2
Register Transfer Logic

2.1 INTRODUCTION

The design of computer system is carried out at several levels of abstraction. They are:

(a) The processor level (also called architecture or system level)
(b) The register level (also called register-transfer level)
(c) The gate level (also called logic level)

The processor level corresponds to a user's view of a computer. The register level is approximately seen by a programmer. The gate level concerns with the hardware design.

This chapter introduces the register level design of the computer, defines register transfer language and shows how it is used to express micro-operations in symbolic form. At the register-transfer level, information bits are grouped into words and the primitive, components are small combinational or sequential circuits intended to store or process words. Combinational and sequential circuits can be used to create simple digital systems. These are the low-level building blocks of digital computer. Simple digital systems are frequently characterized in terms of: the register they contain and the operation they perform. Symbols are defined for arithmetic, logic, and shift micro-operations. A composite arithmetic logic shift unit is developed to show the hardware design of the most common micro-operations.

2.2 REGISTER TRANSFER LANGUAGE AND MICRO-OPERATIONS

A digital system is an interconnection of digital hardware modules that accomplish a specific information-processing task. Digital systems vary in size and complexity from a few integrated circuits to a complex of interconnected and interacting digital computers. Digital system design invariably uses a modular approach. 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 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 towards the left. Figure 2.1 shows the representation of registers in block diagram form. The most common way to represent a register is by a rectangular box with the name of the register inside, as in Fig. 2.1 (a). The individual bits can be distinguished as in (b). The numbering of bits in a 16-bit register can be marked on top of the box as shown in (c). A 16-bit register is partitioned into two parts in (d). Bits 0 through 7 are assigned the symbol L (for low byte) and bits 8 through 15 are assigned the symbol H (for high byte). The name of the 16-bit register is PC. The symbol PC(0-7) or PC(L) refers to the low-order byte and PC(8-15) or PC(H) to the high-order byte.

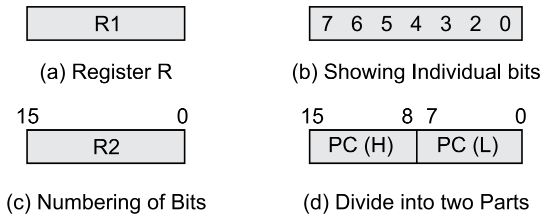


Fig. 2.1 Block diagram of register.

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. It designates a replacement of the content of R2 by the content of R1. By definition, the content of the source register R1 does not change after the transfer. A statement that specifies a register transfer implies that circuits are available from the outputs of the source register to the inputs of the destination register and that the destination register has a parallel load capability. Normally, we want the transfer to occur only under a predetermined control condition.

This can be shown by means of an *if-then* statement.

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

where P is a control signal generated in the control section. It is sometimes convenient to separate the control variables from the register transfer operation by specifying a *control function.* A control function is a Boolean variable that is equal to 1 or 0. The control function is included in the statement as follows:

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

The control condition is terminated with a colon. It symbolizes the requirement that the transfer operation be executed by the hardware only if P = 1. Every statement written in a register transfer

Table 2.1 Basic symbols for register transfers

Symbol	*Description*	*Example*
Letter (and numerals)	Denotes a register	MAR, R2
Parenthesis	Denotes a part of register	R2(0-7) , R2(H)
Arrow ←	Denotes transfer of information	R2 ← R1
Comma,	Separates two micro-operations	R2 ← R1, R1 ← R2

2.4 BUS TRANSFER

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 or a bus system for four registers is shown in Fig. 2.3. Each register has four bits, numbered 0 through 3. The bus consists of four 4 × 1 multiplexers each having four data inputs, 0 through 3, and two selection inputs, S_1 and S_0. 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 A_1 . 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.

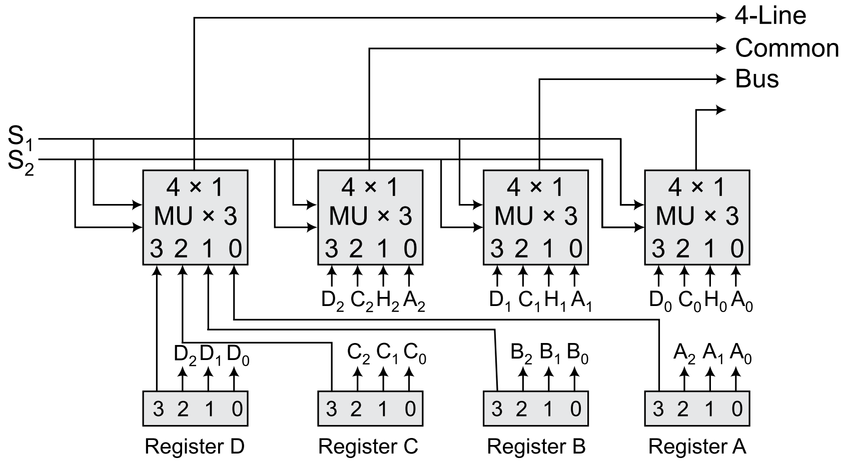


Fig. 2.3 Bus system for four registers.

like an open circuit, which means that 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 one most commonly used in the design of a bus system is the buffer gate.

The graphic symbol of a three-state buffer gate is shown in Fig. 2.4. It 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 equal to 1, the output is enabled and the gate behaves like any conventional buffer, with the output equal to the normal input. When the control input is 0, the output is disabled and the gate goes to a high-impedance state, regardless of the value in the normal input. The high-impedance state of a three-state gate provides a special feature not available in other gates. Because of this feature, a large number of three-state gate outputs can be connected with wires to form a common bus line without endangering loading effects.

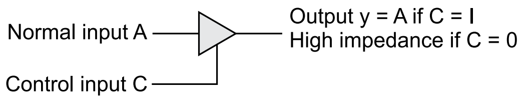


Fig. 2.4 Graphic symbols for three-state buffer.

The construction of a bus system with three-state buffers is demonstrated in Fig. 2.5. The outputs of four buffers are connected together to form a single bus line. (It must be realized that this type of connection cannot be done with gates that do not have three-state outputs.) The control inputs to the buffers determine which of the four normal inputs will communicate with the bus line. No more than one buffer may be in the active state at any given time. The connected buffers must be controlled so that only one three-state buffer has access to the bus line while all other buffers are maintained in a high impedance state. One way to ensure that no more than one control input is active at any given time is to use a decoder, as shown in the diagram. When the enable input of the decoder is 0, all of its four outputs are 0, and the bus line is in a high-impedance state because all four buffers are disabled. When the enable input is active, one of the three-state buffers will be active, depending on the binary value in the select inputs of the decoder. Careful investigation will reveal that Fig. 2.5 is another way of constructing a 4×1 multiplexer since the circuit can replace the multiplexer in Fig. 2.3. To construct a common bus for four registers of n bits each using three-state buffers, we need n circuits with four buffers in each as shown in Fig. 2.5. Each group of four buffers receives one significant bit from the four registers. Each common output produces one of the lines for the common bus for a total of n lines. Only one decoder is necessary to select between the four registers.

2.4.2 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 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

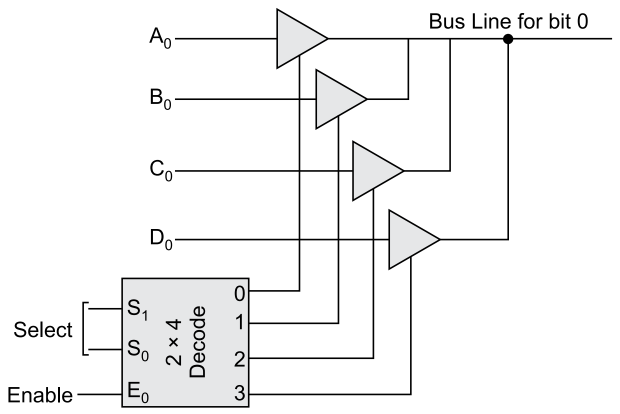


Fig. 2.5 Bus line with three-state buffers.

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 memory 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 memory write operation transfers the content of a register R1 to a memory word M selected by the address in address AR. The notation is:

Write: M [AR] ← R1

2.5 MICRO-OPERATIONS

A micro-operation is an elementary operation performed with the data stored in registers. Following are the four categories of micro-operations:

(i) Register transfer micro-operations: Transfer binary information from one register to another.

(ii) Arithmetic micro-operations: Perform arithmetic operations on data stored in registers.

(iii) Logic micro-operations: Perform bit manipulation operations.

(iv) Shift micro-operations: Perform shift operations.

(v) The register transfer micro-operation has already been discussed. Other three will be discussed here. Also, register transfer micro-operation does not change the information content when information moves from source to destination register, while other three micro-operations change the content during operation.

2.5.1 Arithmetic Micro-operations

The basic arithmetic micro-operations are:

1. Addition
2. Subtraction
3. Increment
4. Decrement
5. Arithmetic shift

The increment and decrement micro-operations are implemented with a combinational circuit or with a binary up-down counter as these micro-operations use plus-one and minus-one operation respectively.

Two basic arithmetic operations (multiplication and divide) are not included in the basic set of micro-operations. They are implemented by means of a combinational circuit. In general, the multiplication micro-operation is implemented with a sequence of add and shift micro-operations. Division is implemented with a sequence of subtract and shift micro-operations.

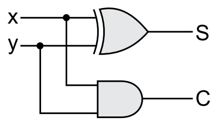
Binary adder

To implement the add micro-operation with hardware, we need the registers that hold the data and the digital component that performs the arithmetic addition.

The digital circuit which performs the addition of two bits and produces a sum and a carry as output is called half adder (Fig. 2.6). The digital circuit that forms the arithmetic sum of two bits and a previous carry is called a full-adder (Fig. 2.7). The digital circuit that generates the arithmetic sum of two binary numbers of any lengths 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. Figure 2.6 shows the interconnections of full-adders (FA) to provide any n-bit binary adder. The augends 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 C_0 and the output carry is C_n. The S outputs of the full-adders generate the required sum bits.

x	y	c	s
0	0	0	0
0	1	0	0
1	0	0	0
1	1	1	1

(a) Truth Table



(b) Logic Diagram

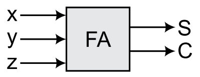
Fig. 2.6 Half-adder circuit.

Binary adder-subtractor

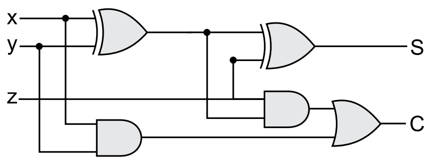
The subtraction of binary numbers can be done most conveniently by means of complementing one of the numbers. The subtraction A – B can be done by taking the 2's complement of B and adding it to A. The 2's complement can be obtained by taking the l's complement and adding one to the least significant pair of bits. The l's complement can be implemented with inverters and a one can be added to the sum through the input carry (Fig. 2.9). 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

Inputs			Outputs	
x	y	c	c	s
0	0	0	0	0
0	0	0	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

(a) Truth Table



(c) Block Diagram



(c) Logic Diagram

Fig. 2.7 Full-adder circuit.

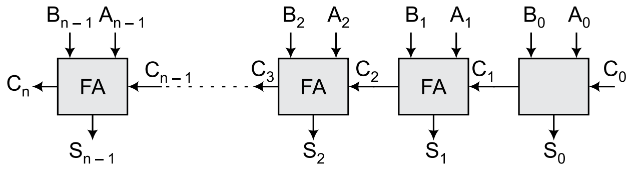


Fig. 2.8 n-bit binary adder.

adder-subtractor circuit is shown in Fig. 2.10. The mode input M controls the operation. When M = 0 the circuit is an 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 B ⊕ 0 = B. The full-adders 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 C_0 = 1. The B inputs are all complemented and a 1 is added through the input carry. The circuit performs the operation A plus the 2's complement of B. For unsigned numbers,

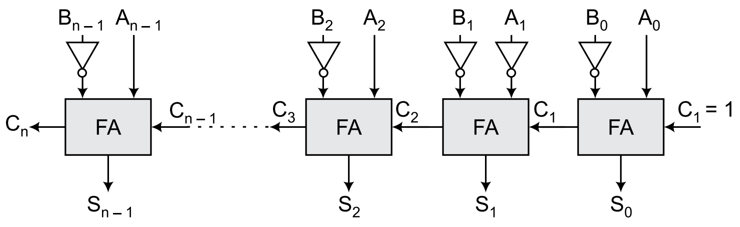


Fig. 2.9 n-bit binary subtractor.

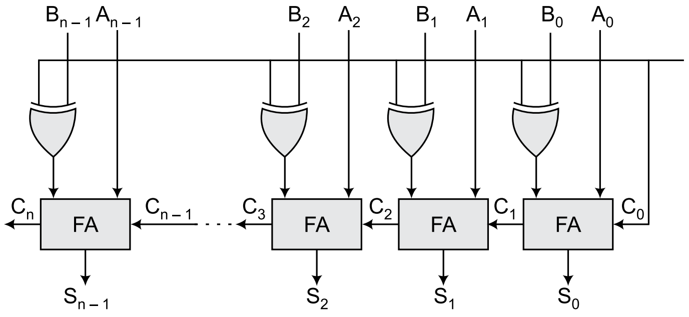


Fig. 2.10 n-bit adder-subtractor.

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.

Binary incrementer

The increment micro-operation adds one to a number in a register. For example, if a 4-bit register has a binary value 0110, it will go to 0111 after it is incremented. This micro-operation is easily implemented with a binary counter or by means of half-adders (Fig. 2.6) connected in cascade. The diagram of a 4-bit combinational circuit incrementer is shown in Fig. 2.11. One of the inputs to the least significant half-adder (HA) is connected to logic-1 and the other input is connected to the least significant bit of the number to be incremented. The output carry from one half-adder is connected to one of the inputs of the next-higher-order half-adder. The circuit receives the four bits from A_0 through A_3 adds one to it, and generates the incremented output in S_0 through S_3. The output carry C, will be 1 only after incrementing binary 1111. This also causes outputs S_0 through S_3 to go to 0.

The circuit of Fig. 2.11 can be extended to an n-bit binary incrementer by extending the diagram to include n half-adders. The least significant bit must have one input connected to logic-1. The other inputs receive the number to be incremented or the carry from the previous stage.

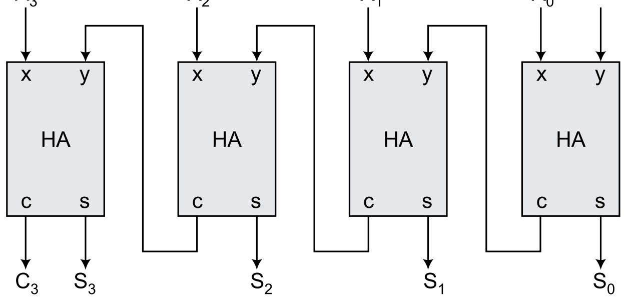


Fig. 2.11 4-bit binary incrementer.

Arithmetic micro-operation circuit

The arithmetic micro-operations listed in Table 2.3 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. 2.12. 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 is 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, S, and S_0. The input carry C_{in} goes to the carry input of the FA in the least significant position. The other carries are connected from one stage to the next. The output of the binary adder is calculated from the following arithmetic sum:

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

where A is the 4-bit binary number at the X inputs and Y is the 4-bit binary number at the Y inputs of the binary adder. C_{in} is the input carry, which can be equal to 0 or 1. Note that the symbol + in the equation above denotes an arithmetic plus. By controlling the value of Y with the two selection inputs S_1 and S_0 and making C_{in} equal to 0 or 1, it is possible to generate the eight arithmetic micro-operations listed in Table 2.4.

Table 2.3 Some arithmetic micro-operations.

Micro-operation	*Symbol*	*Description*
Add	$R3 \leftarrow R2 + R1$	Content of R1 and R2 are added and result is stored in R3
Subtract	$R3 \leftarrow R1 - R2$	Subtract the content of R2 from the contents of R1 and the result is stored in R3. This operation may also be written as $R3 \leftarrow R1 + + 1$
1's complement	$R2 \leftarrow \overline{R2}$	Complements the content of R2(1's) and store it in R2.
2's complement	$R2 \leftarrow \overline{R2} + 1$	2's complement the content of R2
Increment	$R1 \leftarrow R1 + 1$	Increment the content of R1 by 1
Decrement	$R1 \leftarrow R1 - 1$	Decrement the content of R1 by 1.

When $S_1S_0 = 00$, the value of B is applied to the Y inputs of the adder. If $C_{in} = 0$, the output $D = A + B$. If $C_{in} = 1$, output $D = A + B + 1$. Both cases perform the add micro-operation with or without adding the input carry.

When $S_1S_0 = 01$, the complement of B is applied to the Y inputs of the adder. If $C_{in} = 1$, then $D = A + B' + 1$. This produces A plus the 2's complement of B, which is equivalent to a subtraction

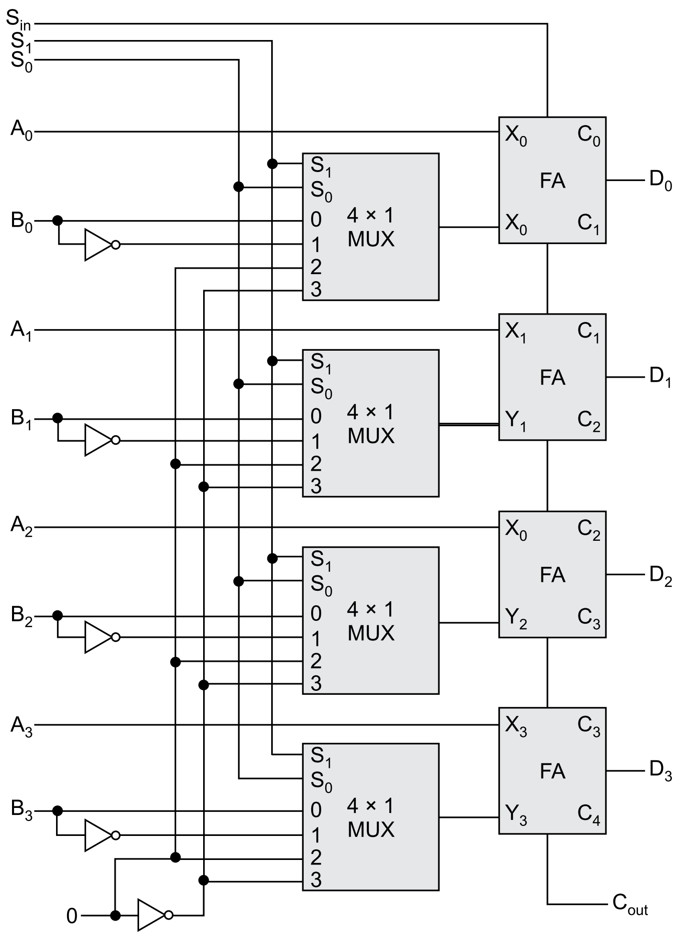


Fig. 2.12 4-bit arithmetic circuit.

Table 2.4 Arithmetic circuit function table

Select			*Input*	*Output*	*Micro-operation*
S_1	S_0	C_{in}	*Y*	$D = A + Y + C_{in}$	
0	0	0	B	D = A + B	ADD
0	0	1	B	D = A + B + 1	ADD WITH CARRY
0	1	0	B*	D = A + B*	SUBTRACT WITH BORROW
0	1	1	B*	D = A + 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

of A – B. When C_{in} = 0, then D = A + B. This is equivalent to a subtract with borrow, that is, A – B – 1.

When S_1S_0 = l0, the inputs from B are neglected, and instead, all D's are inserted into the Y inputs. The output becomes D = A + 0 + $C_{in.}$ This gives D = A when C_{in} = 0 and D = A + 1 when C_{in} = 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 S_1S_0 = 11, all l's are inserted into the Y inputs of the adder to produce the decrement operation D = A – 1 when C_{in} = 0. This is because a number with all l'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 C_{in} = 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.

2.5.2 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

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

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

1010 Content of R1

$\underline{1100}$ Content of R2

0110 Content of R1 after 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 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 l's 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. Another reason for adopting two sets of symbols is to be able to distinguish the symbol +, when used to symbolize an arithmetic plus, from a logic OR operation. Although the + symbol has two meanings, it will be possible to distinguish between them by noting where the symbol occurs. When the symbol + occurs in a micro-operation, it will denote an arithmetic plus. When it occurs in a control (or Boolean) function, it will denote an OR operation. We will never use it to symbolize an OR micro-operation. For example, in the statement

$$P + Q: R1 \leftarrow R2 + R3, R4 \leftarrow R5 \vee R6$$

Logic micro-operations specify binary operations on the strings of bits in registers. These are bit-wise operations, i.e., they work on individual bits of data. In principle, there are 16 different logic operations that can be defined over two binary variables. They can be determined from all possible truth tables obtained with two binary variables as shown in Table 2.5. 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. Note that the functions are determined from the 16 binary combinations that can be assigned to F.

Table 2.5 Truth table for 16 functions of two variables

x	*y*	F_1	F_2	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	1	1	1	1	1	1	1	1	1
0	1	0	0	0	0	1	1	1	0	0	0	0	0	1	1	1	1
1	0	0	0	1	1	0	1	1	0	0	0	1	1	0	0	1	1
1	1	0	1	0	1	0	0	1	0	1	1	0	1	0	1	0	1

The 16 Boolean functions of two variables x and y are expressed in algebraic form in the first column of Table 2.6. 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 2.6 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.

Table 2.6 Logic micro-operations

Boolean Function	*Micro-operation*	*Name*
$F_0 = 0$	$F \leftarrow 0$	Clear
$F_1 = XY$	$F \leftarrow A \wedge \overline{B}$	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

Contd...

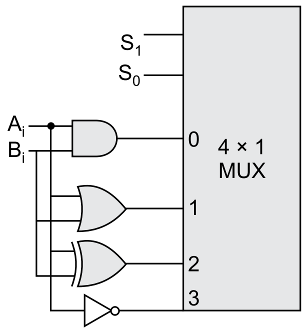
Contd...

$F_7 = X + Y$	$F \leftarrow A \vee B$	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
$F_{15} = 1$	$F \leftarrow$ a11 1's	Set to All 1'S

Hardware implementations

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 2.13 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 S_1 and S_0 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 Fig. 2.13(b) lists the logic micro-operations obtained for each combination of the selection variables.



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

Fig. 2.13 Logic circuit.

Applications

One application of logic micro-operations is 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. Some of these applications are:

Table 2.7 Application of logic micro-operations

Selective–set	Sets (i.e., placing 1) the required bits
Selective-complement	Complement the required bits
Selective–clear	Clears (i.e., placing 0) the required bits
Mask	Masks the required bits
Insert	Insert a new value
Clear	Produces 0’s

Selective set

The selective-set operation sets to 1 the bits in register A where there are corresponding l’s in register B. It does not affect bit positions that have 0’s in B.

x x x x x x x x x (any bit string in register)

$\underline{1\ 1\ 0\ 1\ 1\ 0\ 0\ 0\ 0}$ (contents of register B)

1 1 x 1 1 x x x x

Since OR operation results in logic 1 when one of the inputs is 1, therefore, the OR micro-operation can be used to selectively set the bits of a register.

Selective complement

The operation complements the bits of register A where there are corresponding l’s in B. It does not affect bit positions of register A corresponding to 0’s in register B.

1 1 0 0 1 1 0 0 (content of register A)

$\underline{1\ 1\ 1\ 1\ 0\ 0\ 0\ 0}$ (content of register B)

0 0 1 1 1 1 0 0 (Register A after operation)

Since, Ex-OR micro-operation complements the input, when one of the input is logic 1, therefore, Ex-OR micro-operation can be used to selectively complement bits of a register.

Selective-clear

The selective-clear operation clears to 0 the bits in A only where there are corresponding l’s in B. There is no change in bit position of register A corresponding to 0’s on register B.

1 0 1 0 1 0 1 0 (content of register A)

$\underline{1\ 1\ 1\ 1\ 0\ 0\ 0\ 0}$ (content of register B)

0 0 0 0 1 0 1 0 (content of register A after operation)

This operation can be performed using AND operation with complemented B input. Therefore, the corresponding logic micro-operation is

$$A \leftarrow A \wedge B'$$

Mask

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.

1 0 1 0 1 0 1 0 (content of register A)

$\underline{1\ 1\ 1\ 1\ 0\ 0\ 0\ 0}$ (content of register B)

1 0 1 0 0 0 0 0 (content of register A after operation)

The mask operation can be performed by using AND operation. This operation is more convenient to use than the selective-clear operation because AND micro-operation is frequently used than the micro-operation used for selective-clear.

Insert

The *insert* operation inserts a new value into a group of bits. This is done by first masking the bits and then ORing them with the required value.

0 1 1 0 1 0 1 0 (content of register A)

$\underline{1\ 1\ 1\ 1\ 0\ 0\ 0\ 0}$ (content of register B)

0 0 0 0 1 0 1 0 (content of register A after masking operation)

Now, insert a new value

1 0 1 0 0 0 0 0 (content of register A)

$\underline{1\ 0\ 0\ 1\ 0\ 0\ 0\ 0}$ (content of register B for insert)

1 0 0 1 1 0 1 0 (content of register A after insertion)

Clear

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 micro-operation.

1 0 1 0 (content of register A)

$\underline{1\ 0\ 1\ 0}$ (content of register B for insert)

0 0 0 0 (content of register A after operation)

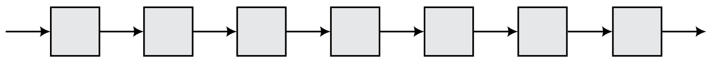
2.5.3 Shift Micro-operations

Shift micro-operations shift the contents of a register either left or right. These micro-operations are generally used for serial transfer of data. They are also used along with arithmetic, logic, and other data-processing operations.

When 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.

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:


(a) Logical shift right

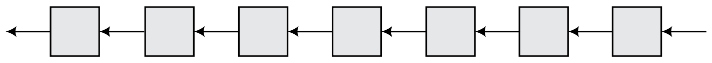
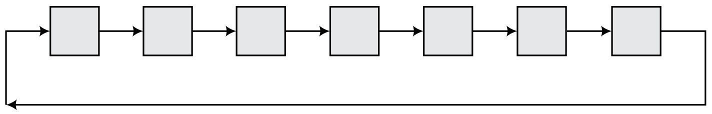

(b) Logical shift left

Fig. 2.14 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.


(a) Circular shift right

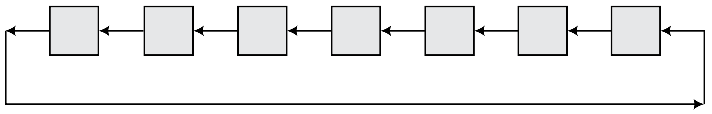

(b) Circular shift left

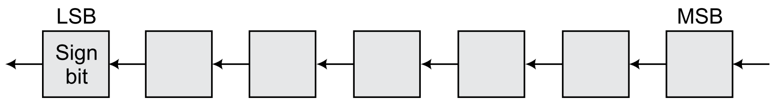
Fig. 2.15 Circular shift.

Arithmetic shift

An arithmetic shift micro-operation shifts a signed binary number to the left or right. Notations used for arithmetic shift left and arithmetic shift right is ashl and ashr respectively.

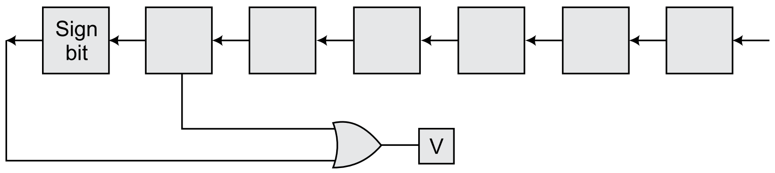
An arithmetic shift left multiplies a signed binary number by 2 and an arithmetic shift right divides a signed binary number by 2. The leftmost bit in a register holds the sign bit. Both shifts must leave the sign bit unchanged because sign of the number remains the same when it is multiplied/divided by 2.

This micro-operation inserts a 0 into LSB and shifts all other bits to the left. The initial bit of MSB is lost and replaced by the immediate left of it.



(a) Arithmetic shift left

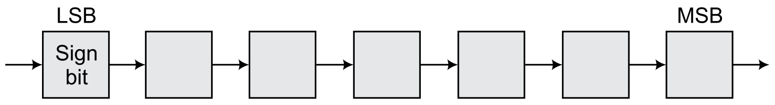
An arithmetic shift-left operation must be checked for overflow. Before the shift, if the leftmost two bits differ, the shift will result in an overflow.



(b) Arithmetic shift left with overflow

i.e., if V = 0, there is no overflow, but if V = 1, there is an overflow. In this case, a sign reversal occurs.

Arithmetic shift-right micro-operation leaves the sign bit unchanged and shift the number (including the sign bit) to the right. Thus, sign bit remains the same, MSB receives the bit from sign bit, and so on for other bits. The bit in LSB is lost.



(c) Arithmetic shift right

Fig. 2.16 Arithmetic shift.

Hardware implementation

These micro-operations can be implemented either by bidirectional shift register with parallel load or by combinational circuit shifter with multiplexers (see Fig. 2.17). Using bidirectional shift register, 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.

Table 2.9 Function table of arithmetic logic shift unit

Operation Select					*Operation*	*Function*
S_3	S_2	S_1	S_0	C_{in}		
0	0	0	0	0	F = A	Transfer A
0	0	0	0	1	F = A + 1	Inerement 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 + B*	Subtract with borrow
0	0	1	0	1	F = A + B* + 1	Subtraction
0	0	1	1	0	F = A – 1	Decrement A
0	0	1	1	0	F = A	Transfer A
0	1	0	0	X	F = A Ù* B	And
0	1	0	1	X	F = A Ú* B	or
0	1	1	0	X	F = A Å* B	X or
0	1	1	0	X	F = A*	Complement A
1	0	X	X	X	F = Snr A	Shift right A into F
1	1	X	X	X	F = Snl A	Shift left A into F

Solved Examples

Example 1: The outputs of two registers R_0 and R_1 are connected through 2 × 1 multiplexers to the input of the third register R_2. Each register is 4-bit wide and the required transfers are controlled by the two timings T_0 and T_1 as follows:

$$T_0: R_2 \leftarrow R_0$$

The timing variables are mutually exclusive, (only one variable is active = 1 at any given time). Draw the block diagram showing the hardware implementation of the register transfers. Include the connections necessary from the two timing variables to the selection inputs of the multiplexers and to the load input of register R_2.

Solution: Since there are two registers from which to select so the multiplexers are 2 ×1, but the registers are 4-bit wide so we need four 2 × 1 multiplexers. To select from one of the two registers R_0 and R_1, we need one selection variable S.

The block diagram and truth table is as follows:

Example 2: A digital computer has a common bus system for 8 registers of 32bits each. The bus is controlled with multiplexers

(i) How many selection inputs are there in each multiplexer?

(ii) What sizes of multiplexers are needed?

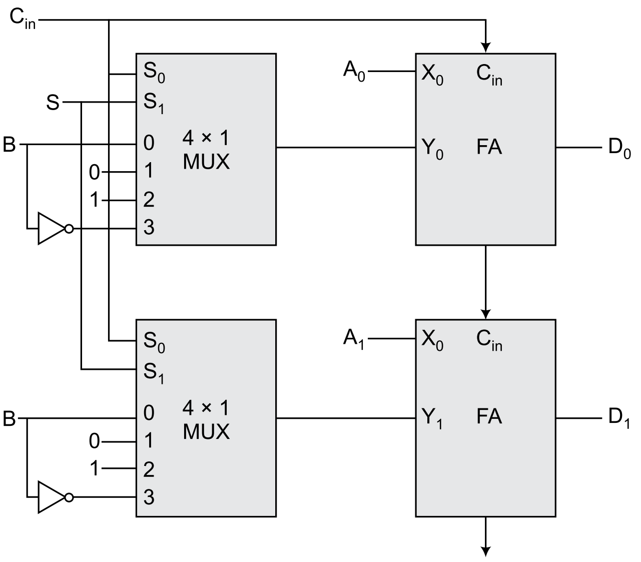
Example 4: Design an arithmetic circuit with one selection variable S and two n-bit data inputs A and B. The circuit generates the following four arithmetic operations in conjunction with carry input C_{in}. Draw the logic diagram for the first two bits.

S	C_{in}	Operation
0	0	D = A + B (add)
0	1	D = D + 1 (increment)
1	0	D = A – 1 (decrement)
1	1	D = A + $\overline{B}$ + 1 (subtract)

Solution:

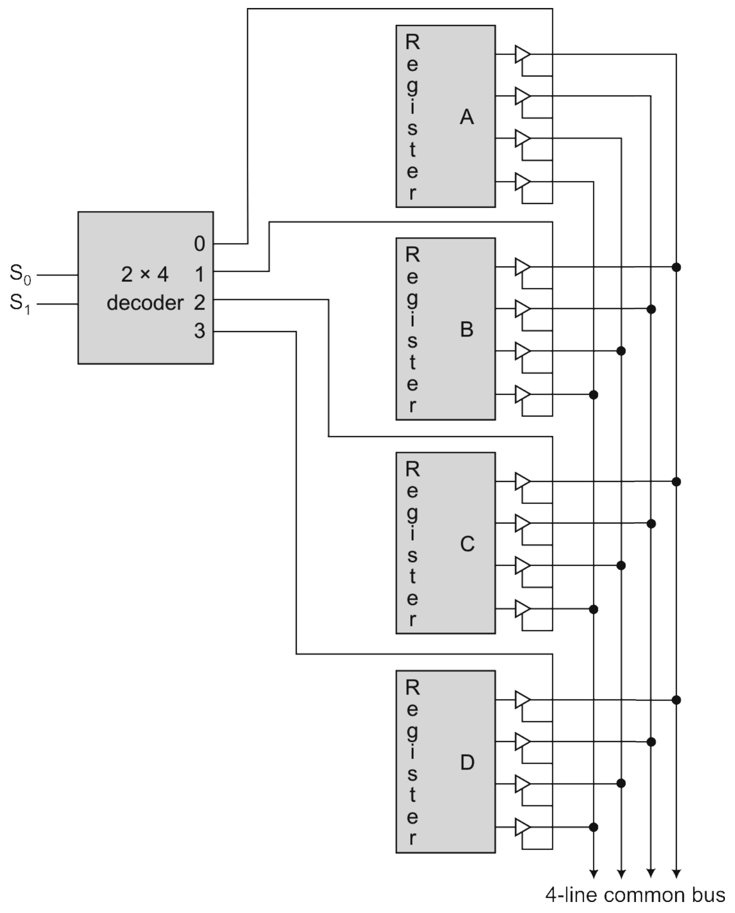
		Inputs to FA		*Operation*
S	*C_{in}*	*X*	*Y*	
0	0	A	B	A + B
0	1	A	0	A + 1
1	0	A	1	A – 1
1	1	A	$\overline{B}$	A – B

The combinational circuit for the above is



Example 5: Using Tri-state buffers and the decoder construct a bus system to transfer information from four registers. Each register is 4-bit wide.

Solution: To select one of the four registers we need two selection inputs for a 2 × 4 decoder. The Tri-state buffer bus system is



Example 6: If the register A holds binary value 11011001. Determine the value of register B and the logic micro-operation to be performed in order to change the value of A to:

(i) 01101101

(ii) 11111101

Solution: (i) A = 11011001

B = 10110100

A ← A XOR B = 01101101

(ii) A = 11011001

B = 11111101

A = ← A OR B = 11111101

Example 7: How many multiplexers are required to design a common bus architecture using multiplexer? Also find the size of each multiplexer. There are total 32 registers of 8 bits.

Solution: Number of multiplexers = number of bits

= 8

Therefore, total 8 number of multiplexers are required.

Size of multiplexer = number of registers: 1

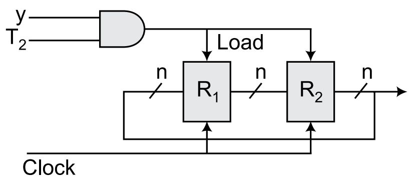
32:1

Hence, the size of each multiplexer required is 31:1

Example 8: Show the block diagram of the hardware that implements the following register transfer statements.

$$yT_2: R_2 \leftarrow R_1. R_1 \leftarrow R_2$$

Solution:

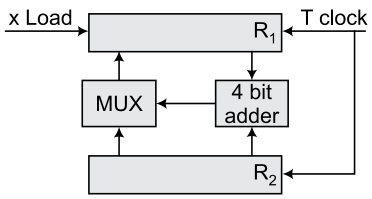


Example 9: Consider the following register transfer statements

$$xT: R1 \leftarrow R1 + R1 \text{ (if } x = 1); xT: R1 \leftarrow R2 \text{ (if } x = 0)$$

where T represents clock. Draw a diagram showing the hardware implementation of the above statements.

Solution:



Block diagram for transfer statement

$$xT: R1 \leftarrow R1 + R1 \text{ (if } x = 1); xT: R1 \leftarrow R2 \text{ (if } x = 0)$$

Example 10: The following transfer statements specify a memory. Explain the memory operation in each case

(a) $R_2 \leftarrow M[AR]$

(b) $M[AR] \leftarrow R_3$

(c) $R_5 \leftarrow M[R_5]$

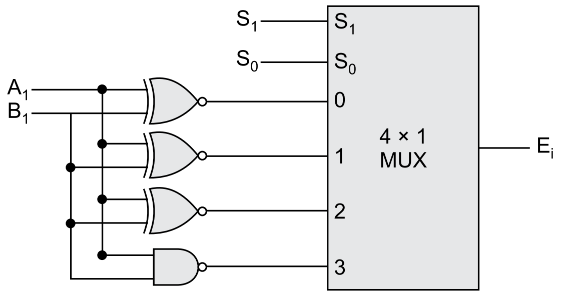
Solution: (a) Read memory word specified by the address in AR into register R_2.

(b) Write content of register R_3 into the memory word specified by the address in AR.

(c) Read memory word specified by the address in R_5 and transfer content to R_5 (destroys previous value).

Example 11: Design a digital circuit that performs the four logic operations of exclusive OR, exclusive NOR, NOR and NAND. Use two selection variables. Show the logic diagram of one typical stage.

Solution:



Example 12: A 8-bit register AR, BR, CR and DR initially have the following values.

AR = 11110010

BR = 11111111

CR = 10111001

DR = 11101010

Determine the 8-bit values in each register after the execution of the following sequence of micro-operations

$AR \leftarrow AR + BR$	Add AR to BR
$CR \leftarrow CR$ AND DR, $BR \leftarrow BR + 1$	AND DR to CR, increment BR
$AR \leftarrow AR - CR$	Subtract CR from AR

Solution: (a) AR = 11110010

BR = 11111111 (+)

AR = 11110001 BR = 11111111 CR = 10111001 DR = 11101010

```
(b) CR = 10111001                      BR = 11111111
    DR = 11101010 (AND)                          +1
    --------------------               --------------------
(c) CR = 10101000                      BR = 00000000
    AR = 11110001                      DR = 11101010
    AR = 11110000 (–7)
    CR = 10101000
    --------------------
    AR = 01001001   BR = 00000000      CR = 10101000   DR = 11101010
```

Review Questions

1. Define the following terms:
   (a) Register transfer language
   (b) Micro-operations
   (c) Bus Transfer
   (d) Memory Transfer
   (e) Control Function
2. Can two or more operations be executed at the same time? If yes, explain.
3. Give the construction of a bus system for eight registers. If each register has 8bits. How many multiplexers are used?
4. Give the construction of binary adder/subtractor.
5. Differentiate between:
   (a) Selective set vs selective complement
   (b) Selective clear vs masking'
   (c) Logical shift vs arithmetic shift
   (d) Add vs add with carry micro-operation
6. Describe the design of 4-bit carry-look-ahead adder.
7. Show the block diagram of the hardware that implements the following register transfer statement:

   $$T: R2 \leftarrow R1, R1 \leftarrow R2$$

8. Design a 4-bit adder subtractor with example.
9. What do you mean by high speed adder? Discuss design of high speed adders.
10. Represent the following conditional control statement by two register transfer statements with control functions.

    If (P = 1) then (R1 $\leftarrow$ R2) else if (Q = 1) then (R1 $\leftarrow$ R3)
11. A digital computer has a common bus system for 16 registers of 32 bits each. The bus is

constructed with multiplexers.

(a) How many selection inputs are there in each multiplexer?

(b) What size of multiplexers are needed?

(c) How many multiplexers is there in the bus?

12. The following transfer statements specify a memory. Explain the memory operation in each case.

(a) R2 ← M[AR] (b) M[AR) ← R3

(c) R5 ← M[R5]

13. Draw the block diagram for the hardware that implements the following statements:

$$x + yz\text{: } AR \leftarrow AR + BR$$

where AR and BR are two n-bit registers and *x, y,* and z are control variables. Include the logic gates for the control function. (Remember that the symbol + designates an OR operation in a control or Boolean function but that it represents an arithmetic plus in a micro-operation.)

14. Show the hardware that implements the following statement. Include the logic gates for the control function and a block diagram for the binary counter with a count enable input.

$$xyT_0 + T_1 + y'T_2\text{: } AR \leftarrow AR + 1$$

15. Consider the following register transfer statements for two 4-bit registers R1 and R2.

$$xT\text{: } R1 \leftarrow R1 + R2$$

$$x'T\text{: } R1 \leftarrow R2$$

Every time that variable $T = 1$, either the content of R2 is added to the content of R1 if $x = 1$, or the content of R2 is transferred to R1 if $x = 0$. Draw a diagram showing the hardware implementation of the two statements. Use block diagrams for the two 4-bit registers, a 4-bit adder, and a quadruple 2-to-l line multiplexer that selects the inputs to R1. In the diagram, show how the control variables X and T select the inputs of the multiplexer and the load input of register R1.

16. Design a 4-bit combinational circuit decrementer using four full-adder circuits.

GATE Questions

1. The register which keeps track of the execution of a program and which contains the memory address of the instruction currently being executed is called

(a) Index register (b) Memory address register

(c) Program counter (d) Instruction register

2. The register which holds the address of the location to or from which data are to be transferred is called

(a) Index register (b) Instruction register

(c) Memory address register (d) Memory data register

3. The register which contains the data to be written into or read out of the addressed location is called
   (a) Memory address register (b) Memory data register
   (c) Program computer (d) Index register
4. Which of the following is used as storage location both in the ALU and the control section of a computer?
   (a) Accumulator (b) Register (c) Adder (d) Decoder
5. A single bus structure is primarily found in
   (a) Mainframes (b) Supercomputers
   (c) High-performance machines (d) Mini-and microcomputers
6. The unit of a computer system which executes program, communicates with and often controls the operation of other subsystems of the computer is the
   (a) CPU (b) Control unit (c) Flow unit (d) Peripheral unit
7. The ability of a medium-sized computer system to increase in data processing capability by addition of such devices as mass storage device, I/O devices, etc. is called
   (a) Computer expandability (b) Computer mobility
   (c) Computer enhancement (d) Computer upward capability
8. The technique which repeatedly used the same block of internal storage during different stages of problem is called
   (a) Overlay (b) Overlapping (c) Swapping (d) Reuse
9. The register used as a working area in CPU is
   (a) Program counter (b) Instruction register
   (c) Instruction decoder (d) Accumulator
10. Which of the following is/are advantages of virtual memory?
    (a) Faster access to memory on an average.
    (b) Processes can be given protected address spaces.
    (c) Both (a) and (b)
    (d) Programs larger than the physical memory size can be run.
11. Arrange the following configurations for CPU in decreasing order of operating speeds: Hardwired control, vertical microprogramming, horizontal microprogramming
    (a) Hardwired control, vertical microprogramming, horizontal microprogramming
    (b) Hard-wired control, horizontal microprogramming, vertical microprogramming
    (c) Horizontal microprogramming, vertical microprogramming, hard-wired control
    (d) Vertical microprogramming, horizontal microprogramming, hard-wired control
12. The main difference between a CISC and RISC processor is/are that a RISC processor typically
    (a) Has fewer instructions and addressing modes
    (b) Has more registers