. The function table for the arithmetic circuit is:

Reference: M Morris Mano, Computer System Architecture, Third Edition, Pearson Education

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

Figure 6: 4-bit Arithmetic Circuit Function Table

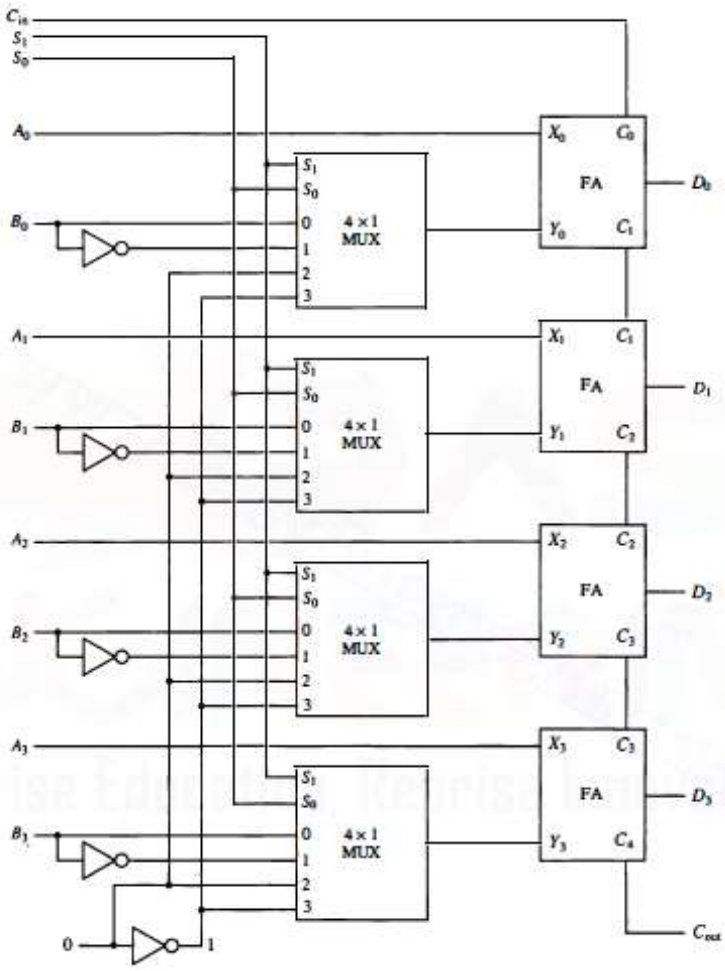


Figure 7: 4-bit Arithmetic Circuit

3. **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.

For example, the exclusive-OR microoperation with the contents of two registers R1 and R2 is symbolized by the statement

$$P: R1 \oplus R2$$

It specifies a logic microoperation to be executed on the individual bits of the registers provided that the control variable P = 1.
As a numerical example, assume that each register has four bits. Let the content of R1 be 1010 and the content of R2 be 1100. The exclusive-OR microoperation stated above symbolizes the following logic computation:

1010	Content of $R1$
1100	Content of $R2$
0110	Content of $R1$ after $P = 1$

There are 16 different logic operations that can be performed with two binary variables. They can be determined from all possible truth tables obtained with two binary variables as shown in following figure. In this each of the 16 columns F_0 through F_{15} represents a truth table of one possible Boolean function for the two variables x and y. Note that the functions are determined from the 16 binary combinations that can be assigned to F.

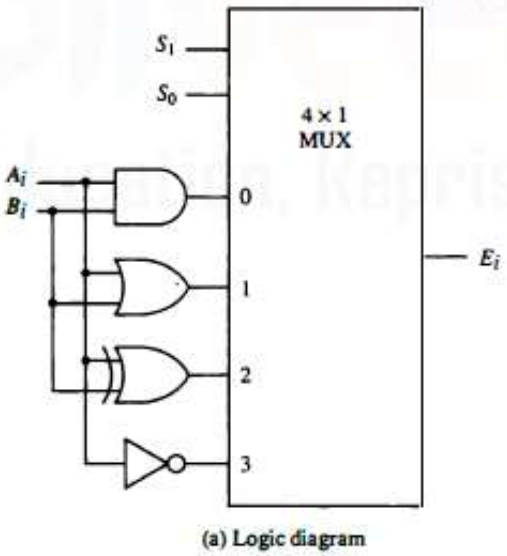
x	y	F_0	F_1	F_2	F_3	F_4	F_5	F_6	F_7	F_8	F_9	F_{10}	F_{11}	F_{12}	F_{13}	F_{14}	F_{15}
0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
0	1	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
1	0	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
1	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1

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

Figure 8: Sixteen possible Logic Microoperations

Reference: M Morris Mano, Computer System Architecture, Third Edition, Pearson Education

Block Diagram of Hardware Implementation: 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 by which all others can be derived. Following figure shows one stage of a circuit that generates the four basic logic microoperations. It consists of four gates and a multiplexer. Each of the four logic operations is generated through a gate that performs the required logic. The outputs of the gates are applied to the data inputs of the multiplexer. The two selection inputs S_1 and S_0 choose one of the data inputs of the multiplexer and direct its value to the output.



(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

Figure 9: One Stage of Four Bit Logic Microoperations Implementation

4. **Shift Microoperation:** Shift microoperations are used for serial transfer of data. They are also used with arithmetic, logic, and other data-processing operations. Various features of these microoperations are
   a. The contents of a register can be shifted to the left or the right. While the bits are shifted, the first flip-flop receives its binary information from the serial input.
   b. During a shift-left operation the serial input transfers a bit into the right most position whereas during a shift-right operation the serial input transfers a bit into the leftmost position.
   c. The information transferred through the serial input determines the type of shift. There are three types of shifts viz. logical, circular and arithmetic.
   d. A **logical shift** is one that transfers 0 through the serial input.
   e. 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.
   f. An **arithmetic shift** is a microoperation 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. The leftmost bit in a register holds the sign bit, and the remaining bits hold the number. The sign bit is 0 for positive and 1 for negative. Negative numbers are in 2's complement form.

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

Figure 10: Shift microoperations

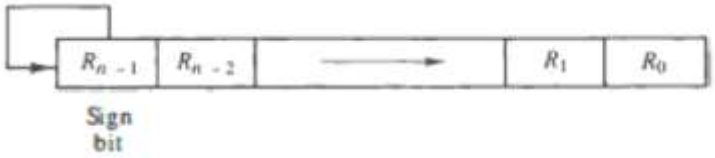
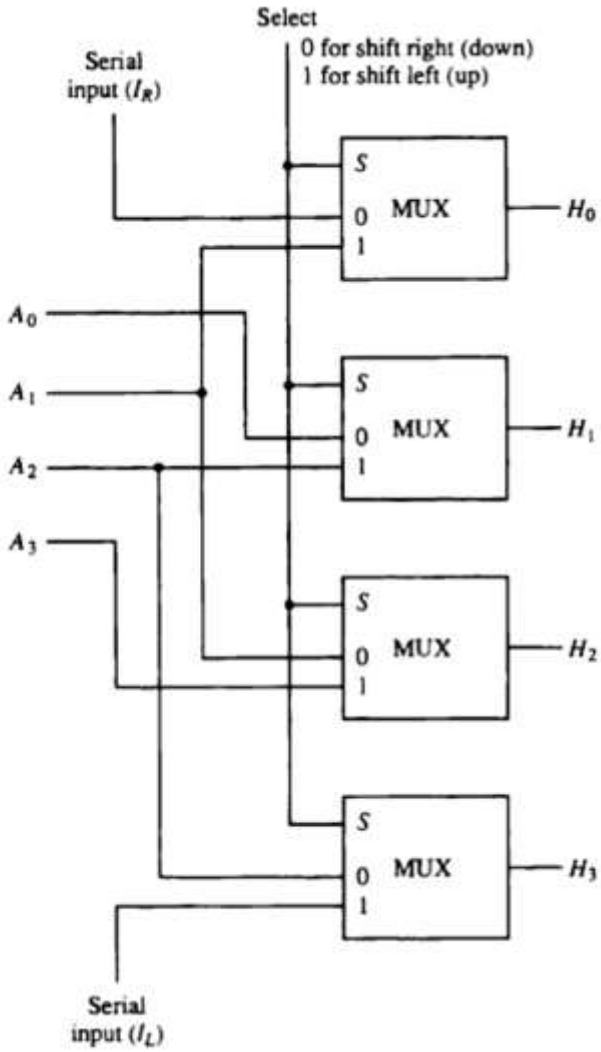


Figure 11: Arithmetic Shift Microoperation

Block Diagram of Hardware Implementation: Following figure shows implementation of four-bit circular shift microoperation using Select variable to choose from right and left shift.

Reference: M Morris Mano, Computer System Architecture, Third Edition, Pearson Education



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

Figure 12: Implementing 4-bit Circular Shift microoperations