**Chapter 4 – REGISTER TRANSFER AND MICROOPERATIONS**

**REGISTER TRANSFER LANGUAGE**

**The symbolic notation used to describe the microoperation transfers among registers .**

**A register transfer language is a system for expressing in symbolic form the microoperation sequences among the registers of a digital module.**

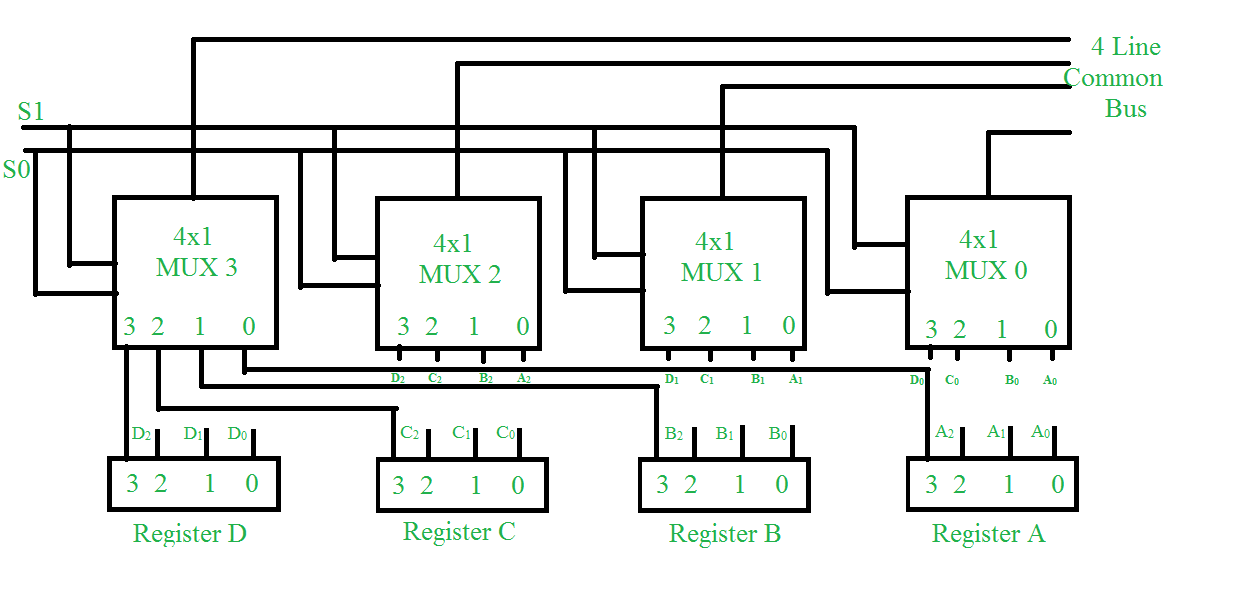
**MICROOPERATIONS : THE OPERATIONS EXECUTED ON DATA STORED IN REGISTERS.**

**MICROOPERATION : A MICROOPERATION IS AN ELEMENTARY OPERATION PERFORMED ON THE INFORMATION STORED IN ONE OR MORE REGISTERS.**

**EX : SHIFT,COUNT,CLEAR & LOAD.**

**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 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.



The construction of this bus system for 4 registers is shown above. The bus consists of 4×1 multiplexers with 4 inputs and 1 output and 4 registers with bits numbered 0 to 3. There are 2 select inputs S0 and S1 which are connected to the select inputs of the multiplexers.

The output 1 of register A is connected to input 0 of MUX 1 and similarly other connections are made as shown in the diagram. The data transferred to the bus depends upon the select lines. A function table for the various combinations of select lines is shown below.

|  |  |  |
| --- | --- | --- |
| S1 | S0 | Register selected |
| 0  0  1  1 | 0  1  0  1 | A  B  C  D |

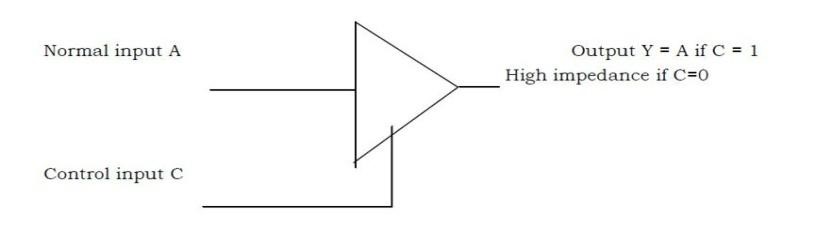
**Three-State Bus Buffers**

A bus system can be constructed with three-state gates instead of multiplexers. A three-state gate is a digital circuit that exhibits three states. Two of the states are signals equivalent to logic 1 and 0 as in a conventional gate. The third state is a high impedance state. The high impedance state behaves like an open circuit which means that the output is disconnected and does not have a logic significance. Three state gates may perform and conventional logic such As AND or NAND.

### 

### 

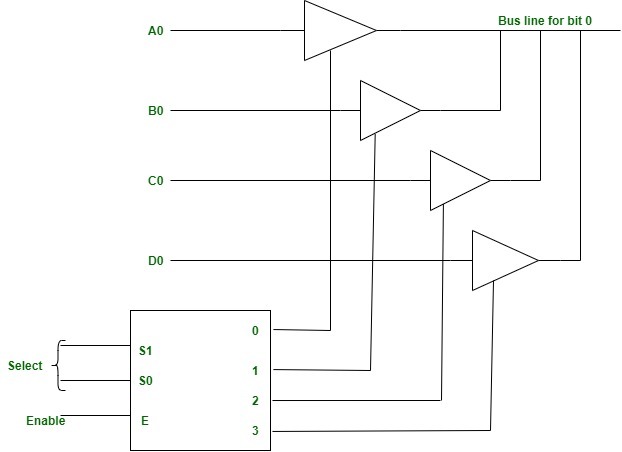
### Graphic symbols for three-state buffer

****

The buffer exhibits three states. It has 3 pins which include:  
**Input** – accepts 1 or 0 (0 – disable and 1 – enable)  
**Output** – if 3-state control is 0 then output follows input(according to the input 0 and 1).

**Definition:**  
A three state bus buffer is an integrated circuit that connects multiple data sources to a single bus. The open drivers can be selected to be either a logical high, a logical low, or high impedance which allows other buffers to drive the bus.

1. As in a conventional gate, 1 and 0 are two states.
2. Third state is a high impedance state.
3. The third state behaves like an open circuit.
4. If the output is not connected, than there is no logical significance.
5. To form a single bus line, all the outputs of the 4 buffers are connected together.
6. The control input will now decide which of the 4 normal inputs will communicate with the bus line.
7. The decoder is used to ensure that only one control input is active at a time.
8. The diagram of a 3-state buffer can be seen as below.

****

## 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.

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:

Read: DR ← M (AR)

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

The write operation transfers the content of a data register to a memory word M selected by the address. Assume that the input data are in register R1 and the address is in AR. The write operation can be stated symbolically as follows:

Write: M (AR) ← R1

This causes a transfer of information from R1 into the memory word M selected by the address in AR.

### Arithmetic Microoperations

The microoperations most often encountered in digital computer are classified into four categories:

1. Register transfer microoperations transfer binary information from one register to another.
2. Arithmetic microoperations perform arithmetic operations on numeric data stored in registers.
3. Logic microoperations perform bit manipulation operations on nonnumeric data stored in registers.
4. Shift microoperations perform shift operations on data stored in registers.

The register transfer microoperation was introduced in Sec. 2.2. This type of microoperation does not change the information content when the binary information moves from the source register to the destination register. The other three types of microoperations change the information content during the transfer.

The basic arithmetic microoperations are addition subtraction increment decrement and shift. Arithmetic shifts are explained later in conjunction with the shift microoperations. The arithmetic microoperation defined by the statement

R3 ← R1 + R2

specifies an add microoperation. It states that the contents of register R1 are added to the contents of register R2 and the sum transferred to register R3. To implement this statement with hardware we need three registers and the digital component that performs the addition operation. The other basic arithmetic microoperations are listed in Table 2-3. Subtraction is most often implemented through complementation and addition. Instead of using the minus operator, we can specify the subtraction by the following statement.

**-**

R3 ← R1 + R2 + 1

R2 is the symbol for the 1‘s complement of R2. Adding 1 to the 1‘s complement produces the 2‘s complement. Adding the contents of R1 to the 2‘s complement of R2 is equivalent to R1 – R2.

### 

### Table- Arithmetic Microoperation

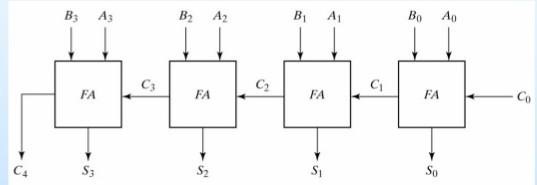
|  |  |
| --- | --- |
| Symbolic designation | Description |
| R3 ← R1 + R2  R3 ← R1 – R2  R2 ← R2  R2 ← R2 + 1  R3 ← R1 – R2 +1  R1 ← R1 + 1  R1 ← R1 – 1 | Contents of R1 plus R2 transferred to R3  Contents of R1 minus R2 transferred to R3  Complement the contents of R2 (1`s Complement)  2`s Complement the contents of R2 (negate)  R1 plus the 2`s complement of R2 (Subtraction)  Increment the contents of R1 by one  Decrement the contents of R1 by one |

The increment and decrement microoperations are symbolized by plus one and minus one operations, respectively. These microoperations are implemented with a combinational circuit or with a binary up-down counter.

## Binary Adder

The digital circuit that forms the arithmetic sum of two bits and a previous carry is called a full-adder . The digital circuit that generates the arithmetic sum of two binary numbers of any length is called a binary adder. The binary adder is constructed with full-adder circuits connected in cascade, with the output carry from one full-adder connected to the input carry of the next full-adder.

### 4 bit binary adder



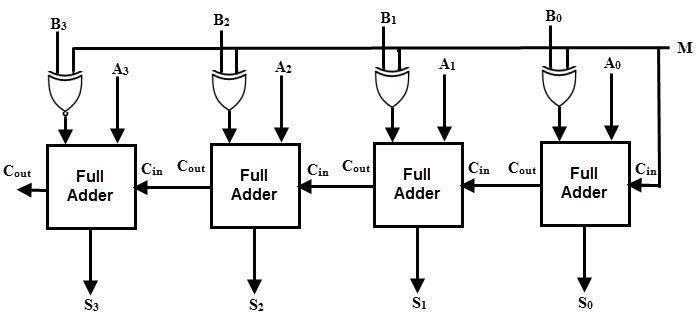
The above figure shows the interconnections of four full-adders (FA) to provide a 4-bit binary adder. The augend bits of A and the addend bits of B are designated by subscript numbers from right to left, with subscript 0 denoting the low-order bit. The carries are connected in a chain through the full-adders. The input carry to the binary adder is c0 and the output carry is C4. The S outputs of the full-adders generate the required sum bits.

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

## 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. A 4-bit adder-subtractor circuit is given below. The mode input M controls the operation. When M = 0 the circuit is 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, We have B0 =B. The full-address receive the value of B, the input carry is 0 and the circuit performs A plus B. When M=1 we have B  1 = B‘ and Co = 1. The B inputs are all complemented and a 1 is added through the input carry. The circuit performs the operation A plus the

### 4-bit adder – subtractor



2‘s complement of B. for unsigned numbers, this gives A- B if A≥ B or the 2‘ s complement of ( B-A ) if A< B. For signed numbers, the result is A- B provided that there is no overflow.

#### Logic Microoperations

Logic microoperations 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 ← R1 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

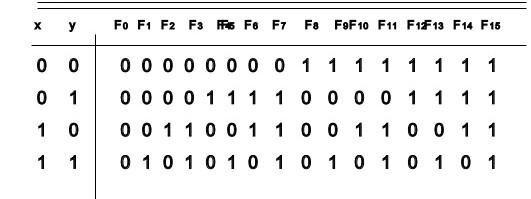
The content of R1, after the execution of the microoperation, is equal to the bit-by – bit exclusive-OR operation on pairs of bits in R2 and previous values of R1. The logic microoperations are seldom used in scientific computations, but they are very useful for bit manipulation of binary data and for making logical decisions.

Special symbols will be adopted for the logic microoperations OR, AND and complement to distinguish them from the corresponding symbols used to express Boolean functions. The symbol v will be used to denote and OR microoperation and the symbol ٨ to denote and AND microoperation. When it occurs in a control (Or Boolean) function it will denote an OR operation. We will never use it to symbolize an OR microoperation. For example in the statement.

P + Q; R1 ← R2 + R3, R4 ← R5 v R6

## List of Logic Microoperations

There are 16 different logic operations that can be performed with two binary variables. They can be determined from all possible tables obtained with two binary variables as shown in the given Table. In this table, each of the 16 columns F0 through F15 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. This Table shows the Truth Table for 16 functions of two Variables



The 16 Boolean functions of two variables x and y are expressed in algebraic from in the first column of Table 2-4. The 16 logic microoperations are derived from these functions by replacing variable x by the binary content of register A and variable y by the binary content of register B.

The logic microoperations listed in the second column represent a relationship between the binary content of two registers A and B

## 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 from which all others can be derived.

The selective-set operation sets to 1 the bits in register A where there are corresponding 1‘s in register B. It does not affect bit positions that have 0‘s in B. the following numerical example clarifies this operation:

1010 A before

1100 B (logic operand)

1110 A after

The two leftmost bits of B are 1‘s, so the corresponding bits of A are set to 1. One of these two bits was already set and the other has been changed from 0 to 1. The two bits of A with corresponding 0‘s in B remain unchanged.

The selective complement operation complements bits in A where there are corresponding 1‘a in B. It does not affect bit positions that have 0‘s in B. for example:

1010 A before

1100 B (logic oerand) 0110 A after

Again the two leftmost bits of B are 1s, so the corresponding bits of A are complemented.

One can deduce that the selective-complement operation is just an exclusive-OR microoperation. Therefore the exclusive-OR microoperation can be used to selectively complement bits of a register.

The selective-clear operation clears to 0 the bits in A only where there are corresponding 1‘s in B. for example:

1010 A before

1100 B (logic operand)

0010 A after

Again the two leftmost bits of B are 1‘s, so the corresponding bits of A are cleared to 0. One can deduce that the Boolean operation performed on the individual bits is AB‘.

Again the two leftmost bits of B are 1‘s, so the corresponding bits of A are cleared to 0. One can deduce that the Boolean operation performed on the individual bits is AB. The corresponding logic microoperation is

A← A Λ 𝐵

The mask operation is similar to the selective-clear operation except that the bits of A are cleared only where there are corresponding 0‘s. in B. The mask operation is an AND micro operation as seen from the following numerical example:

1010 A before

1100 B (logic operand)

1000 A after Masking

The two rightmost bits of A are cleared because the corresponding bits of B are 0s. The two leftmost bits are left unchanged because the corresponding bits of B are 1 s.

The insert operation that executes a new value in to a group of bits. This is done by first masking the bits and then ORing them with the required value. For example, suppose that an A register contains eight bits, 0110 1010. To replace the four leftmost bits by the value 1001 we first mask the four unwanted bits;

0110 1010 A before

0000 1111 B (logic operand)

0000 1010 A after Masking

And then insert the new value:

0000 1010 A before

1001 0000 B (insert)

1001 1010 A after insertion

The mask operation is an AND microoperation and the insert operation is and OR microoperation.

The clear operation compares the words in A and B and produces an all 0‘s result if the two numbers are equal. This operation is achieved by an exclusive-OR microoperation as shown by the following example:

1010 A

1010 B

\_\_\_\_

0000 A ← A B

When A and B are equal, the two corresponding bits are either both 0 or both 1. In either case the exclusive-OR operation produces a 0. The all – 0`s result is then checked to determine if the two numbers were equal.

### Shift Microoperations

Shift microoperations are used for serial transfer of data. They are also used in conjunction with arithmetic, logic and other data-processing operations.

A logical shift is one that transfers 0 through the serial input. We will adopt the symbols and shr for logical shift-left and shift-right microoperations. For example:

R1←shI R1

R2 ←shr R2

10101001

01010010 – Shift Left

10101001

01010100 – Shift Right

are two microoperations that specify a 1-bit shift to the left of the content of register R1 and 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. The bit transferred to the end position through the serial input is assumed to be 0 during a logical shift.

The circular shift (also known 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.

Cil – 1010 1001

Shift left – 0101 0011

Cir – 1010 1001

Shift right – 1101 0100

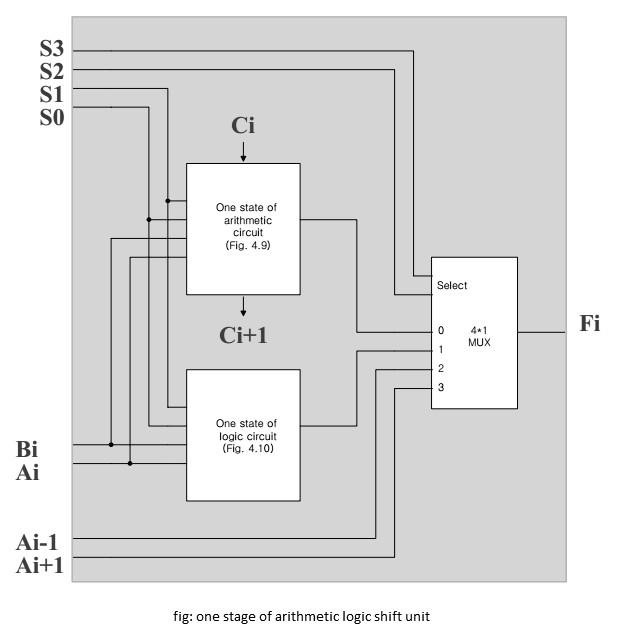
An arithmetic shift is a microoperation that shifts a signed binary number to the left or right. An arithmetic shift-left multiplies a signed binary number by 2. An arithmetic shift right divides the number by 2. Arithmetic shifts must have the sign bit unchanged because the sign of the number remains 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.

### Arithmetic Logic shift Unit

Instead of having individual registers performing the microoperations directly computer systems employ a number of storage registers connected to a common operational unit called an arithmetic logic unit abbreviated ALU. To perform a microoperation, the contents of specified registers are placed in the inputs of the common ALU. The ALU performs an operation and the result of the operation is then transferred to a destination register. The ALU is a combinational circuit so that the entire register transfer operation from the source registers through the ALU and into the destination register can be performed during one clock pulse period. The shift microoperations are often performed in a separate unit but sometimes the shift unit is made part of the overall ALU.

One stage of an arithmetic logic shift unit is shown in the given figure. The subscript I designates a typical stage. Inputs Ai and Bi are applied to both the arithmetic and logic units. A particular microoperation is selected with inputs S1 and S0 . A 4 x 1 multiplexer at the output chooses between an arithmetic output in Ei and a logic output in Hi. The data in the multiplexer are selected with inputs S3 and S2. The other two data inputs to the multiplexer receive inputs Ai-1 for the shift-right operation and Ai+1for the shift-left operation. The diagram shows just one typical stage. The output carry Ci+1  of a given arithmetic stage must be connected to the input carry Ci of the next stage in sequence.

### One stage of arithmetic logic shift unit



The input carry to the first stage is the input carry Cin which provides a selection variable for the arithmetic operations.

The circuit whose one stage is specified in the above figure, provides eight arithmetic operation four logic operations and two shift operations. Each operation is selected with the five variables S3, S2, S1, S0 and Cin. The input carry Cinis used for selecting an arithmetic operation only.