Computer Architecture ECS-355

Mr. Abdul Rehman **email id: abdul@northern.edu.pk** **Whatsapp# 0308-7792217**

(Week 3) Lecture 5-6

Learning objectives:

- Understanding of Register Transfer & micro-operations
- Register transfer
- Bus and memory transfer
- Arithmetic microoperations
- Logic microoperations
- Shift microoperations

(Note: It is important to note that we are considering "Basic computer architecture" processor which we were discussing in the class and it's the continuation of the same concepts).

Resources: Beside these lecture handouts, this lesson will draw from the following
Text Book: Computer System Architecture by Morris Mano (3rd Edition) and
Reference book:Computer Architecture, by William Stallings (4th Edition).

Lecture:

REGISTER TRANSFER AND MICROOPERATIONS

It focuses on the fundamental concepts related to how data is transferred between registers and the various microoperations that can be performed within a computer system.

1. **Register Transfer**

The opertions executed on data stoed in registers are called **microoperations**. It is an elementary operation performed on the information stored in one or more registers. It is possible to specify the sequence of microoperations in a computer by explaining every operation in words, but this procedure usually involve a lengthy descriptive explaination. The symbolic notation used to describe the microoperation transfers among registes is called Register transfer language.

Computer registers are labeled with capital letters (and sometimes numbers) to show what each register does. For example, the register that stores a memory address is called the Memory Address

Register (MAR). Other examples include the Program Counter (PC), Instruction Register (IR), and Processor Register (RI).

Each register is made up of small storage units called flip-flops. If the register has n bits, these flip-flops are numbered starting from 0 (on the right) to n - 1 (on the left). Figure 4-1 below shows the representation of registers in block diagramn form.

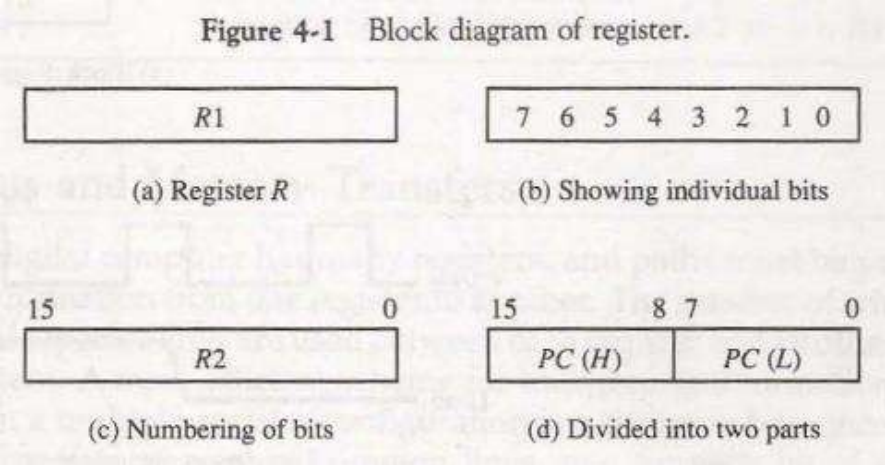


Figure 4-1 Block diagram of register.

The most common way to represent a register is by a rectangular box with the name of the register inside, as in Fig. 4-l(a). The individual bits can be distinguished as in (b). For example, in a 16-bit register, the bits can be numbered above the box, from 0 to 15 (shown in part (c)). In part (d), the 16-bit register is divided into two parts: the low byte (bits 0-7, labeled "L") and the high byte (bits 8-15, labeled "H"). (for low byte) and bits & through 15 are assigned the symbol H(for high byte). The name of the 16-bit register is PC. The symbol PC0-7) or PC(L) refers to the low-order byte and PC(8-15) or PC(H) to the high-order byte.

The transfer from one register to another is written as:

$$R2 \leftarrow R1$$

It denotes a transfer of the content of register RI into register R2. It designates a replacement of the content of R2 by the content of RI. By definition, the content of the source register Rl does not change after the transfer.

Transfers usually happen under specific conditions controlled by signals. For example:

$$\text{If } (P = 1) \text{ then } (R2 \leftarrow R1)$$

Here, P is a control signal, and the transfer occurs only if P is 1 (active). A control function is a simple on/off signal (either 1 or 0) that tells the system when to perform the transfer. This can also be written as:

$$P: R2 \leftarrow R1$$

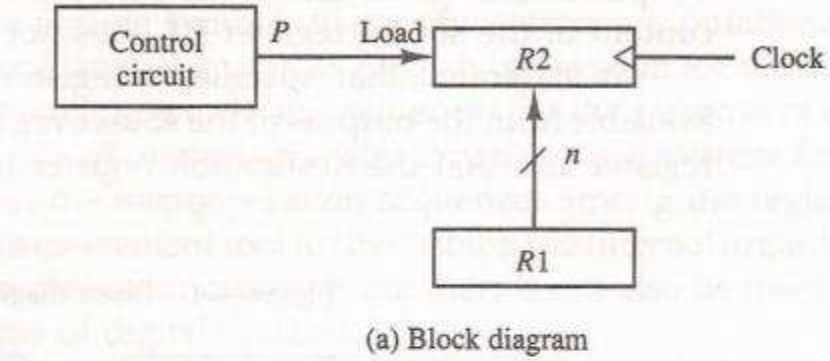
Meaning, if P is 1, then the transfer happens.

Every time a register transfer statement is written, it implies that there is hardware built to make that transfer happen. Figure 4-2 shows how data is transferred from R1 to R2. The number of bits in the registers is indicated by n, and this will be replaced by the actual number once the register size is known.

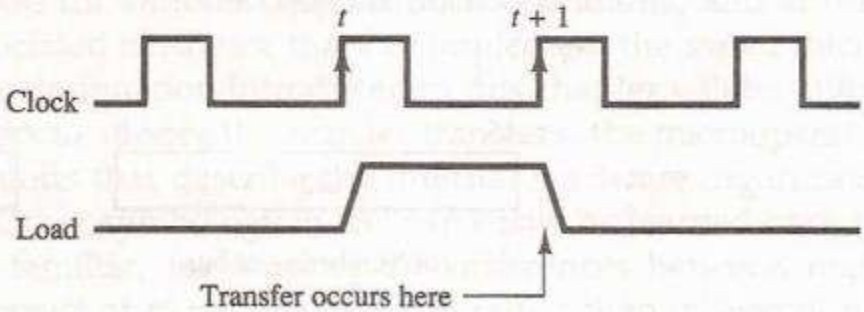
R2 has a load input, which is controlled by a signal P. This signal works in sync with the clock.

Mr. Abdul Rehman **email id: abdul@northern.edu.pk** **Whatsapp# 0308-7792217**

Figure 4-2 Transfer from R1 to R2 when P = 1.



(a) Block diagram



(b) Timing diagram

According to the timing diagram, P gets activated by the rising edge of a clock pulse at time t. The actual transfer into R2 happens on the next clock pulse at t+1 when the load input is active, allowing data to move from R1 to R2 in parallel (all bits at once).If P goes back to 0 after t+1, the transfer stops. But if P stays active, the transfer will continue with each clock pulse.Although the clock is not mentioned directly in the register transfer statement, it's assumed that all ransfers happen at a clock edge. Even if P becomes active right after time t, the transfer won't occur until the next clock pulse at t+1.

2. Bus and Memory Transfer

Computer Architecture ECS-355

Mr. Abdul Rehman **email id: abdul@northern.edu.pk** **Whatsapp# 0308-7792217**

A typical digital computer has many registers, and there needs to be a way to transfer data between them. If each register were connected to all the others using separate wires, it would require too many wires. A more efficient method is to use a common bus system. A bus system is a set of shared lines, with one line for each bit of data, used to transfer information between registers one at a time. Control signals decide which register sends data to the bus during each transfer.

One way to build a bus system is with multiplexers (MUX). Multiplexers choose which register's data is placed on the bus. The figure (Fig. 4-3) shows a bus system for four registers, each with 4 bits (labeled 0 to 3). There are four multiplexers, each with four data inputs and two selection inputs. Instead of showing 16 crossing wires, labels are used to show connections. For example, the first output of register A connects to the first input of multiplexer MUX 0.

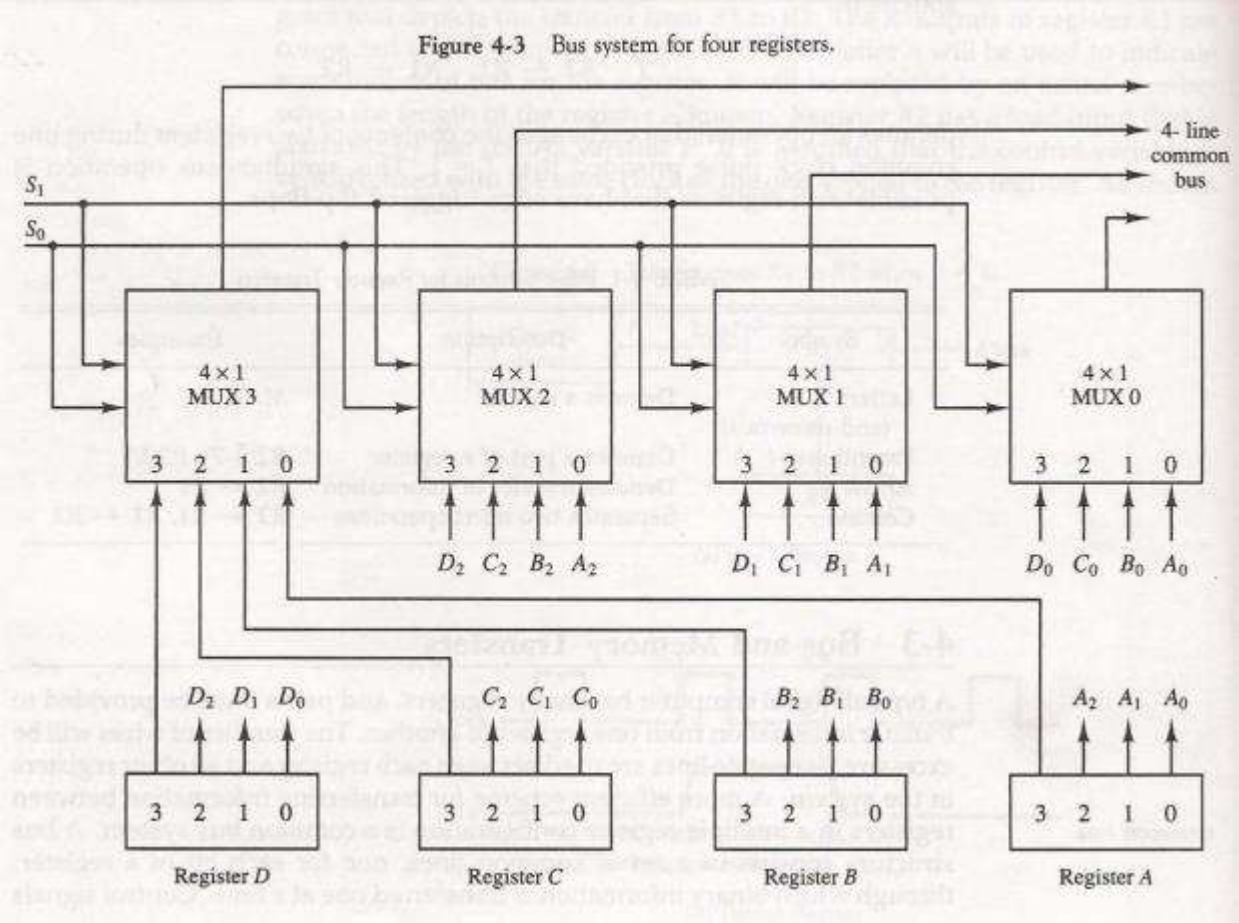


Figure 4-3 Bus system for four registers.

Each multiplexer handles the same bit position across all registers. MUX 0 handles the first bit (bit 0) of all four registers. MUX 1 handles the second bit (bit 1), and so on for the other bits. This way, data from any register can be selected and sent through the bus by the multiplexers.

The two selection lines, S_1 and S_0, control which register's data is transferred to the common bus. For example, when $S_1S_0 = 00$, the data from register A is sent to the bus. Similarly, other registers like B are selected based on the values of S_1S_0. Table 4-2 shows the register that is selected by the bus for each of the four possible binary value of the selection lines.

TABLE 4-2 Function Table for Bus of Fig. 4-3

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

A bus system uses multiplexers to select and transfer data from multiple registers. For k registers with n bits, n multiplexers are needed, each having k data inputs. 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. The symbolic statement for a bus transfer may mention the bus or its presence may be implied in the statement.The symbolic statement for a bus transfer may mention the bus or its presence may be implied in the statement. When the bus is incudes in the statement, the register transfer is symbolized as follows:

$$\text{BUS} \leftarrow \text{C}, \qquad \text{R1} \leftarrow \text{BUS}$$

The content of register C is placed on the bus, and the bus content is loaded into register R1 by activating its load control input, which can be simplified to showing the direct transfer as:

$$\text{R1} \leftarrow \text{C}$$

Three-State Bus Buffers

A bus system can be built using **three-state gates** instead of multiplexers. A three-state gate has three output states: **logic 1**, **logic 0**, and a **high-impedance (disconnected) state**. The high-impedance state acts like an open circuit, meaning the output is disconnected and does not have a

logic significance. Three-state gates may perform any conventional logic such as AND or NAND. However, the most common three-state gate used in bus systems is the **buffer gate**.

The graphic symbol of a three-state buffer gate is shown in Fig. 4-4.

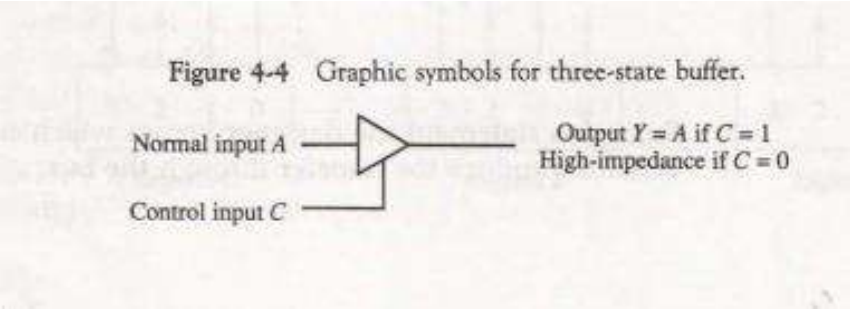


Figure 4-4 Graphic symbols for three-state buffer.

The three-state buffer is distinguished from a normal buffer by having both a normal input and a control input. The control input determines the output state. When the control input is 1, the buffer behaves normally, passing the input to the output. When the control input is 0, the output goes to high impedance, effectively disconnecting from the bus. This allows multiple buffers to connect to the same bus line without interference.

The construction of a bus system with three-state buffers is demonstrated in Fig. 4-5. In a bus system using three-state buffers, the outputs of multiple buffers (e.g., four buffers) are connected to a single bus line. Only one buffer can be active at a time, while others remain in a high-impedance (disconnected) state. This prevents interference between multiple buffers.

To ensure only one buffer is active, a decoder is used. When the decoder's enable input is 0, all buffers are disabled, and the bus remains in a high-impedance state. When the enable input is 1, the decoder selects one buffer based on its inputs to communicate with the bus.

To create a bus for four registers with n bits each, you need n sets of four buffers, where each set handles a specific bit from the registers. A single decoder controls which register is selected to communicate with the bus.

Mr. Abdul Rehman **email id: abdul@northern.edu.pk** **Whatsapp# 0308-7792217**

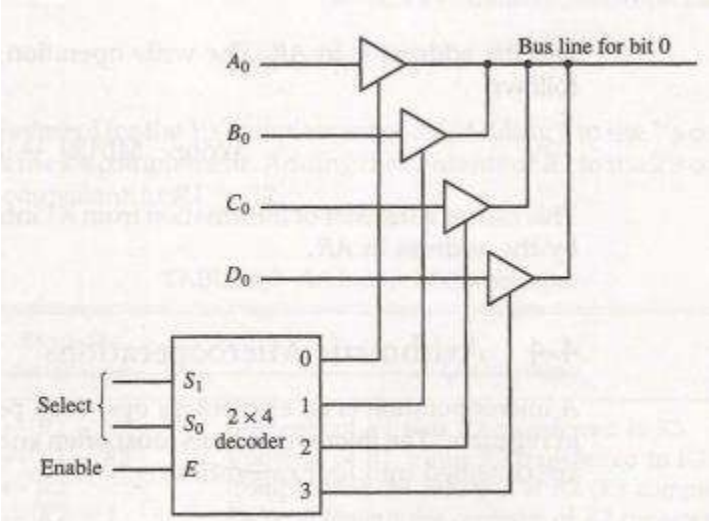


Figure 4-5 Bus line with three state-buffers.

Memory Transfer

The memory unit performs two main operations: read and write. In a read operation, data is transferred from a specific memory word (M) to a register. The memory word is chosen by its address (stored in the address register, AR). The read operation can be written as:

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

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

$$\text{Write:} \quad M[AR] \leftarrow R1$$

3. Arithmetic Microoperations

The basic arithmetic microoperations are addition, subtraction, increment, decrement, and shift. The arithmetic microoperation defined by the statement specifies an add operation.

$$R3 \leftarrow R1 + R2$$

Some of the other arithmetic microoperations are listed in Table 4-3.

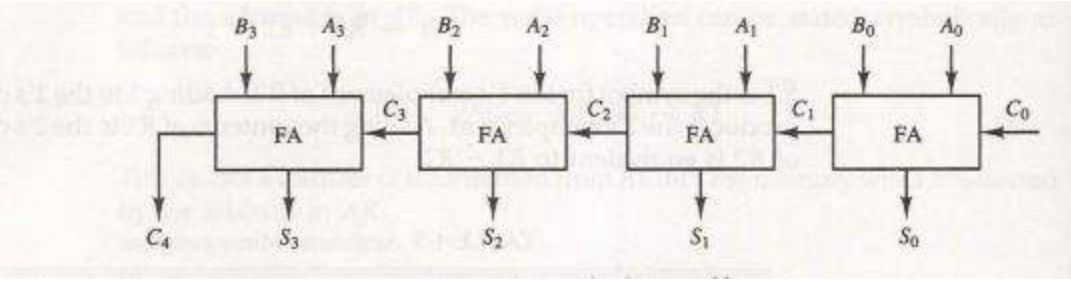
TABLE 4-3 Arithmetic Microoperations

Symbolic designation	Description
$R3 \leftarrow R1 + R2$	Contents of $R1$ plus $R2$ transferred to $R3$
$R3 \leftarrow R1 - R2$	Contents of $R1$ minus $R2$ transferred to $R3$
$R2 \leftarrow \overline{R2}$	Complement the contents of $R2$ (1's complement)
$R2 \leftarrow \overline{R2} + 1$	2's complement the contents of $R2$ (negate)
$R3 \leftarrow R1 + \overline{R2} + 1$	$R1$ plus the 2's complement of $R2$ (subtraction)
$R1 \leftarrow R1 + 1$	Increment the contents of $R1$ by one
$R1 \leftarrow R1 - 1$	Decrement the contents of $R1$ by one

Binary Adder

To perform an addition operation in hardware, we need registers to hold data and a digital component for the arithmetic process. The basic circuit for adding two bits and a carry is called a full-adder. A binary adder adds two binary numbers and is built by connecting full-adders in a chain.

For example, a 4-bit binary adder has four full-adders connected, where the carry output of each adder is linked to the carry input of the next one. The data bits (A and B) come from two registers (like R1 and R2), and the sum can be stored in another register or overwrite one of the source registers. An n-bit binary adder requires n full-adders.



Binary Adder-Subtractor

Mr. Abdul Rehman **email id: abdul@northern.edu.pk** **Whatsapp# 0308-7792217**

Binary subtraction can be done easily using 2's complement. To subtract A - B, we take the 2's complement of B and add it to A. The 2's complement is found by taking the 1's complement (inverting the bits) and adding 1 to the result.

An adder-subtractor circuit combines addition and subtraction. It uses exclusive-OR gates along with full adders. The mode input (M) controls the operation:

- When M = 0, the circuit performs addition (A + B).
- When M = 1, it complements B and performs subtraction (A - B) by adding A to the 2's complement of B.

For unsigned numbers, the result will be A - B (if A > B) or the 2's complement of (B - A) (if A < B).

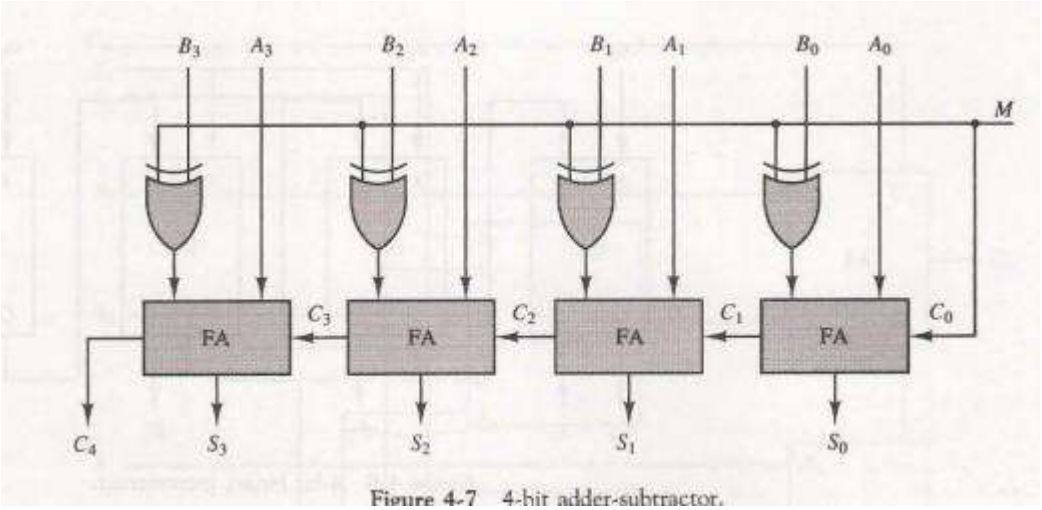


Figure 4-7 4-bit adder-subtractor.

Binary Incrementer

A binary incrementer adds 1 to a number stored in a register. For example, a 4-bit number 0110 becomes 0111 after being incremented. This can be done using a binary counter, which increases the value by 1 every time a clock pulse occurs. Sometimes, you need to increment a number without using a specific register. In that case, a combinational circuit made of half-adders can be used. A 4-bit binary incrementer uses a series of half-adders where the least significant bit (LSB) is connected to logic-1. As the bits are incremented, the carry from each half-adder moves to the

next higher bit. This circuit can be expanded to handle any number of bits by adding more half-adders.

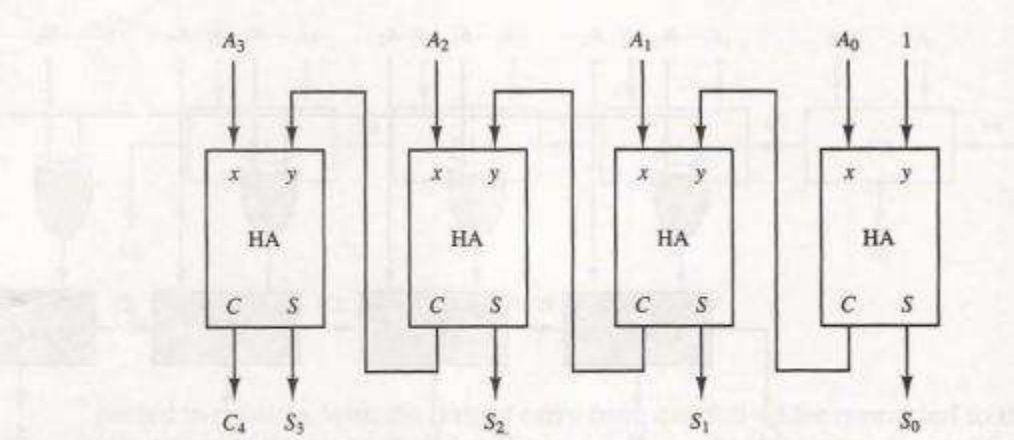


Figure 4-8 4-bit binary incrementer.

Arithmetic Circuit

An arithmetic circuit can perform different arithmetic operations using a single setup. The main part of this circuit is the parallel adder, which adds binary numbers. By controlling the inputs to the adder, various operations can be done.

In a 4-bit arithmetic circuit:

- There are two 4-bit inputs, A and B, and a 4-bit output, D.
- The inputs from A go directly into the adder, while B goes through multiplexers, which can select B, B's complement, 0, or 1.
- The adder combines **A**, a controlled version of **B**, and a carry input (**C_in**) to produce the output.

Computer Architecture ECS-355

Mr. Abdul Rehman **email id: abdul@northern.edu.pk** **Whatsapp# 0308-7792217**

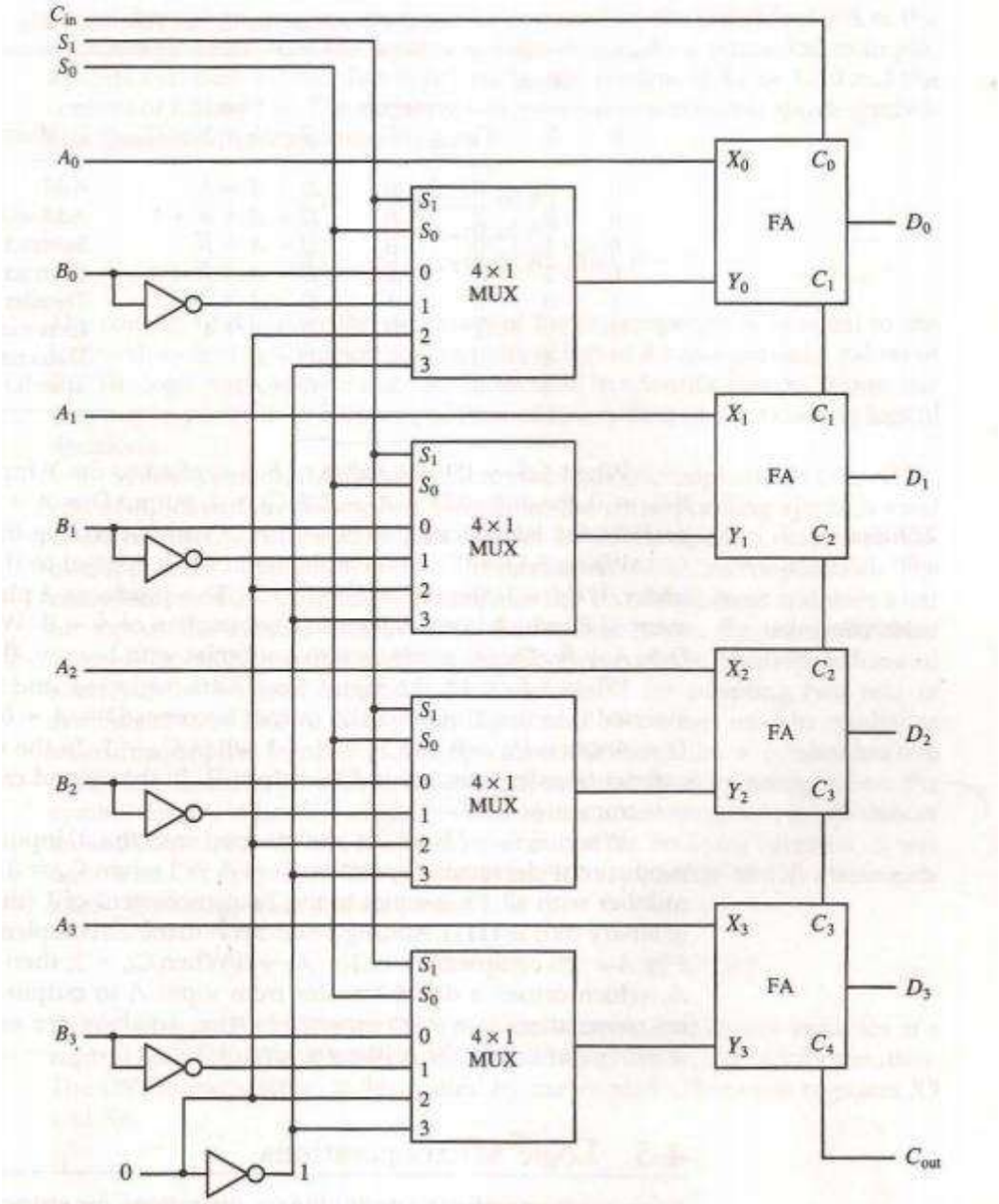


Figure 4-9 4-bit arithmetic circuit.

The final output is calculated using the formula:

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

Where A is the 4-bit binary number at the X inputs andY is the 4-bit binary number at the Y inputs of the binary adder. C is the input carry, which can be equal to 0 or 1. By adjusting the control inputs and carry, the circuit can perform eight different arithmetic operations. Listed in the given table.

TABLE 4-4 Arithmetic Circuit Function Table

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

4. 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

$$P: \ R1 \leftarrow R1 \oplus R2$$

It specifies a logic microoperation to be executed on the individuai 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 RI after P= 1

Mr. Abdul Rehman **email id: abdul@northern.edu.pk** **Whatsapp# 0308-7792217**

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.

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 truth tables obtained with two binary variables as shown in Table 4-5. 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 variables x and y.

TABLE 4-5 Truth Tables for 16 Functions of Two 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

The 16 Boolean functions are expressed in algebraic form in the first column of Table 4-6. 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. It is important to realize that the Boolean functions listed in the first column of Table 4-6 represent a relationship between two binary variables x and y. The logic microoperations listed in the second column represent a relationship between the binary content of two registers A and B. Each bit of the register is treated as a binary variable and the microoperation is performed on the string of bits stored in the registers.

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

5. Shift microoperations

Shift microoperations are used for serial transfer of data and are also applied with arithmetic, logic, and other data operations. Data in a register can be shifted either to the left or right. When bits are shifted, the first flip-flop receives a new bit from the serial input. In a shift-left operation, a bit is transferred into the rightmost position, and in a shift-right operation, a bit is transferred into the leftmost position. The type of shift depends on the serial input. There are three types of shifts: **logical**, **circular**, and **arithmetic**. A **logical shift** is one that transfers 0 through the serial input. We will adopt the symbols shl and shr for logical shift-left and shift-right microoperations.

$$R1 \leftarrow shl\ R1$$

$$R2 \leftarrow shr\ R2$$

are two microoperations that specify a 1-bit shift to the left of the content of register RI and a 1-bit shift to the right of the content of register R2. The reigister 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 .

Mr. Abdul Rehman **email id: abdul@northern.edu.pk** **Whatsapp# 0308-7792217**

In a **circular shift** (or rotate), bits are rotated around the register without losing any information. The serial output is connected to the serial input, and we use the symbols cil and cir for circular left and right shifts. The symbolic notation for these shift microoperations is provided in Table 4-7.

TABLE 4-7 Shift Microoperations

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

An **arithmetic shift** is a microoperation that shifts a signed binary number either left or right. An arithmetic shift-left multiplies the number by 2, while an arithmetic shift-right divides it by 2. The sign bit, which determines whether the number is positive or negative, must remain unchanged during the shift since the sign doesn't change with multiplication or division by 2.

In Figure 4-11, the leftmost bit (R_{n-1}) is the sign bit, while the rest of the bits hold the actual number.

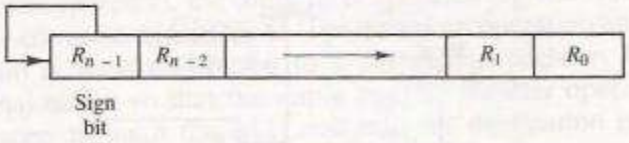


Figure 4-11 Arithmetic shift right.

For arithmetic shift-right, the sign bit stays the same, and the rest of the bits are shifted to the right, with the rightmost bit being lost. For arithmetic shift-left, a 0 is inserted into the least significant bit (R_0), and the other bits are shifted left. If the sign bit changes after the shift, it means there's an overflow. An overflow occurs when R_{n-1} is different from R_{n-2} before the shift.

An overflow flip-flop V can detect this condition:

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

If $V_s = 1$, it indicates an overflow and a possible sign change. If $V_s = 0$, there's no overflow. The value of V_s is transferred to the overflow flip-flop during the shift.

.