UNIT-I
Data Representation

Binary Number System

We have already mentioned that computer can handle with two types of signals, therefore, to represent any information in computer, we have to take help of these two signals. These two signals corresponds to two levels of electrical signals, and symbolically we represent them as 0 and 1. In our day to day activities for arithmetic, we use the Decimal Number System. The decimal number system is said to be of base, or radix 10, because it uses ten digits and the coefficients are multiplied by power of 10. A decimal number such as 5273 represents a quantity equal to 5 thousands plus 2 hundreds, plus 7 tens, plus 3 units. The thousands, hundreds, etc. are powers of 10 implied by the position of the coefficients. To be more precise, 5273 should be written as:

$$5\times10^3 + 2\times10^2 + 7\times10^1 + 3\times10^0$$

However, the convention is to write only the coefficient and from their position deduce the necessary power of 10. In decimal number system, we need 10 different symbols. But in computer we have provision to represent only two symbols. So directly we cannot use decimal number system in computer arithmetic.

For computer arithmetic we use binary number system. The binary number system uses two symbols to represent the number and these two symbols are 0 and 1. The binary number system is said to be of base 2 or radix 2, because it uses two digits and the coefficients are multiplied by power of 2. The binary number 110011 represents the quantity equal to:

$$1\times2^5 + 1\times2^4 + 0\times2^3 + 0\times2^2 + 1\times2^1 + 1\times2^0 = 51 \text{ (in decimal)}$$

We can use binary number system for computer arithmetic.

Representation of Unsigned Integers

Any integer can be stored in computer in binary form. As for example: The binary equivalent of integer 107 is 1101011, so 1101011 are stored to represent 107. What is the size of Integer that can be stored in a Computer? It depends on the word size of the Computer. If we are working with

8-bit computer, then we can use only 8 bits to represent the number. The eight bit computer means the storage organization for data is 8 bits. In case of 8-bit numbers, the minimum number that can be stored in computer is 00000000 (0) and maximum number is 11111111 (255) (if we are working with natural numbers).

So, the domain of number is restricted by the storage capacity of the computer. Also it is related to number system; above range is for natural numbers. In general, for n-bit number, the range for natural number is from 0 to 2^n-1. Any arithmetic operation can be performed with the help of binary number system.

Signed Integer

We know that for n-bit number, the range for natural number is from 0 to $2^n - 1$.
For n-bit, we have all together 2^n different combination, and we use this different combination to represent 2^n numbers, which ranges from 0 to $2^n - 1$.
If we want to include the negative number, naturally, the range will decrease. Half of the combinations are used for positive number and other half is used for negative number.
For n-bit representation, the range is from $-2^n - 1$ to $+2^n - 1$.
For example, if we consider 8-bit number, then range for natural number is from 0 to 255; but for signed integer the range is from -127 to +127.

Representation of signed integer

We know that for n-bit number, the range for natural number is from $2^n - 1$. There are three different schemes to represent negative number:

- Signed-Magnitude form.
- 1's complement form.
- 2's complement form.

Signed magnitude form:
In signed-magnitude form, one particular bit is used to indicate the sign of the number, whether it is a positive number or a negative number. Other bits are used to represent the magnitude of the number.
For an n-bit number, one bit is used to indicate the signed information and remaining (n-1) bits are used to represent the magnitude. Therefore, the range is from $-2^n - 1$ to $+2^n - 1$.
Generally, Most Significant Bit (MSB) is used to indicate the sign and it is termed assigned bit. 0 in signed bit indicates positive number and 1 in signed bit indicates negative number.

For example, 01011001 represents + 169 and 11011001 represents - 169

The concept of complement

The concept of complements is used to represent signed number.
Consider a number system of base-r or radix-r. There are two types of complements,

- The radix complement or the r's complement.
- The diminished radix complement or the (r - 1)'s complement.

Representation of Signed integer in 1's complement form:

Consider the eight bit number 01011100, 1's complements of this number is 10100011. If we perform the following addition:
If we add 1 to the number, the result is 100000000. 0 1 0
1 1 1 0 0

1 0 1 0 0 0 1 1
---
1 1 1 1 1 1 1 1
Since we are considering an eight bit number, so the 9th bit (MSB) of the result cannot be stored. Therefore, the final result is 00000000. Since the addition of two numbers is 0, so one can be treated as the negative of the other number. So, 1's complement can be used to represent negative number.

Consider the eight bit number 01011100, 2's complements of this number is 10100100. If we perform the following addition:

0 1 0 1 1 1 0 0
1 0 1 0 0 0 1 1
---
1 0 0 0 0 0 0 0 0

Since we are considering an eight bit number, so the 9th bit (MSB) of the result cannot be stored. Therefore, the final result is 00000000. Since the addition of two numbers is 0, so one can be treated as the negative of the other number. So, 2's complement can be used to represent negative number.

Decimal	2's Complement	1's complement	Signed Magnitude
+7	0111	0111	0111
+6	0110	0110	0110
+5	0101	0101	0101
+4	0100	0100	0100
+3	0011	0011	0011
+2	0010	0010	0010
+1	0001	0001	0001
+0	0000	0000	0000
-0	-----	1111	1000
-1	1111	1110	1001
-2	1110	1101	1010
-3	1101	1100	1011
-4	1100	1011	1100
-5	1011	1010	1101
-6	1010	1001	1110
-7	1001	1000	1111
-8	1000	------	-------

Representation of Real Number

Binary representation of 41.6875 is 101001.1011
Therefore any real number can be converted to binary number system
There are two schemes to represent real number: Fixed-point representation and Floating-point representation.

Fixed-point representation

Binary representation of 41.6875 is 101001.1011
To store this number, we have to store two information,
-- the part before decimal point and
-- the part after decimal point.
This is known as fixed-point representation where the position of decimal point is fixed and number of bits before and after decimal point are also predefined.
If we use 16 bits before decimal point and 8 bits after decimal point, in signed magnitude form,
One bit is required for sign information, so the total size of the number is 25 bits (
1(sign) + 16(before decimal point) + 8(after decimal point)).

Floating-point representation

In this representation, numbers are represented by a mantissa comprising the significant digits and an exponent part of Radix R. The format is:

$mantissa * R^{exponent}$

Numbers are often normalized, such that the decimal point is placed to the right of the first non- zero digit.
For example, the decimal number, 5236 is equivalent to $5.236 * 10^3$
To store this number in floating point representation, we store 5236 in mantissa part and 3 in exponent part. IEEE has proposed two standard for representing floating-point number:

- Single precision
- Double precision

Single Precision:

S: sign bit: 0 denoted + and 1 denotes -

E: 8-bit excess -27 exponent

M: 23-bit mantissa

Double Precision:

S: sign bit: 0 denoted + and 1 denotes -

E: 11-bit excess -1023 exponent

M: 52-bit mantissa

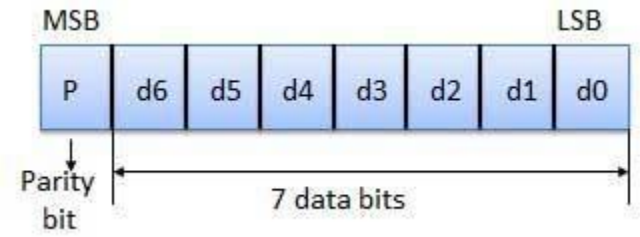
Error Detecting Codes

To detect and correct the errors, additional bits are added to the data bits at the time of transmission.

- The additional bits are called **parity bits**. They allow detection or correction of the errors.
- The data bits along with the parity bits form a **code word**.

Parity Checking of Error Detection

It is the simplest technique for detecting and correcting errors. The MSB of an 8-bits word is used as the parity bit and the remaining 7 bits are used as data or message bits. The parity of 8- bits transmitted word can be either even parity or odd parity.



Even parity -- Even parity means the number of 1's in the given word including the parity bit should be even (2,4,6,.).

Odd parity -- Odd parity means the number of 1's in the given word including the parity bit should be odd (1,3,5,.).

Use of Parity Bit

The parity bit can be set to 0 and 1 depending on the type of the parity required.

- For even parity, this bit is set to 1 or 0 such that the no. of "1 bits" in the entire word is even. Shown in fig. (a).
- For odd parity, this bit is set to 1 or 0 such that the no. of "1 bits" in the entire word is odd. Shown in fig. (b).

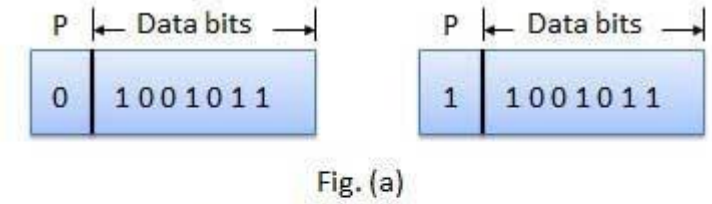


Fig. (a)

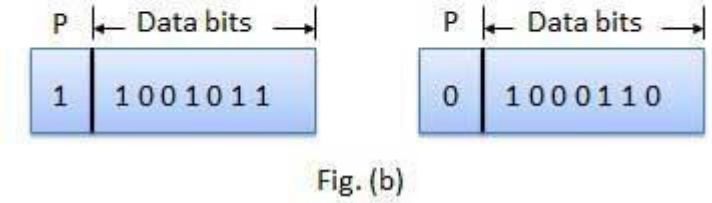
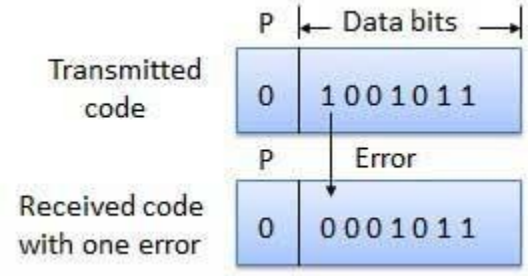


Fig. (b)

How Does Error Detection Take Place?

Parity checking at the receiver can detect the presence of an error if the parity of the receiver signal is different from the expected parity. That means, if it is known that the parity of the transmitted signal is always going to be "even" and if the received signal has an odd parity, then the receiver can conclude that the received signal is not correct. If an error is detected, then the receiver will ignore the received byte and request for retransmission of the same byte to the transmitter.



Register Transfer and Micro-operations

Register Transfer Language

A digital system is an interconnection of digital hardware modules that accomplish a specific information-parsing task. The modules are constructed from such digital components as registers, decoders, arithmetic elements, and control logic. The various modules are interconnected with common data and control paths to form a digital computer system. Digital modules are best defined by the registers they contain and the operations that are performed on the data stored in them. The operations
executed on data stored in registers are called micro-operations.

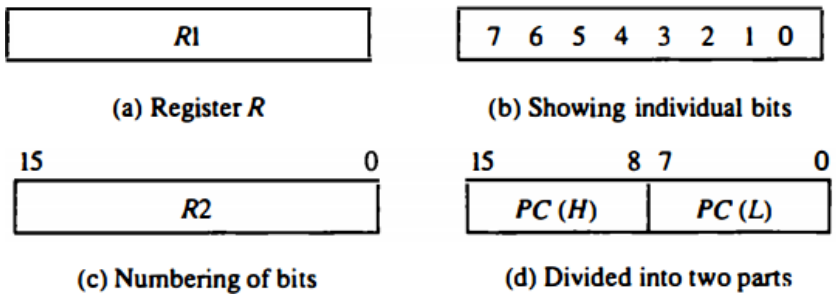
The internal hardware origination of a digital computer is best defined by specifying:
1. The set of registers it contains and their function.
2. The sequence of micro operations performed on the binary information stored in the registers.
3. The control that initiates the sequence of micro operations.

The symbolic notation used to describe the micro operation transfers among registers is called a register transfer language. The term "register transfer" imples the availability of hardware logic circuits that can perform a stated micro operation and transfer the result of the operation to the same or another register.

Register Transfer

Computer registers are designated by capital letters (sometimes followed by numerals) to denote the function of the register. 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 MR. Other designations for registers 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 i sequence from 0 through n - 1, starting from 0 i the rightmost position and increasing the numbers toward the left. Figure shows the representation of registers in block diagram for. The most common way to represent a register is by a rectangular box with the name of the register inside.

Block diagram of register.



A statement that specifies a register transfer imples that circuits are available from the outputs of the source register to the inputs of the destination register and that the destination register has a parallel load capability. Every statement written in a register transfer notation implies a hardware construction for implementing the transfer.
The basic symbols o f the register transfer notation are listed in Table. Registers are denoted by capital letters, and numerals may follow the letters. Parentheses are used to denote a part of a register by specifying the range of bits or by giving a symbol name to a portion of a register. The

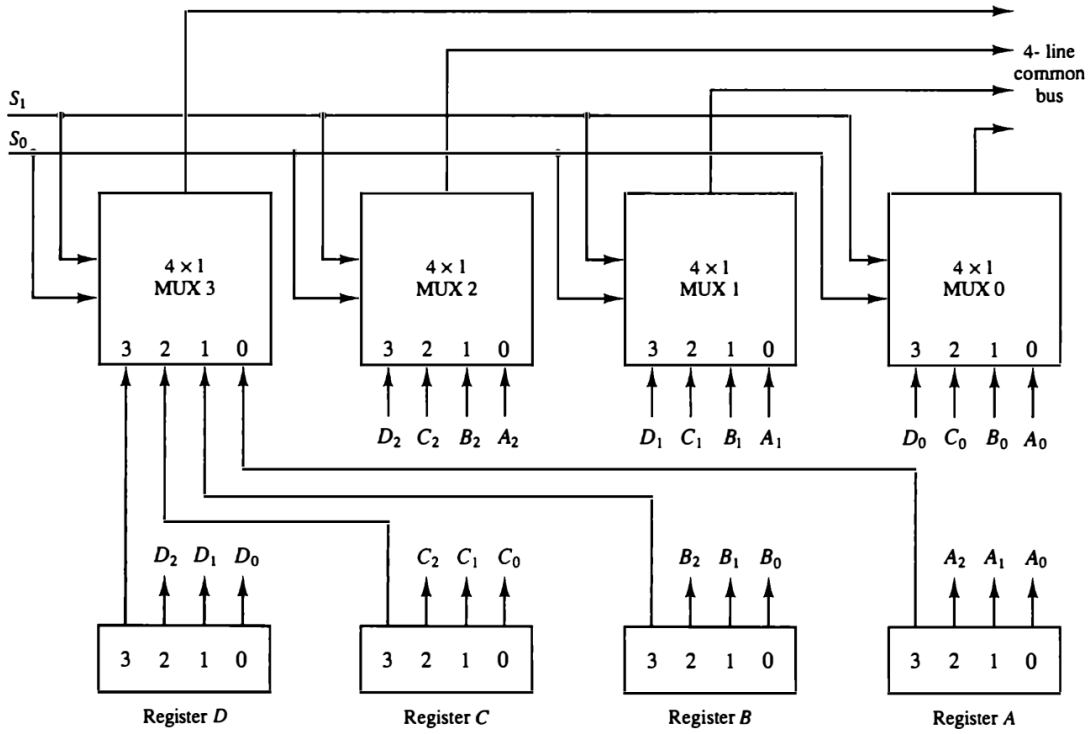
arrow denotes a transfer of information and the direction of transfer. A comma is used to separate two or more operations that are executed at the same time.

Basic Symbols for Register Transfers

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* ← *R1*
Comma ,	Separates two microoperations	*R2* ← *R1*, *R1* ← *R2*

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. One way of constructing a common bus system is with multiplexers. The multiplexers select the source register whose binary information is then placed on the bus. The construction of a bus system for four registers is shown in Figure.



Each register has four bits, numbered 0 through 3. The bus consists of four 4x1 multiplexers each having four data inputs, 0 through 3, and two selection inputs. S1 and S0. In order not to complicate the diagram with 16 lines crossing each other, we use labels to show the connections from the outputs of the registers to the inputs of the multiplexers. For example, output 1 of register A is connected to input 0 of MUX 1 because this input is labeled A1. The diagram shows that the bits in the same significant position in each register are connected to the data inputs of one multiplexer to form one line of the bus. Thus MUX 0 multiplexes the four 0 bits of the registers, MUX 1 multiplexes the four 1 bits of the registers, and similarly for the other two bits.

The two selection lines Si and S 0 are connected to the selection inputs of all four multiplexers. The selection lines choose the four bits of one register and transfer them into the four-line common bus. When S1S0 = 00, the 0 data inputs of all four multiplexers are selected and applied to the outputs that form the bus. This causes the bus lines to receive the content of register A since the outputs of this register are connected to the 0 data inputs of the multiplexers. Similarly, register B is selected if S1S0 = 01, and so on. Table shows the register that is selected by the bus for each of the four possible binary value of the selection lines.

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

In general, a bus system will multiplex k registers of n bits each to produce an n - line common bus. The number of multiplexers needed to construct the bus is equal to n, the number of bits in each 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. The symbolic statement for a bus transfer may mention the bus or its presence may be implied in the statement. When the bus is includes in the statement, the register transfer is symbolized as follows:

BUS □ C, R1 □ BUS

The content of register C is placed on the bus, and the content of the bus is loaded into register R1 by activating its load control input. If the bus is known to exist in the system, it may be convenient just to show the direct transfer.
R1 □ C

From this statement the designer knows which control signals must be activated to produce the transfer through the bus.

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. It is necessary to specify the address of M when writing memory transfer operations. This will be done by enclosing the address in square brackets following 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 R1 into the memory word M selected by the address in AR.

Arithmetic Micro-operations

The basic arithmetic micro operations are addition, subtraction, increment, decrement and shift. The basic arithmetic micro operations are listed in Table. Subtraction is most often implemented through complementation and addition. Instead of using minus operator, we can specify the subtraction by the following statement:

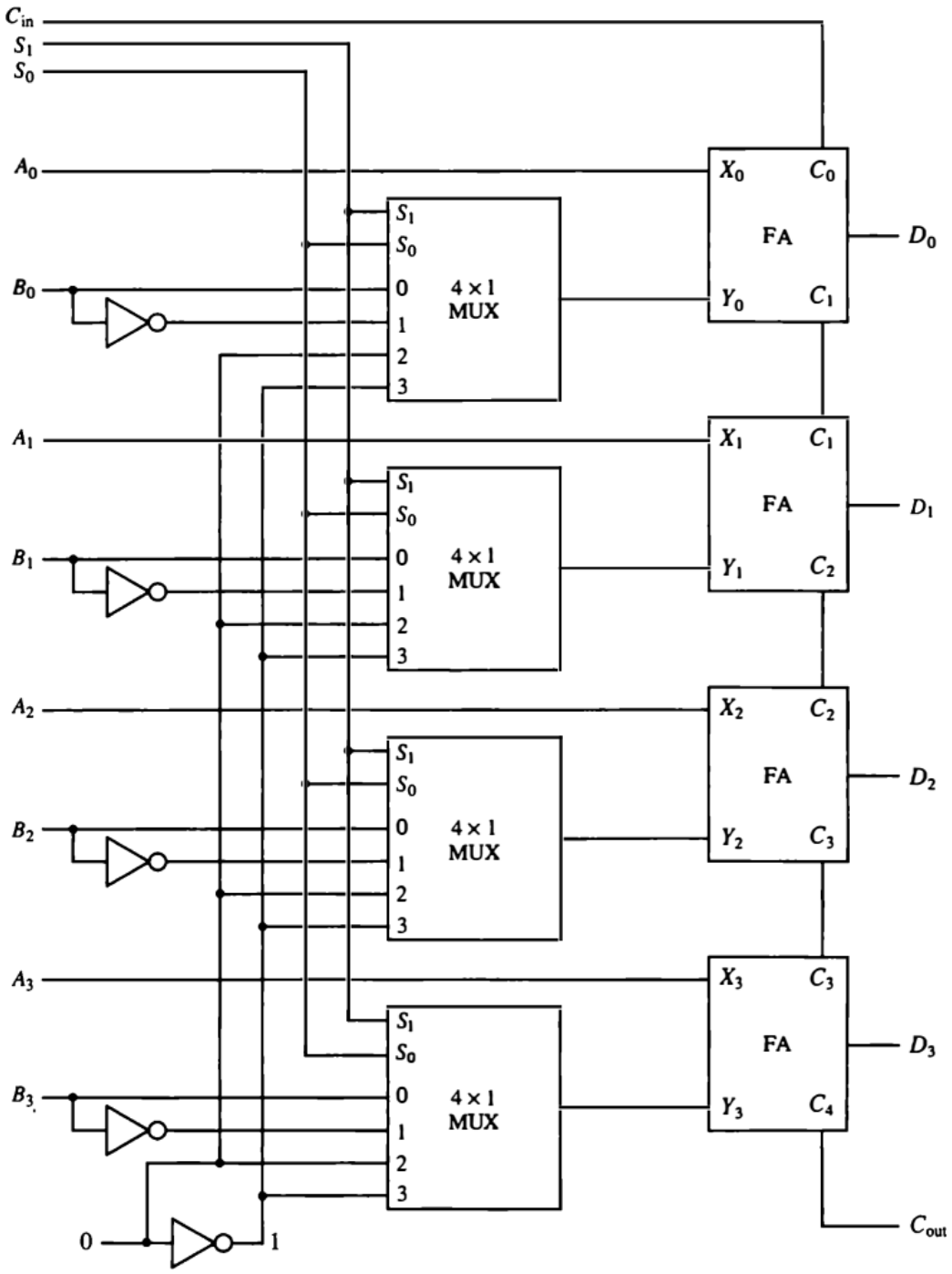
$R3 \leftarrow R1 + \overline{R2} + 1$

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

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

The arithmetic micro operations listed in Table can be implemented in one composite arithmetic circuit. The basic component of an arithmetic circuit is the parallel adder. By controlling the data inputs to the adder, it is possible to obtain different types of arithmetic operations. The diagram of a 4-bit arithmetic circuit is shown in 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 four inputs from B are connected to the data inputs of the multiplexers. The multiplexers data inputs also receive the complement of B.



The other two data inputs are connected to logic-0 and logic-1 . Logic-0 is a 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. S1 and S0 . The input carry On 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 + Cin

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. Cin is the input carry, which can be equal to 0 or 1. Note that the symbol + in the equation above denotes an arithmetic plus. By controlling the value of Y with the two selection inputs S1 and S0 and making Cin equal to 0 or 1, it is possible to generate the eight arithmetic micro operations listed in 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

When S1S0 = 00, the value of B is applied to the Y inputs of the adder. If Cin = 0, the output D = A + B. If Cin = 1, output D = A + B + 1. Both cases perform the add micro operation with or without adding the input carry.

When S1S0 = 01, the complement of B is applied to the Y inputs of the adder. If Cin = 1, then D = A + B +1. This produces A plus the 2's complement of B, which is equivalent to a subtraction of A - B. When Cin= 0, then D = A + B . This is equivalent to a subtract with borrow, that is, A - B - 1.

When S1S0 = 10, the inputs from B are neglected, and instead, all 0's are inserted into the Y inputs. The output becomes D = A + 0 + Cin. This gives D = A when Cin= 0 and D = A + 1 when Cin = 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.

When S1S0 = 11, all 1's are inserted into the Y inputs of the adder to produce the decrement operation D = A - 1 when Cin = 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's complement of 1 = A - 1. When Cin= 1, then D = A - 1 + 1 = A, which causes a direct transfer from input A to output D. Note that the micro operation D = A is generated twice, so there are only seven distinct micro operations in the arithmetic circuit.

Logic Micro operations

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 micro operation with the contents of two registers R1 and R2 is symbolized by the statement

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

It specifies a logic micro operation to be executed on the individual bits of the registers provided that the control variable P = 1. The content of R1, after the execution of the micro operation, is

equal to the bit-by-bit exclusive-OR operation on pairs of bits in R2 and previous values of R1. The logic micro operations 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 micro operations OR, AND, and complement, to distinguish them from the corresponding symbols used to express Boolean functions. The symbol V will be used to denote an OR micro operation and the symbol ∧ to denote an AND micro operation. The complement micro operation is the same as the Ts complement and uses a bar on top of the symbol that denotes the register name. By using different symbols, it will be possible to differentiate between a logic micro operation and a control (or Boolean) function.

List of Logic Micro operations

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. 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.

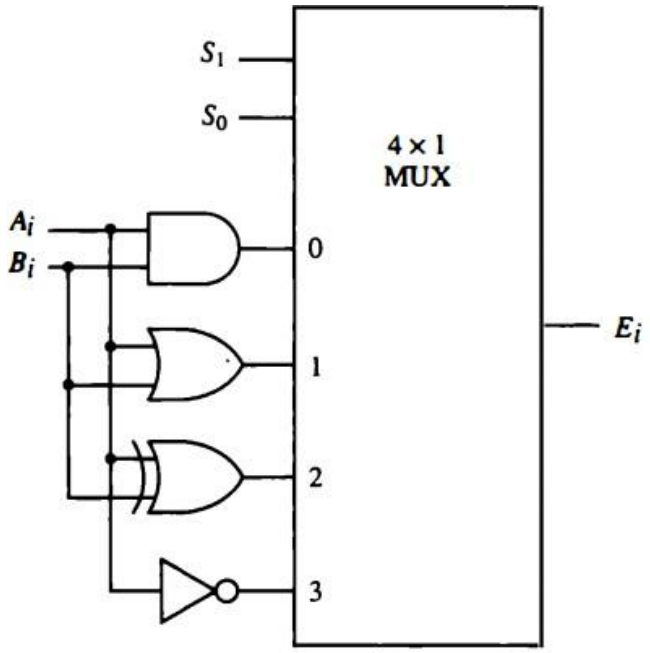
x	y	F_0	F_1	F_2	F_3	F_4	F_5	F_6	F_7	F_8	F_9	F_{10}	F_{11}	F_{12}	F_{13}	F_{14}	F_{15}
0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
0	1	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
1	0	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
1	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1

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

Note that the functions are determined from the 16 binary combinations that can be assigned to F. The 16 Boolean functions of two variables x and y are expressed in algebraic form in the first column of Table. The 16 logic micro operations 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 represent a relationship between two binary variables x and y. The logic micro operations 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 micro operation is performed on the string of bits stored in the registers.

Hardware Implementation

The hardware implementation of logic micro operations 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 micro operations, most computers use only four — AND, OR, XOR (exclusive-OR), and complement from which all others can be derived. Figure shows one stage of a circuit that generates the four basic logic micro operations. 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 S1 and S0 choose one of the data inputs of the multiplexer and direct its value to the output. The diagram shows one typical stage with subscript i. For a logic circuit with n bits, the diagram must be repeated n times for i = 0, 1, 2, . . . , n - 1. The selection variables are applied to all stages. The function table in Figure lists the logic micro operations obtained for each combination of the selection variables.



(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

Logic micro operations are very useful for manipulating individual bits or a portion of a word stored in a register. They can be used to change bit values, delete a group of bits, or insert new bit values into a register.

Shift Micro operations

Shift micro operations are used for serial transfer of data. They are also used in conjunction with arithmetic, logic, and other data-processing operations. The contents of a register can be shifted to the left or the right. At the same time that the bits are shifted, the first flip-flop receives its binary information from the serial input. During a shift-left operation the serial input transfers a bit into the rightmost position. During a shift-right operation the serial input transfers a bit into the leftmost position. The information transferred through the serial input determines the type of shift. 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 micro operations. For example:

R1 ←shl R1

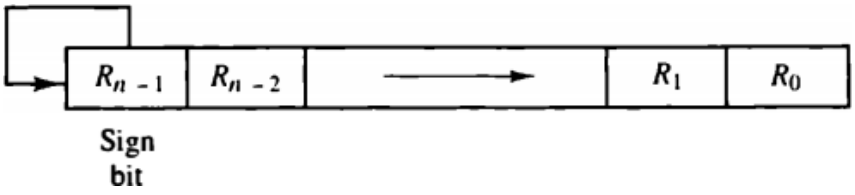
R2 ← shrR2

are two micro operations that specify a 1-bit shift to the left of the content of register Rl 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 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. We will use the symbols cil and cir for the circular shift left and right, respectively. The symbolic notation for the shift micro operations is shown in Table.

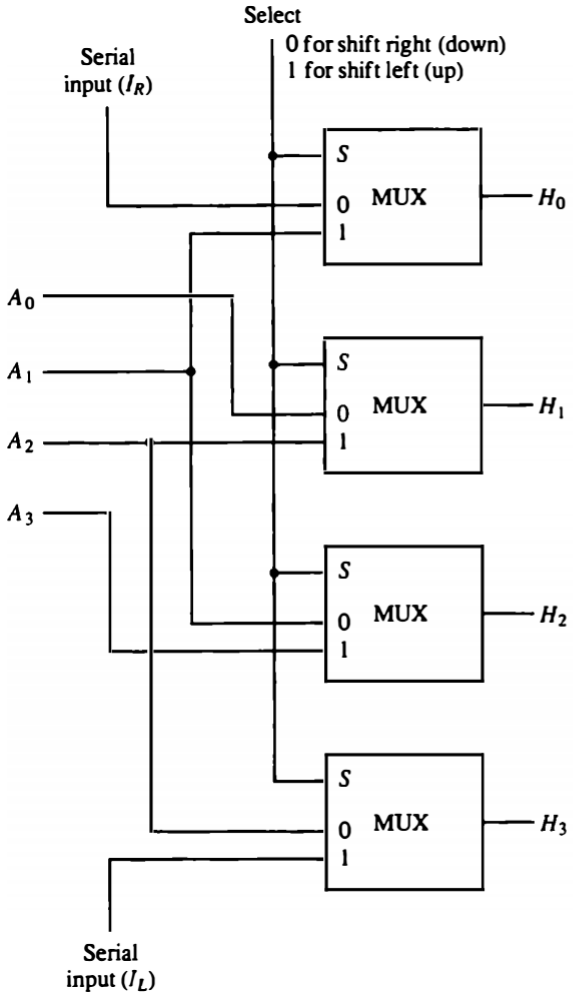
Symbolic designation	Description
$R \leftarrow$shl R	Shift-left register R
$R \leftarrow$shr R	Shift-right register R
$R \leftarrow$cil R	Circular shift-left register R
$R \leftarrow$cir R	Circular shift-right register R
$R \leftarrow$ashl R	Arithmetic shift-left R
$R \leftarrow$ashr R	Arithmetic shift-right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

An arithmetic shift is a micro operation that shifts a signed binary number to the left or right. An arithmetic shift-left multiplies a signed binary number by 2. An arithmetic shift-right divides the number by 2. Arithmetic shifts must leave the sign bit unchanged because the sign of the number remains the same when it is multiplied or divided by 2. 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. Figure shows a typical register of n bits. Bit R_{n-1} in the leftmost position holds the sign bit. R_{n-2} is the most significant bit of the number and Ro is the least significant bit. The arithmetic shift-right leaves the sign bit unchanged and shifts the number (including the sign bit) to the right. Thus R_{n-1} remains the same, R_{n-2} receives the bit from R_{n-1}, and so on for the other bits in the register. The bit in Ro is lost. The arithmetic shift-left inserts a 0 into Ro, and shifts all other bits to the left. The initial bit of R_{n-1} is lost and replaced by the bit from R_{n-2}. A sign reversal occurs if the bit in R_{n-1} changes in value after the shift. This happens if the multiplication by 2 causes an overflow. An overflow occurs after an arithmetic shift left if initially, before the shift, R_{n-1} is not equal to R_{n-2}.



Hardware Implementation

A possible choice for a shift unit would be a bidirectional shift register with parallel load. Information can be transferred to the register in parallel and then shifted to the right or left. In this type of configuration, a clock pulse is needed for loading the data into the register, and another pulse is needed to initiate the shift. In a processor unit with many registers it is more efficient to implement the shift operation with a combinational circuit. In this way the content of a register that has to be shifted is first placed onto a common bus whose output is connected to the combinational shifter, and the shifted number is then loaded back into the register. This requires only one clock pulse for loading the shifted value into the register.



Function table

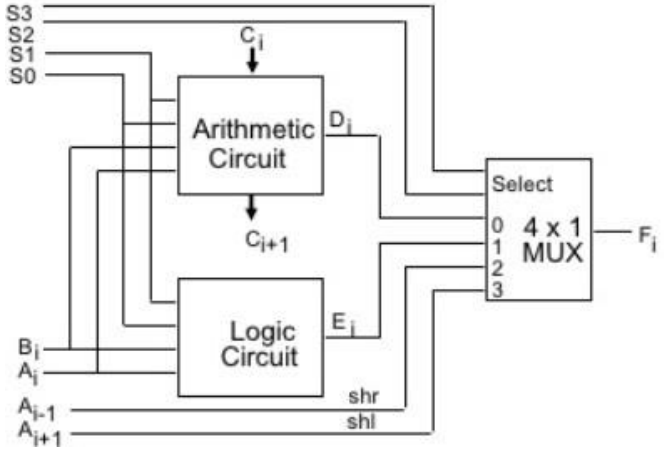
Select	Output			
S	H_0	H_1	H_2	H_3
0	I_R	A_0	A_1	A_2
1	A_1	A_2	A_3	I_L

A combinational circuit shifter can be constructed with multiplexers as shown in Figure. The 4- bit shifter has four data inputs, A0 through A3 , and four data outputs, H0 through H3. There are two serial inputs, one for shift left (IL) and the other for shift right (IR)- When the selection input S = 0, the input data are shifted right (down in the diagram). When S = 1, the input data

are shifted left (up in the diagram). The function table in Figure shows which input goes to each output after the shift. A shifter with n data inputs and outputs requires n multiplexers. The two serial inputs can be controlled by another multiplexer to provide the three possible types of shifts.

Arithmetic Logic Shift Unit

Instead of having individual registers performing the micro operations directly, computer systems employ a number of storage registers connected to a common operational unit called an arithmetic logic unit, abbreviated ALU. To performing a micro operation, 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 micro operations are often performed in a separate unit, but sometimes the shift unit is made part of the overall ALU. The arithmetic, logic, and shift circuits introduced in previous sections can be combined into one ALU with common selection variables. One stage of an arithmetic logic shift unit is shown in Figure. The subscript *i* denotes a typical stage. Inputs Ai and Bi are applied to both the arithmetic and logic units. A particular micro operation 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 A_{i-1} for the shift-right operation and A_{i+1} for the shift-left operation. Note that the diagram shows just one typical stage. The circuit must be repeated n times for an n-bit ALU. The output carry C_{i+1} of a given arithmetic stage must be connected to the input carry Ci of the next stage in sequence. 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 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 is used for selecting an arithmetic operation only. Table lists the 14 operations of the ALU. The first eight are arithmetic operations and are selected with S3S2 = 00. The next four are logic operations and are selected with S3S2 = 01. The input carry has no effect during the logic operations and is marked with don't-care x's. The last two operations are shift operations and are selected with S3S2 = 10 and 11. The other three selection inputs have no effect on the shift.

Operation select						
S_3	S_2	S_1	S_0	C_{in}	Operation	Function
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	$F = A + \overline{B}$	Subtract with borrow
0	0	1	0	1	$F = A + \overline{B} + 1$	Subtraction
0	0	1	1	0	$F = A - 1$	Decrement A
0	0	1	1	1	$F = A$	Transfer A
0	1	0	0	×	$F = A \wedge B$	AND
0	1	0	1	×	$F = A \vee B$	OR
0	1	1	0	×	$F = A \oplus B$	XOR
0	1	1	1	×	$F = \overline{A}$	Complement A
1	0	×	×	×	$F = \text{shr } A$	Shift right A into F
1	1	×	×	×	$F = \text{shl } A$	Shift left A into F