Subject title: Allied 4: COMPUTER SYSTEM ARCHITECTURE

YEAR 2018-19 ONWARDS SEMESTER: IV

SUBJECT CODE: 18BIT44A

UNIT I: Data Representation: Fixed point representation- floating point representation- alphanumeric code- register transfer and micro operation: register transfer language – register transfer – arithmetic micro operation – logic micro operation – shift micro operation – arithmetic logic shift unit.

UNIT II: Basic computer organization and design: instruction codes – timing and control – computer register – instruction cycle – input – output and interrupt – Micro programmed control: control memory – address sequencing- design of control unit.

UNIT III: Central processing unit: general register organization – stack organization – instruction formats – addressing modes – data transfer and manipulation – programmed control – reduced instruction set computer – CISC.

UNIT IV: Computer arithmetic: addition and subtraction – multiplication algorithm – division algorithm – input output organization: peripheral devices- input output interface – asynchronous data transfer – modes of transfer – direct memory access – input output processor (IOP).

UNIT V: Memory organization: memory hierarchy – main memory – auxiliary memory – associative memory – cache memory – virtual memory.

Text book

1. M.Morris Mano, “Computer System Architecture”, third edition, PHI, 2001.

Reference book

1. Hayes J.P, ”Computer architecture and organization”, McGraw Hill,1998.

UNIT I: Data Representation: Fixed point representation-floating point representation- alphanumeric code- register transfer and micro operation: register transfer language – register transfer – arithmetic micro operation – logic micro operation – shift micro operation – arithmetic logic shift unit.

Prepared by Mrs. P.Sundari

DATA REPRESENTATION

Fixed-Point Representation

Positive integers, including zero, can be represented as unsigned numbers. In ordinary arithmetic, a negative number is indicated by a minus sign and a positive number by a plus sign. Because of hardware limitations, computers must represent everything with 1's and 0's, including the sign of a number.

The convention is to make the sign bit equal to 0 for positive and to 1 for negative. In addition to the sign, a number may have a binary (or decimal) point. The position of the binary point is needed to represent fractions, integers, or mixed integer-fraction numbers.

The representation of the binary point in a register is complicated by the fact that it is characterized by a position in the register.

There are two ways of specifying the position of the binary point in a register: by giving it a fixed position or by employing a floating-point representation.

The fixed-point method assumes that the binary point is always fixed in one position. The two positions most widely used are

(1) a binary point in the extreme left of the register to make the stored number a fraction, and
(2) a binary point in the extreme right of the register to make the stored number an integer.

The floating-point representation uses a second register to store a number that designates the position of the decimal point in the first register.

Signed numbers

Integer Representation

When an integer binary number is positive, the sign is represented by 0 and the magnitude by a positive binary number. When the number is negative, the sign is represented by 1 but the rest of the number may be represented in one of three possible ways:

1. Signed-magnitude representation
2. Signed-1' s complement representation
3. Signed 2' s complement representation

The signed-magnitude representation of a negative number consists of the magnitude and a negative sign. In the other two representations, the negative number is represented in either the 1's or 2's complement of its positive value.

As an example, consider the signed number 14 stored in an 8-bit register.

+ 14 is represented by a sign bit of 0 in the leftmost position followed by the binary equivalent of 14: 00001110. Note that each of the eight bits of the register must have a value and therefore 0's must be inserted in the most significant positions following the sign bit. Although there is only one way to represent + 14, there are three different ways to represent - 14 with eight bits.

In signed-magnitude representation 1 0001110
In signed-1's complement representation 1 11 10001
In signed-2's complement representation 1 11 10010

The signed-magnitude representation of - 14 is obtained from + 14 by complementing only the sign bit. The signed-1's complement representation of – 14 is obtained by complementing all the bits of + 14, including the sign bit.

The signed-2's complement representation is obtained by taking the 2's complement of the positive number, including its sign bit.

Therefore, the signed-complement is normally used. The 1's complement imposes difficulties because it has two representations of 0 (+ 0 and - 0).

It is seldom used for arithmetic operations except in some older computers. The 1's complement is useful as a logical operation since the change of 1 to 0 or 0 to 1 is equivalent to a logical
complement operation.

Arithmetic Addition

The addition of two numbers in the signed-magnitude system follows the rules of ordinary arithmetic.

If the signs are the same, we add the two magnitudes and give the sum the common sign.

If the signs are different, we subtract the smaller magnitude from the larger and give the result the sign of the larger magnitude.

For example, (+ 25) + (- 37) = - (37 - 25) = - 12 and is done by subtracting the smaller magnitude 25 from the larger magnitude 37 and using the sign of 37 for the sign of the result. This is a process that requires the comparison of the signs and the magnitudes and then performing either addition or subtraction.

Arithmetic Subtraction

Subtraction of two signed binary numbers when negative numbers are in 2' s complement form is very simple and can be stated as follows:

Take the 2's complement of the subtrahend (including the sign bit) and add it to the minuend (including the sign bit). A carry out of the sign bit position is discarded.

This procedure stems from the fact that a subtraction operation can be changed to an addition operation if the sign of the subtrahend is changed.

Overflow

When two numbers of n digits each are added and the sum occupies n + 1 digits, we say that an overflow occurred.

The detection of an overflow after the addition of two binary numbers depends on whether the numbers are considered to be signed or unsigned.

When two unsigned numbers are added, an overflow is detected from the end carry out of the most significant position. In the case of signed numbers, the leftmost bit always represents the sign, and negative numbers are in 2's complement form.

When two signed numbers are added, the sign bit is treated as part of the number and the end carry does not indicate an overflow.

An overflow cannot occur after an addition if one number is positive and the other is negative, since adding a positive number to a negative number produces a result that is smaller than the larger of the two original numbers.

An overflow may occur if the two numbers added are both positive or both negative.

Overflow detection

An overflow condition can be detected by observing the carry into the sign bit position and the carry out of the sign bit position. If these two carries are not equal, an overflow condition is produced. If the two carries are applied to an exclusive-OR gate, an overflow will be detected when the output of the gate is equal to 1 .

Decimal Fixed-Point Representation

The representation of decimal numbers in registers is a function of the binary code used to represent a decimal digit. A 4-bit decimal code requires four flip-flops for each decimal digit.

The representation of 4385 in BCD requires 16 flip-flops, four flip-flops for each digit. The number will be represented in a register with I6 flip-flops as follows:

0100 001 1 1000 0101

Floating-Point Representation

The floating-point representation of a number has two parts.

The first part represents a signed, fixed-point number called the mantissa.

The second part designates the position of the decimal (or binary) point and is called the exponent.

The fixed-point mantissa may be a fraction or an integer. For example,

The decimal number + 6132.789 is represented in floating-point with a fraction and an exponent as follows:

Fraction + 0 .6132789

Exponent + 04

The value of the exponent indicates that the actual position of the decimal point is four positions to the right of the indicated decimal point in the fraction. This representation is equivalent to the scientific notation +0. 6132789 X 10+4.

Floating-point is always interpreted to represent a number in the following form:

$m \times r^{e}$

Only the mantissa m and the exponent e are physically represented in the register (including their signs). The radix r and the radix-point position of the mantissa are always assumed. The circuits that manipulate the floating-point numbers in registers conform with these two assumptions in order to provide the correct computational results.

A floating-point binary number is represented in a similar manner except that it uses base 2 for the exponent. For example, the binary number + 1001 . 1 1 is represented with a n 8-bit fraction and 6-bit exponent a s follows:

Fraction	01001110
Exponent	000100

The fraction has a 0 in the leftmost position to denote positive. The binary point of the fraction follows the sign bit but is not shown in the register. The exponent has the equivalent binary number +4.

The floating-point number is equivalent to

$$m \times 2^{e} = +(.1001\,110)_2 \times 2^{+4} \cdot$$

Alphanumeric Codes

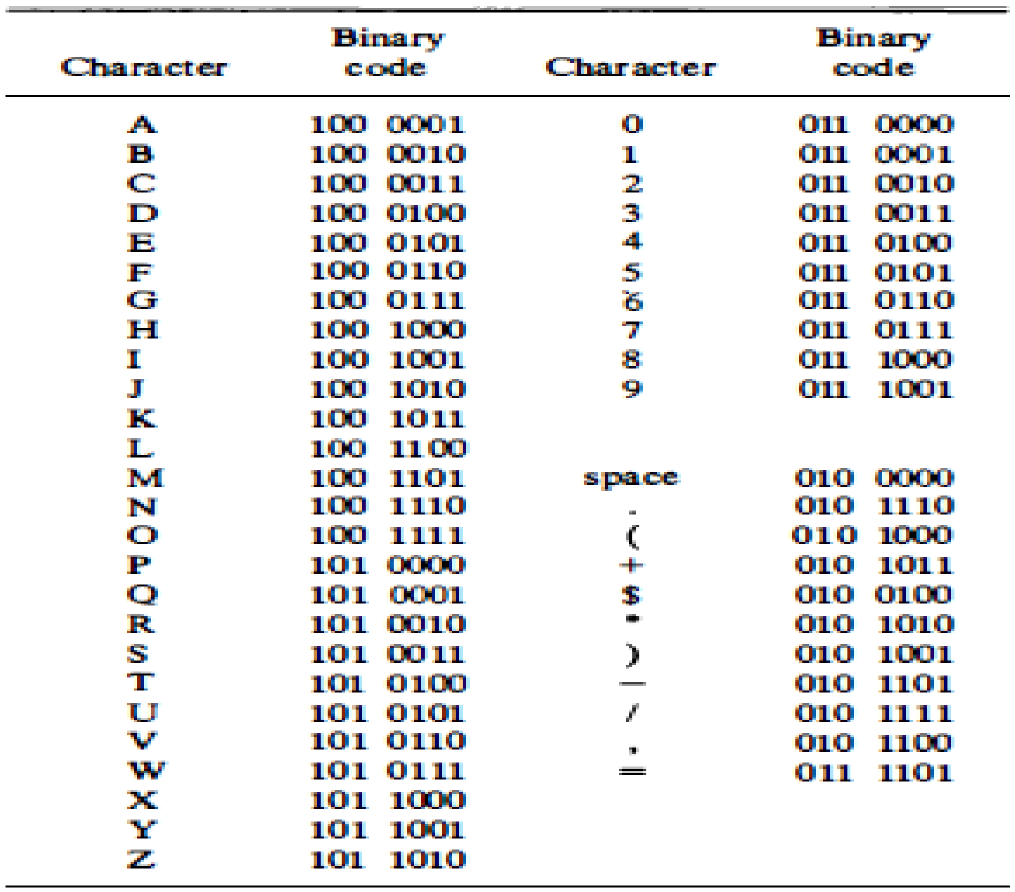
The ASCII (American Standard Code for Information Interchange) code is the standard code commonly used for the transmission of binary information.

Each character is represented by a 7-bit code and usually an eighth bit is inserted for parity.

The code consists of 128 characters. Ninety-five characters represent graphic symbols that include upper- and lowercase letters, numerals zero to nine, punctuation marks, and special symbols. Twenty-three characters represent format effectors, which are functional characters for controlling the layout of printing or display devices such as carriage return, line feed, horizontal tabulation, and back space.

The other 10 characters are used to direct the data communication flow and report its status.

ASCII table



Character	Binary code	Character	Binary code
A	100 0001	0	011 0000
B	100 0010	1	011 0001
C	100 0011	2	011 0010
D	100 0100	3	011 0011
E	100 0101	4	011 0100
F	100 0110	5	011 0101
G	100 0111	6	011 0110
H	100 1000	7	011 0111
I	100 1001	8	011 1000
J	100 1010	9	011 1001
K	100 1011		
L	100 1100		
M	100 1101	space	010 0000
N	100 1110	.	010 1110
O	100 1111	(	010 1000
P	101 0000	+	010 1011
Q	101 0001	$	010 0100
R	101 0010	*	010 1010
S	101 0011	)	010 1001
T	101 0100	—	010 1101
U	101 0101	/	010 1111
V	101 0110	,	010 1100
W	101 0111	=	011 1101
X	101 1000		
Y	101 1001		
Z	101 1010		

EBCDIC

Another alphanumeric (sometimes called alphanumeric) code used in IBM equipment is the EBCDIC (Extended BCD Interchange Code). It uses eight bits for each character (and a ninth bit for parity). EBCDIC has the same character symbols as ASCII but the bit assignment to characters is different. When alphanumeric characters are used internally in a computer for data processing (not for transmission purposes) it is more convenient to use a 6-bit code to represent 64 characters.

A 6-bit code can specify the 26 uppercase letters of the alphabet, numerals zero to nine, and up to 28 special characters. This set of characters is usually sufficient for data-processing purposes. Using fewer bits to code characters has the advantage of reducing the memory space needed to store large quantities of alphanumeric data.

REGISTER TRANSFER AND MICRO OPERATION

Register Transfer Language

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 result of the operation may replace the previous binary information of a register or may be transferred to another register. Examples of micro operations are shift, count, clear, and load.

The internal hardware organization 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.

Register transfer language

The symbolic notation used to describe the micro operation transfers among registers is called a register transfer language.

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 MAR.

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 in sequence from 0 through n - 1, starting from 0 in the rightmost position and increasing the numbers toward the left.

Information transfer from one register to another is designated in symbolic form by means of a replacement operator. The statement

$$R2 \leftarrow R1$$

denotes a transfer of the content of register R1 into register R2.

By definition, the content of the source register R1 does not change after the transfer.

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

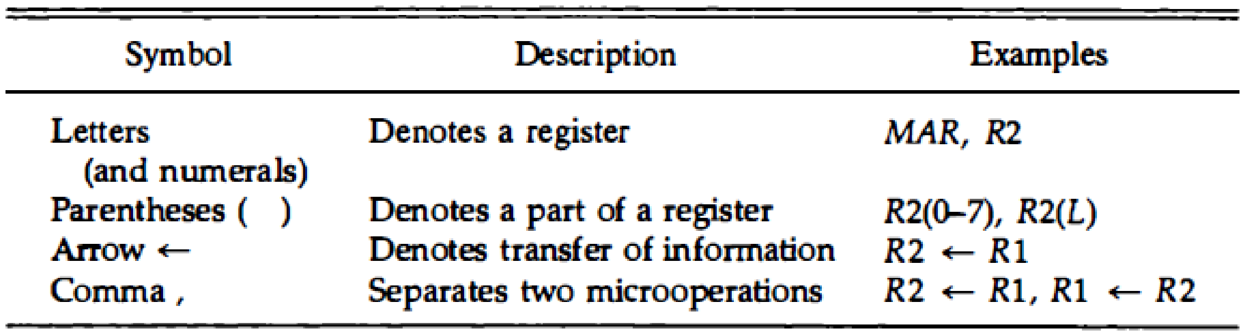
where P is a control signal generated in the control section. It is sometimes convenient to separate the control variables from the register transfer operation by specifying a control function.

A control function is a Boolean variable that is equal to 1 or 0. The control function is included in the statement as follows:

$$P\text{: } R2 \leftarrow R1$$

The control condition is terminated with a colon.

Basic symbols for register transfer



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

Arithmetic Micro operations

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-Subtractor

The subtraction of binary numbers can be done most conveniently by means of complements.

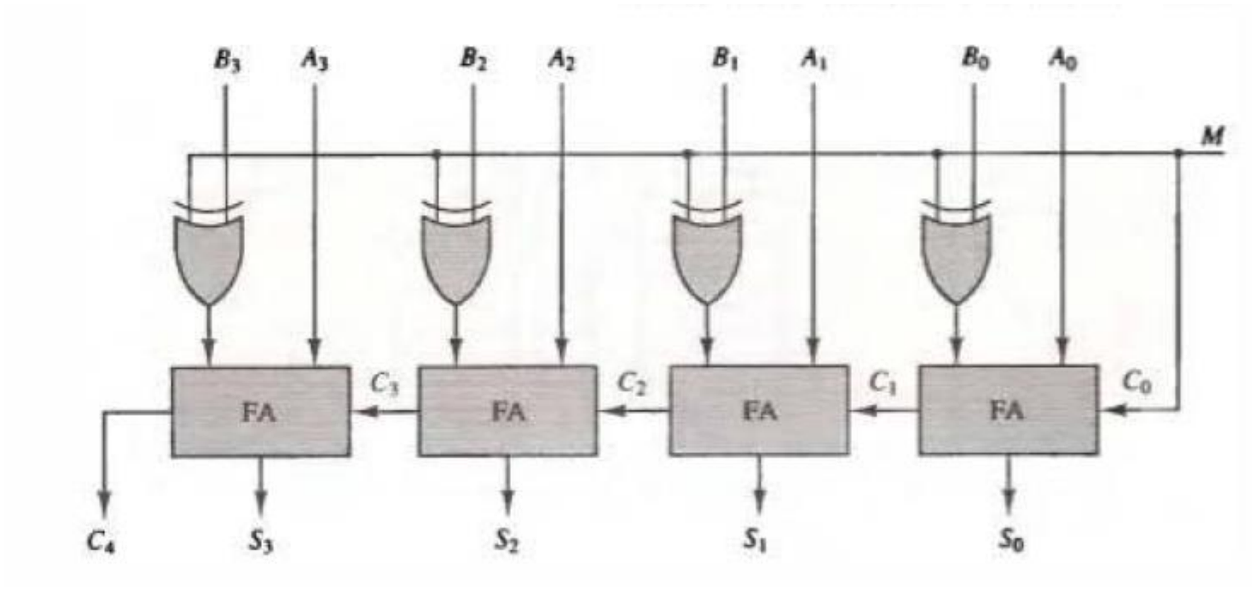
Remember that the subtraction A – B can be done by taking the 2's complement of B and adding it to A.

The 2's complement can be obtained by taking the 1's complement and adding one to the least significant pair of bits.

The 1's complement can be implemented with inverters and a one can be added to the sum through the input carry.

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

Binary Adder-Subtractor

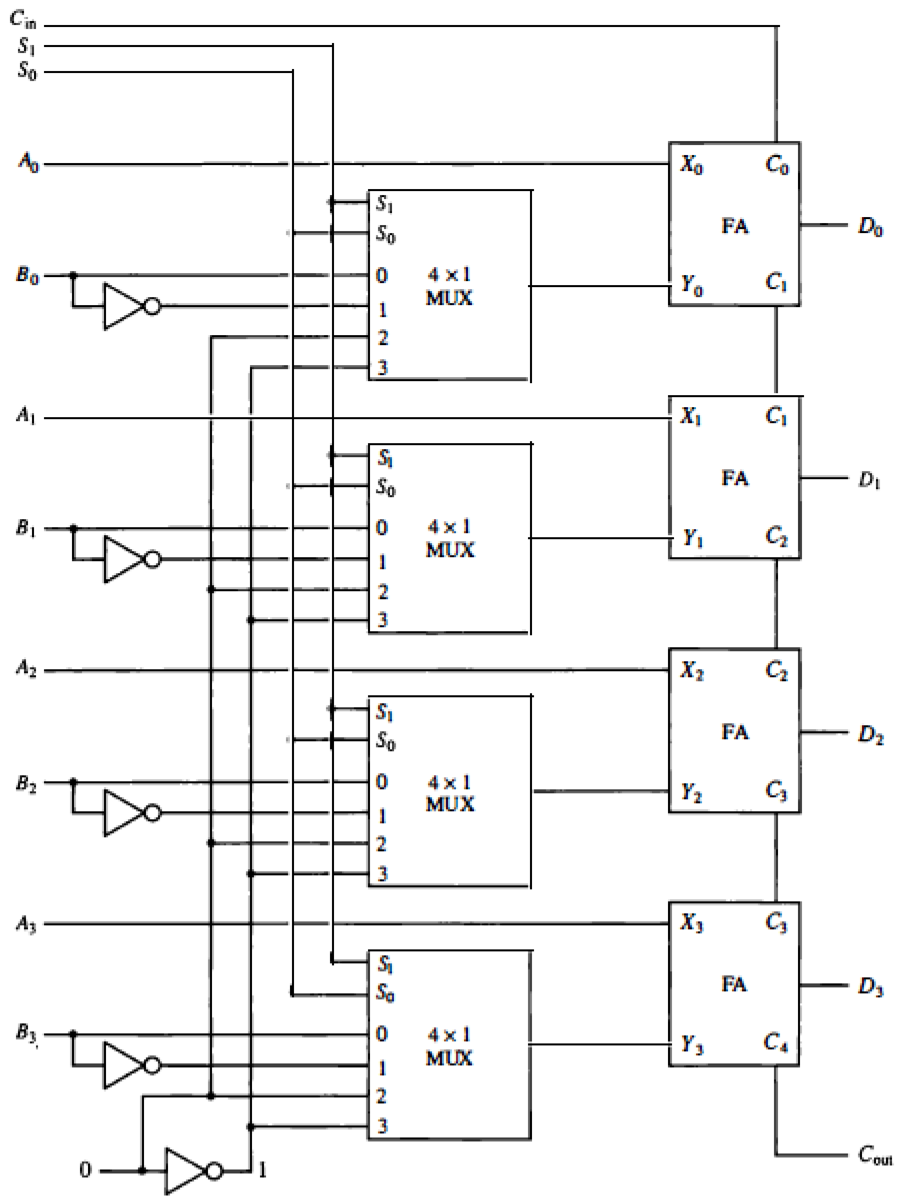


Arithmetic Circuit

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 Fig. 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 The four multiplexers are controlled by two selection inputs, S1 and S0• The input carry Cin goes to the carry input of the FA in the least significant position. The other carries are connected from one stage to the next.

4 bit arithmetic circuit

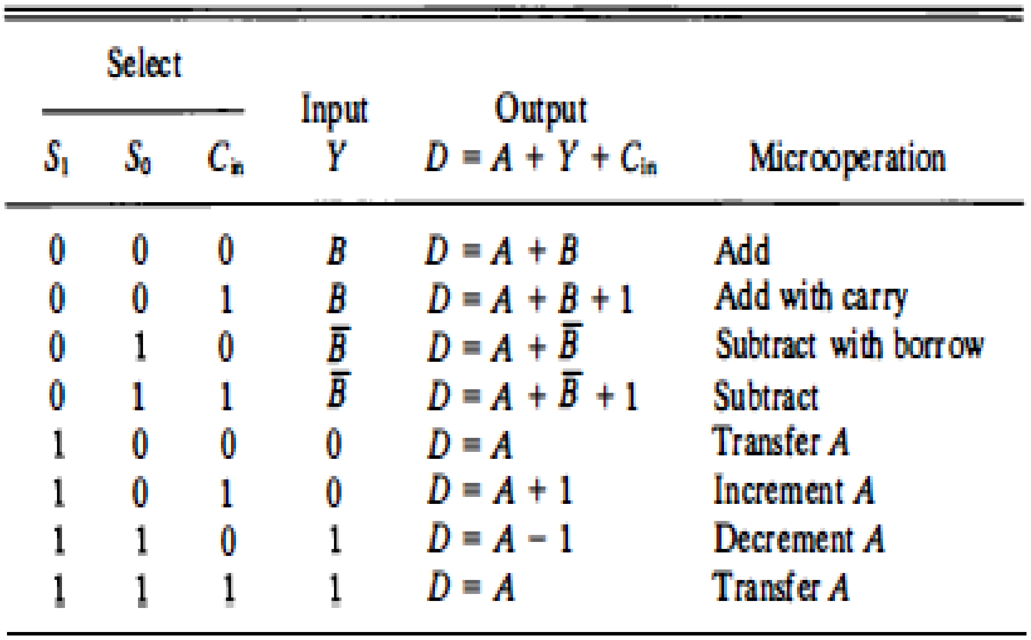


The output of the binary adder is calculated from the following arithmetic sum:

$$D = A + Y + Cin$$

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 So and making Cin equal to 0 or 1, it is possible to generate the eight arithmetic micro operations listed in Table.

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	$\bar{B}$	$D = A + \bar{B}$	Subtract with borrow
0	1	1	$\bar{B}$	$D = A + \bar{B} + 1$	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

Logic Micro operations

Logic micro operations specify binary operations for strings of bits stored in registers.

For example, the exclusive-OR micro operation with the contents of two registers R 1 and R2 is symbolized by the statement

P: R1 $\leftarrow$ R1 xor R2

Special symbols

The symbol V will be used to denote an OR micro operation.

List of Logic Micro operations

There are 16 different logic operations that can be performed with two binary variables.

Truth table for 16 functions of 2 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

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 .

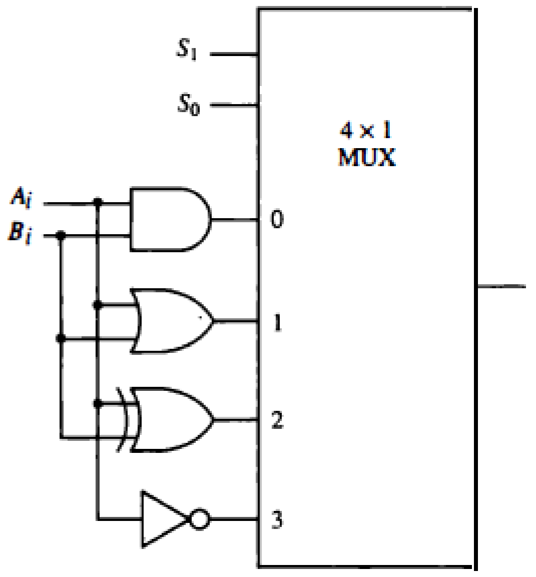
The 16 Boolean functions of two variables

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
$F_4 = x'y$	$F \leftarrow \overline{A} \wedge B \mid$	
$F_5 = y$	$F \leftarrow B$	Transfer B
$F_6 = x \oplus y$	$F \leftarrow A \oplus B$	Exclusive-OR
$F_7 = x + y$	$F \leftarrow A \vee B$	OR
$F_8 = (x + y)'$	$F \leftarrow \overline{A \vee B}$	NOR
$F_9 = (x \oplus y)'$	$F \leftarrow \overline{A \oplus B}$	Exclusive-NOR
$F_{10} = y'$	$F \leftarrow \overline{B}$	Complement B
$F_{11} = x + y'$	$F \leftarrow A \vee \overline{B}$	
$F_{12} = x'$	$F \leftarrow \overline{A}$	Complement A
$F_{13} = x' + y$	$F \leftarrow \overline{A} \vee B$	
$F_{14} = (xy)'$	$F \leftarrow \overline{A \wedge B}$	NAND
$F_{15} = 1$	$F \leftarrow$ all 1's	Set to all 1's

Hardware Implementation

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.

One stage of logic circuit



(a) Logic diagram

S_1	S_0	Output	Operation
0	0	$E = A \wedge B$	AND
0	1	$E = A \vee B$	OR
1	0	$E = A \oplus B$	XOR
1	1	$E = \overline{A}$	Complement

(b) Function table

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:

1. logical,
2. circular, and
3. arithmetic.

Logical shift

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 ← shr R2

are two micro operations that specify a 1-bit shift to the left of the content of register R 1 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.

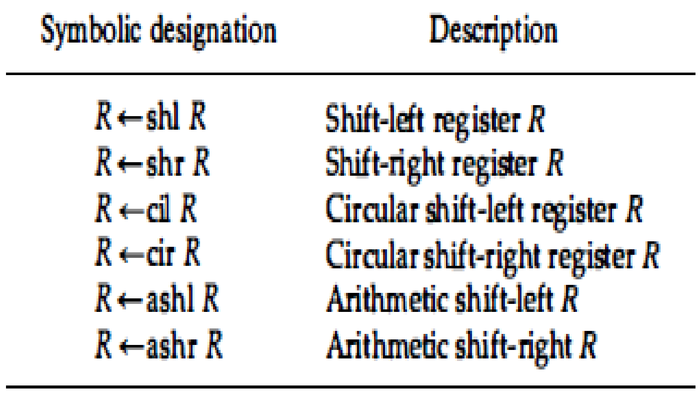
Circular shift

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.

Shift micro operations



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

Arithmetic shift

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.

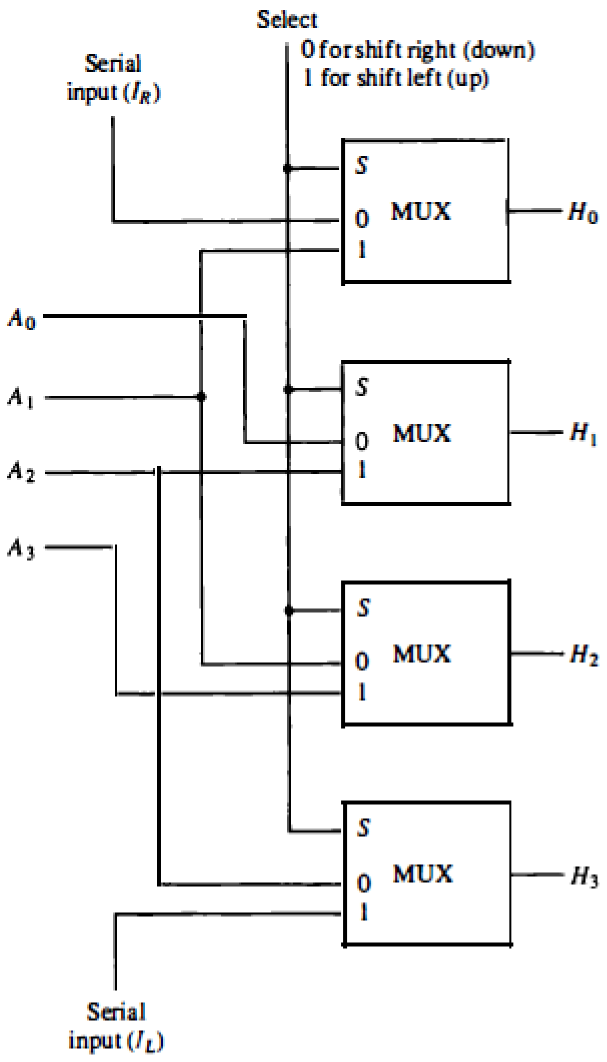
Hardware Implementation

A combinational circuit shifter can be constructed with multiplexers as shown in Fig. 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 (h).

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 Fig. 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.

4 bit combinational circuit shifter



Function table

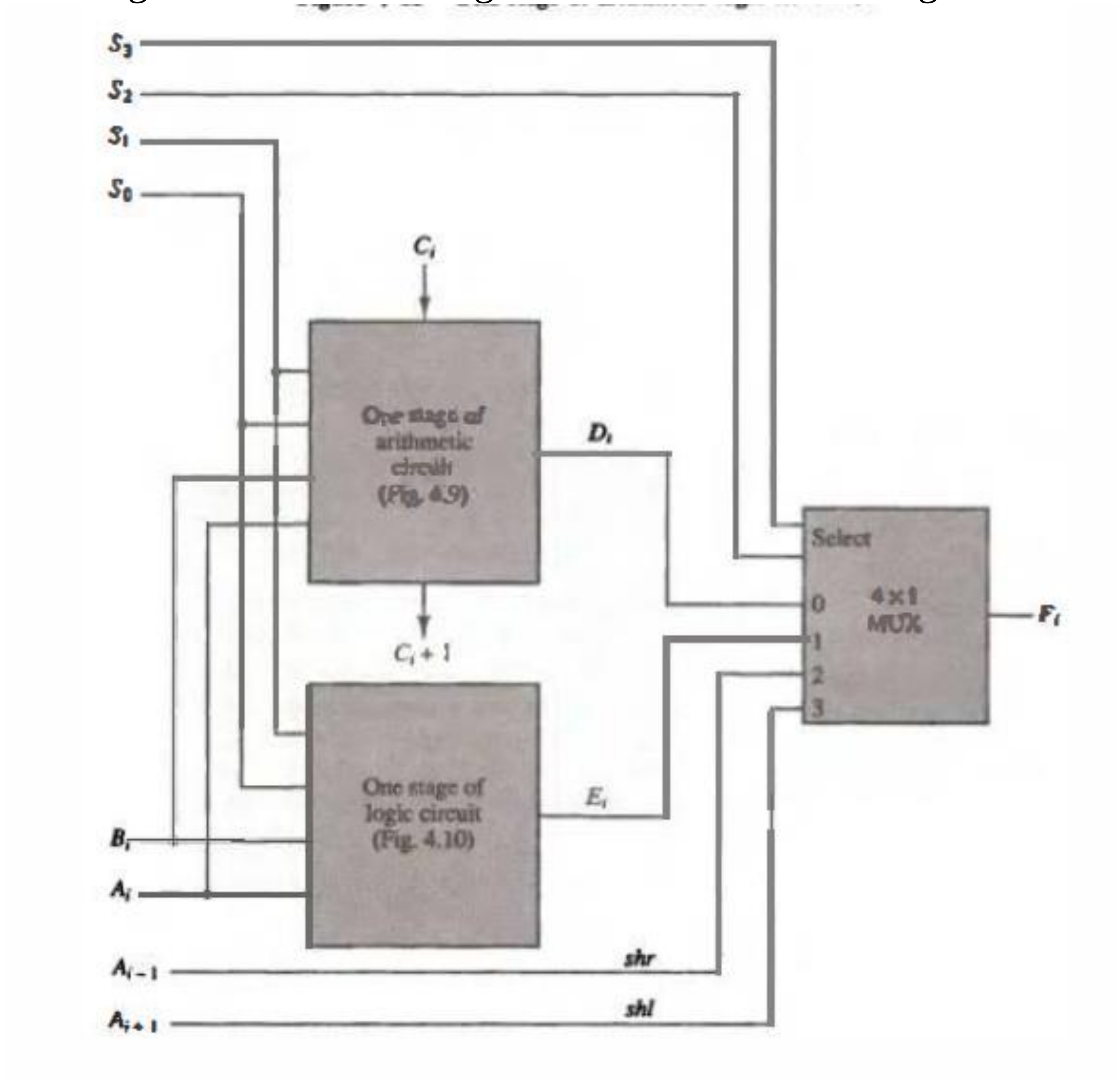
Select	Output			
S	H_0	H_1	H_2	H_3
0	I_R	A_0	A_1	A_2
1	A_1	A_2	A_3	I_L

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.

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.

One stage of an arithmetic logic shift unit is shown in Fig.



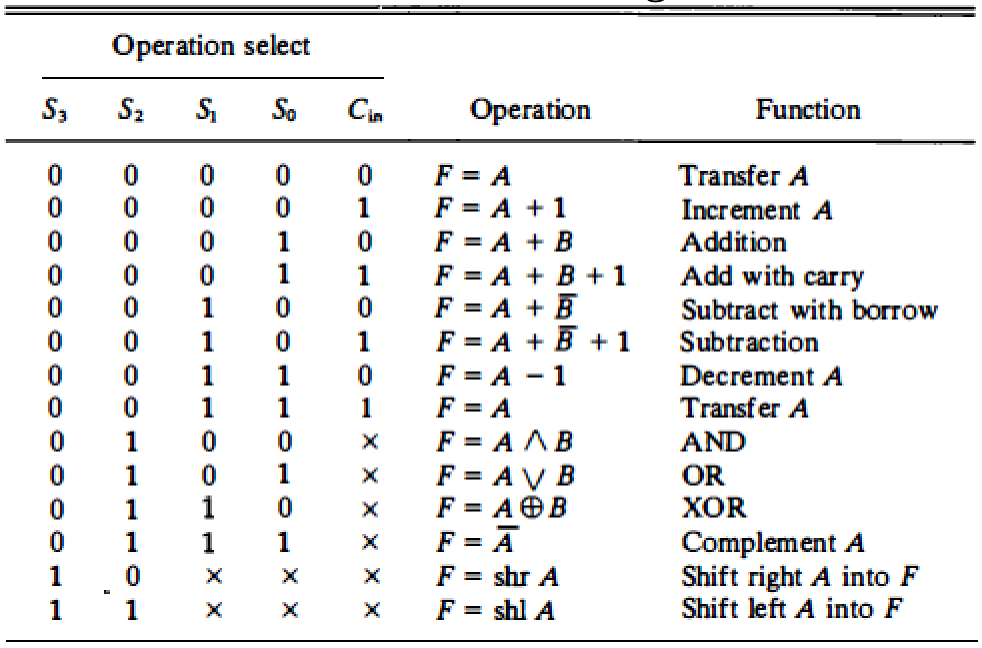
Inputs Ai and Bi are applied to both the arithmetic and logic units. A particular micro operation is selected with inputs S1 and S0•

A 4 x 1 multiplexer at the output chooses between an arithmetic output in E; and a logic output in H;. 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 + 1 for the shift-left operation.

Note that the diagram shows just one typical stage. The circuit of Fig. must be repeated n times for an n-bit ALU. The output carry Ci+ 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 Fig 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 Cin., is used for selecting an arithmetic operation only.

Function table for arithmetic logic shift unit



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

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.