Register Transfer and Microoperations

By,
Er. Nabaraj Bahadur Negi

Contents

Microoperations

- A processor register (CPU register) is one of a small set of data holding places that are part of the computer processor.
- A register may hold an instruction, a storage address, or any kind of data (such as a bit sequence or individual characters).
- The operations on the data in registers are called microoperations.
- Operands can be in one or more registers.
- The functions built into registers are examples of microoperations
    - Shift, Load
    - Count, Clear
    - Increment

- The internal hardware organization of a digital computer is best defined by specifying:
    - The set of registers it contains and their function.
    - The sequence of microoperations performed on the binary information stored in the registers.
    - The control that initiates the sequence of microoperations.

Register transfer language

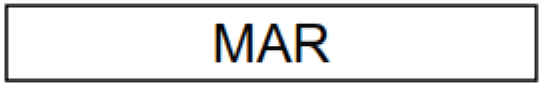
- The language, which is basically used to express the transfer of data among the registers, is called Register Transfer Language (RTL).
- It is the symbolic notation used to describe the microoperation transfers among registers.
- For any function of the computer, the register transfer language can be used to describe the (sequence of) microoperations
- Register transfer language
  - A symbolic language
  - A convenient tool for describing the internal organization of digital computers
  - Can also be used to facilitate the design process of digital systems.

Designation of registers

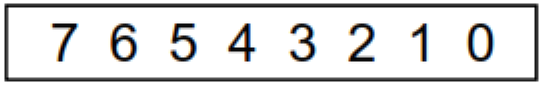
- Registers are designated by capital letters, sometimes followed by numbers (e.g., A, R13, IR)

- Often the names indicate function:
  - MAR - memory address register
  - PC - program counter
  - IR - instruction register

- Registers and their contents can be viewed and represented in *various ways*
  - A register can be viewed as a single entity:



  - Registers may also be represented showing the bits of data they contain



Register transfer

- Computer registers are designated by **capital letters.**
- 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).
- Copying the contents of one register to another is a register transfer.

- The most common way to represent a register is by a rectangular box with the name of the register inside. Fig (a).
- 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.

  For e.g. 8-bit register numbered: Fig (b).
- The numbering of bits in a 8-bit register can be marked on top of the box. Fig (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.

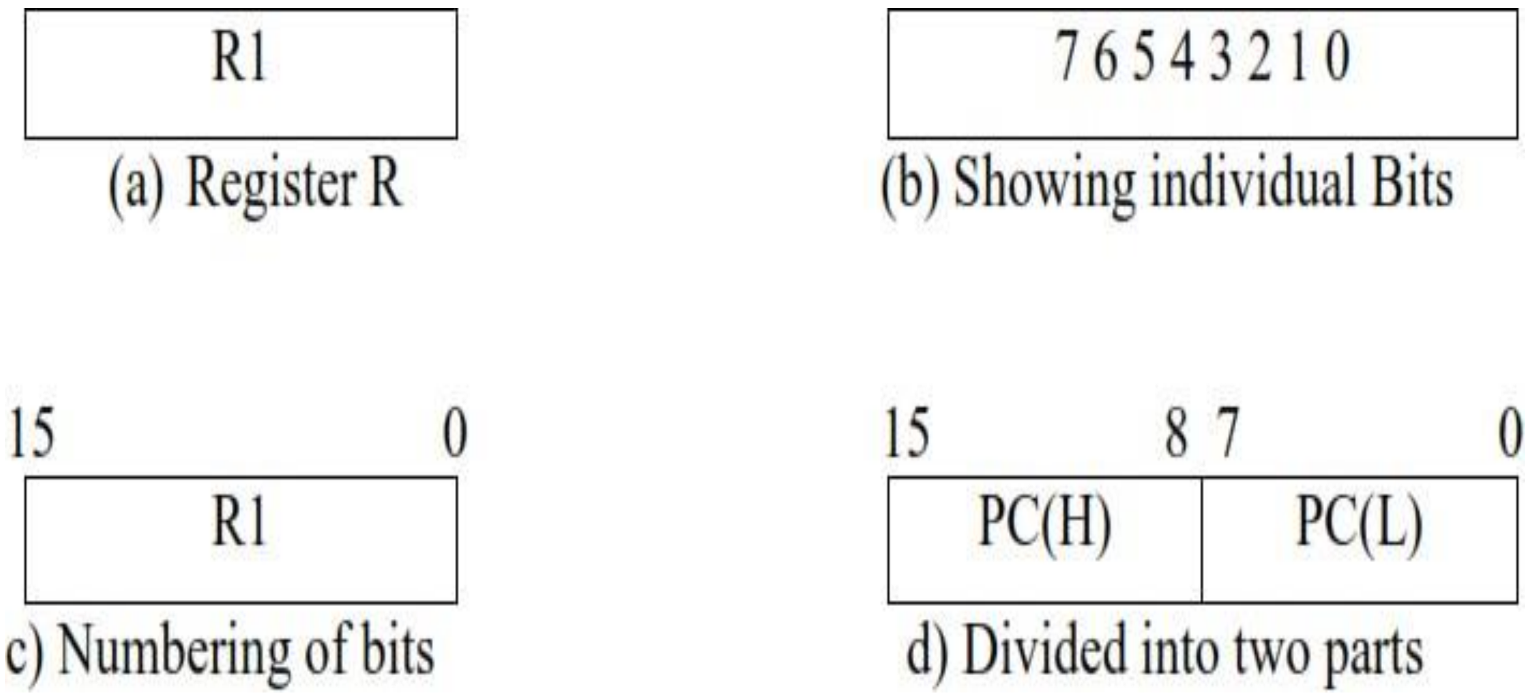


Figure : Block diagram of register

- A register transfer is indicated as

  $R2 \leftarrow R1$

  - In this case the contents of register R2 are copied (loaded) into register R1
  - A simultaneous transfer of all bits from the source R1 to the destination register R2, during one clock pulse
  - Note that this is a non-destructive; i.e. the contents of R1 are not altered by copying (loading) them to R2

- A register transfer such as

  $R3 \leftarrow R5$

  Implies that the digital system has

  - the data lines from the source register (R5) to the destination register (R3)
  - Parallel load in the destination register (R3)
  - Control lines to perform the action

Control Functions

- Often actions need to only occur if a certain condition is true
- This is similar to an “if” statement in a programming language
- In digital systems, this is often done via a control signal, called a control function

– If the signal is 1, the action takes place

- If there is predetermined control condition like

If (P=1) then (R2 $\leftarrow$ R1) then we can write the statement as

P: R2 $\leftarrow$ R1   where P is control signal usually a control function which is Boolean variable that is equal to 1 or 0.

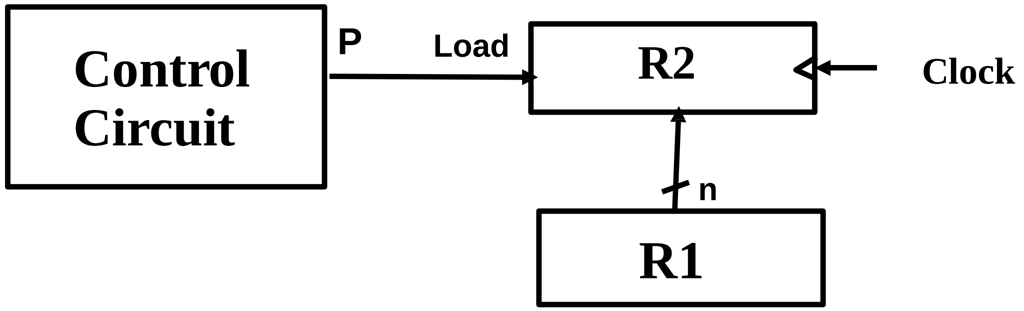
- Every statement written in a register transfer notation implies a hardware construction.
- Following Figure shows the block diagram that depicts the transfer

Hardware implementation of controlled transfers

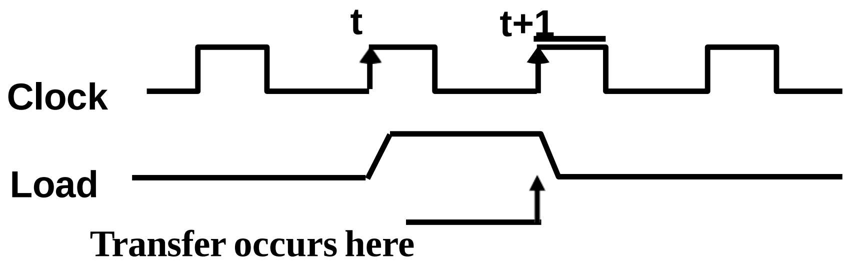
Implementation of controlled transfer

P: R2 $\leftarrow$ R1

Block diagram



Timing diagram



- The letter n will be used to indicate any number of bits for the register. It will be replaced by an actual number when the length of the register is known.
- Register R2 has a load input that is activated by the control variable. It is assumed that the control variable is synchronized with the same clock as the one applied to the register.
- As shown in the timing diagram P is activated in the control section by the edge of a clock pulse at time t.
- The next positive transition of the clock at time t + 1 finds the load input active and the data inputs of R2 are then loaded into the register in parallel.
- P may go back to 0 at time t + 1, otherwise, the transfer will occur with every clock pulse transition while P remains active

Simultaneous operations

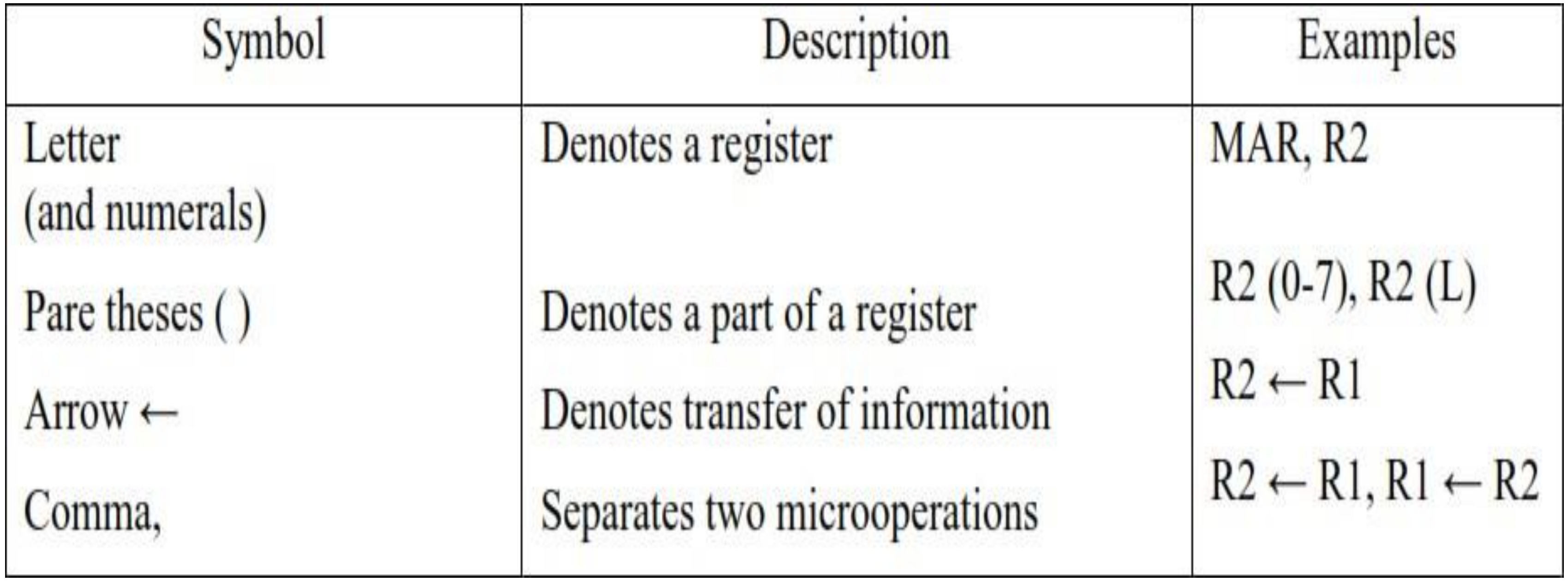
- Registers are denoted by capital letters, and numerals may follow the letters. Parentheses are used to denote a part of a register by specifying the range of bits or by giving a symbol name to a portion of a register. The arrow denotes a transfer of information and the direction of transfer.

- If two or more operations are to occur simultaneously, they are separated with commas

  $P: R3 \leftarrow R5, MAR \leftarrow IR$

- Here, if the control function P = 1, load the contents of R5 into R3, and at the same time (clock), load the contents of register IR into register MAR

Basic symbols for register transfers



Symbol	Description	Examples
Letter (and numerals)	Denotes a register	MAR, R2
Pare theses ( )	Denotes a part of a register	R2 (0-7), R2 (L)
Arrow ←	Denotes transfer of information	$R2 \leftarrow R1$
Comma,	Separates two microoperations	$R2 \leftarrow R1, R1 \leftarrow R2$

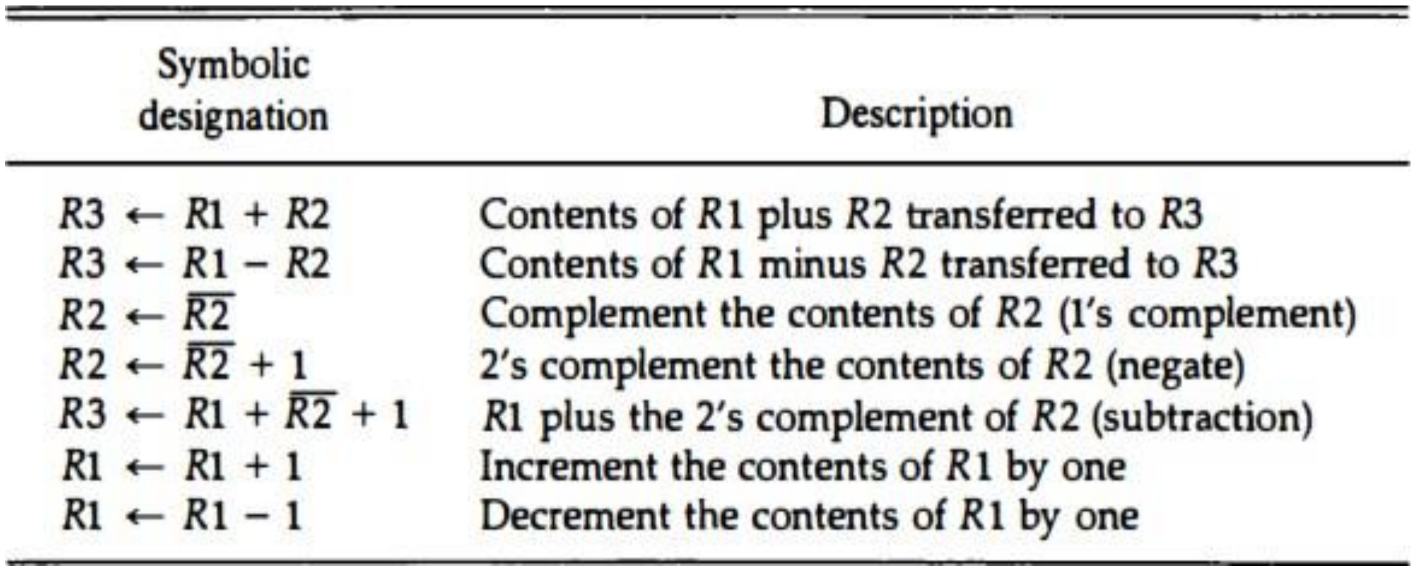
Computer system microoperations are of four types:

1. Register transfer microoperations
2. Arithmetic microoperations
3. Logic microoperations
4. Shift microoperations

Arithmetic microoperations

- The basic arithmetic microoperations are
  - Addition
  - Subtraction
  - Increment
  - Decrement
- The additional arithmetic microoperations are
  - Add with carry
  - Subtract with borrow
  - Transfer/Load
  - etc. …

Table: Summary of typical arithmetic microoperations



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

- These microoperations are implemented with a combinational circuit or with a binary up-down counter.
- The arithmetic operations of multiply and divide are not listed in table above.
- Multiply and divide operations are valid arithmetic operations but are not included in the basic set of microoperations. The only place where these operations can be considered as microoperations is in a digital system, where they are implemented by means of a combinational circuit.

Binary Adder

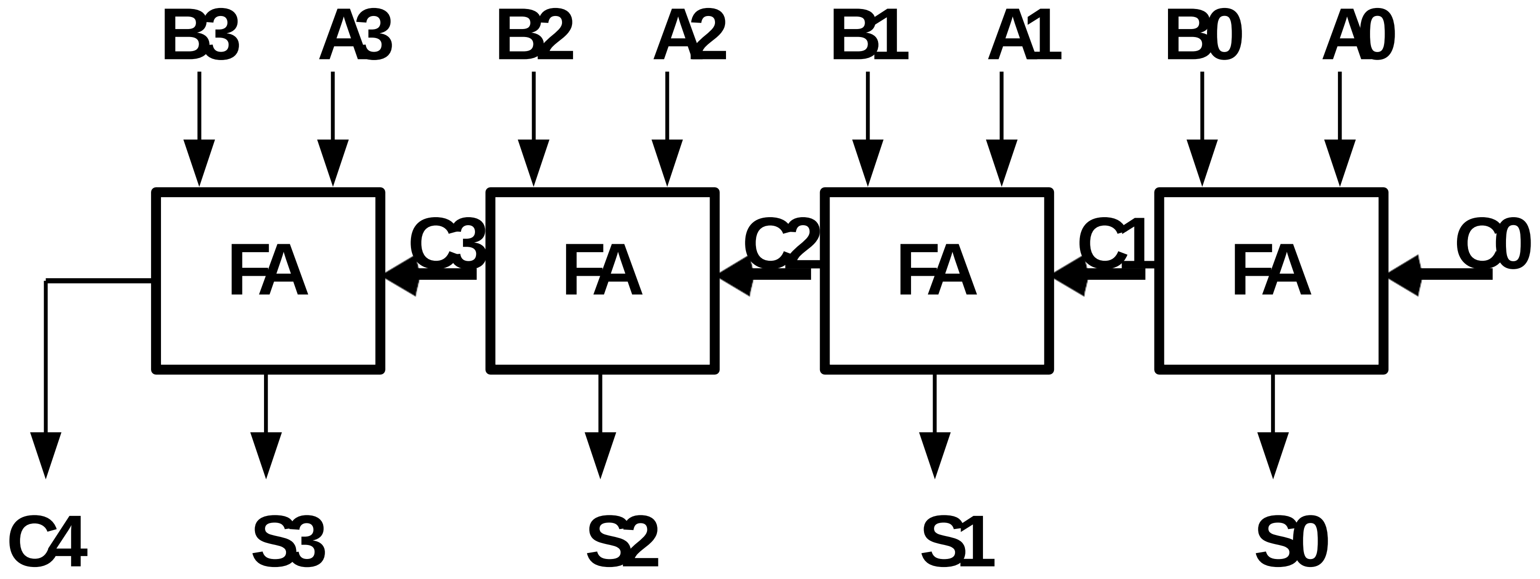


Figure : 4-bit binary adder.

- The registers that hold the data and the digital component that performs the arithmetic addition.
- The digital circuit that generates the arithmetic sum of two binary numbers of any length is called a binary adder.
- The binary adder is constructed with full-adder circuits connected in cascade, with the output carry from one full-adder connected to the input carry of the next full-adder.
- Figure shows the interconnections of four full-adders (FA) to provide a 4-bit binary adder.
- An n-bit binary adder requires n full-adders. The output carry from each full-adder is connected to the input carry of the next-high-order full-adder.
- The n data bits for the A inputs come from one register (such as R1), and the n data bits for the B inputs come from another register (such as R2).

Binary Adder-Subtractor

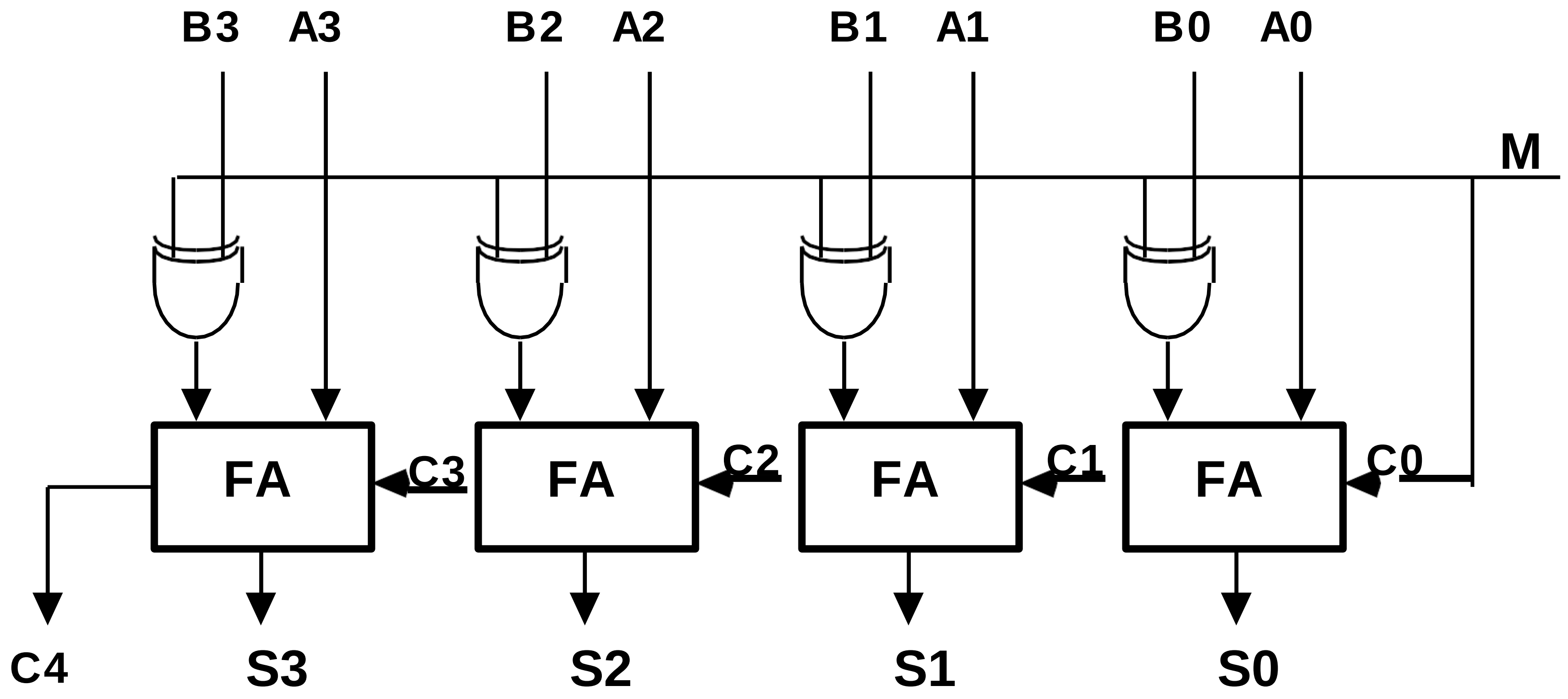


Figure : 4 -bit Adder-Subtractor

- The addition and subtraction operations can be combined into one common circuit by including an exclusive-OR gate with each full-adder.
- The mode input M controls the operation. When M = 0 the circuit is adder and when M=1 the circuit becomes a subtractor, Each exclusive-OR gate receives input M and one of the inputs of B.
- When M = 0, We have $B \oplus 0 = B$. The full-address receive the value of B, the input carry is 0 and the circuit performs Aplus B.
- When M=1 we have $B \oplus 1 = B'$ and Co = 1. The B inputs are all complemented and a 1 is added through the input carry.
- The circuit performs the operation A plus the 2‘s complement of B. for unsigned numbers, this gives A- B if A= B or the 2‘ s complement of (B-A) if A< B. For signed numbers, the result is A- B provided that there is no overflow.

Binary Incrementer

- The increment microoperation 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 can be accomplished by means of connected in cascade.
- Full adder implementation takes more time (has 2 Ex –OR , 2 AND and 1 OR gates) but Half adder takes less time(has 1 =Ex-OR and 1 AND gates).

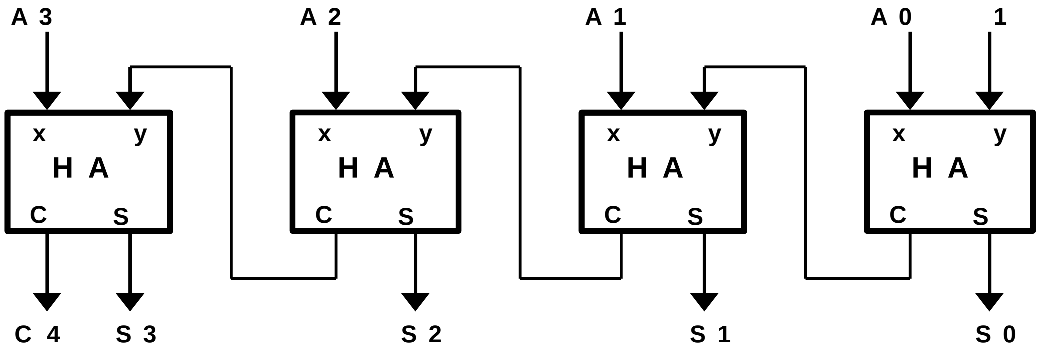


Figure : 4-bit binary incrementer.

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.

- The four multiplexers are controlled by two selection inputs S1 and S0. 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. By controlling the value of Y with the two selection inputs S1 and S0 and making C_{in} equal to 0 or 1, it is possible to generate the eight arithmetic microoperations 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	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

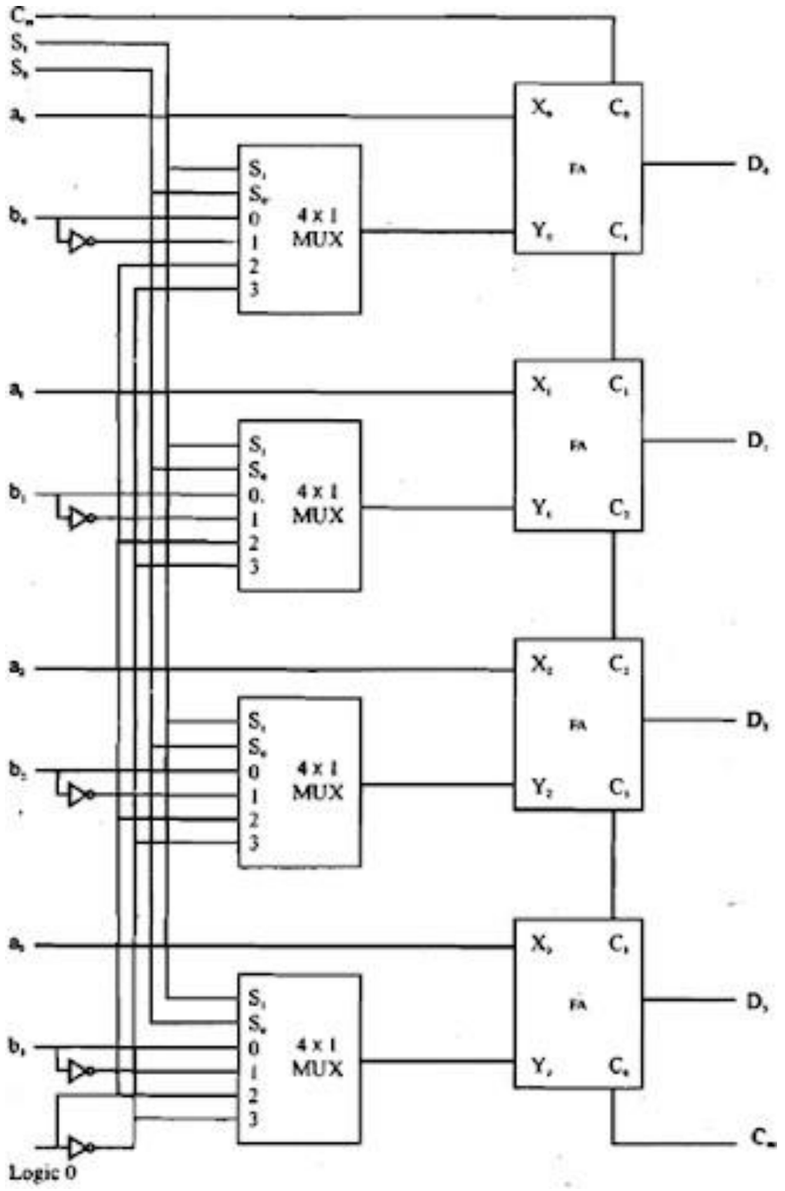
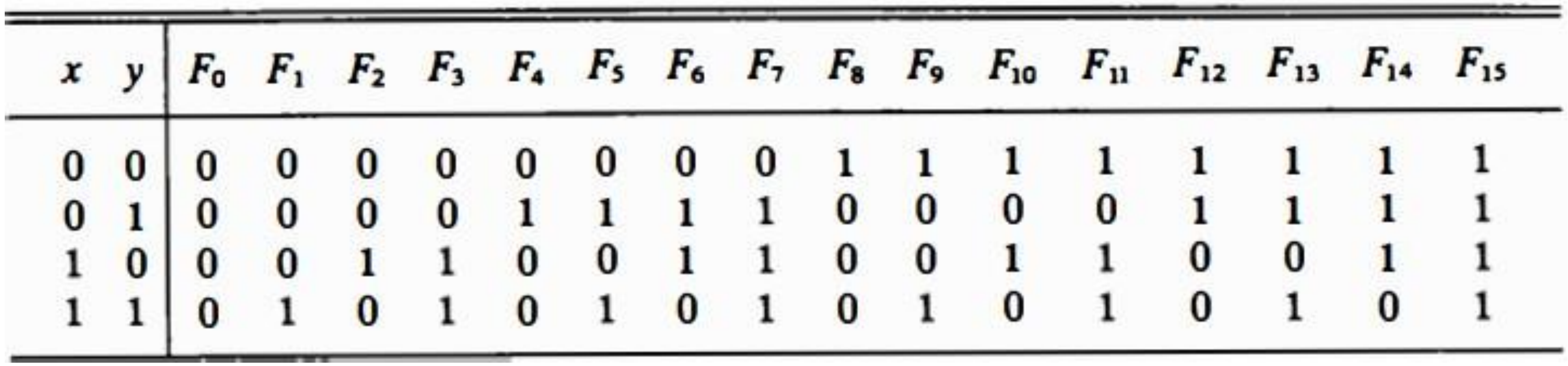


Figure : 4-bit Arithmetic Circuit

Logic microoperations

- Specify binary operations on the strings of bits in registers
  - Logic microoperations are bit-wise operations, i.e., they work on the individual bits of data
  - useful for bit manipulations on binary data
  - useful for making logical decisions based on the bit value
- There are, in principle, 16 different logic functions that can be defined over two binary input variables
- However, most systems only implement four of these
  - AND ($\wedge$), OR ($\vee$), XOR ($\oplus$), Complement/NOT
- The others can be created from combination of these

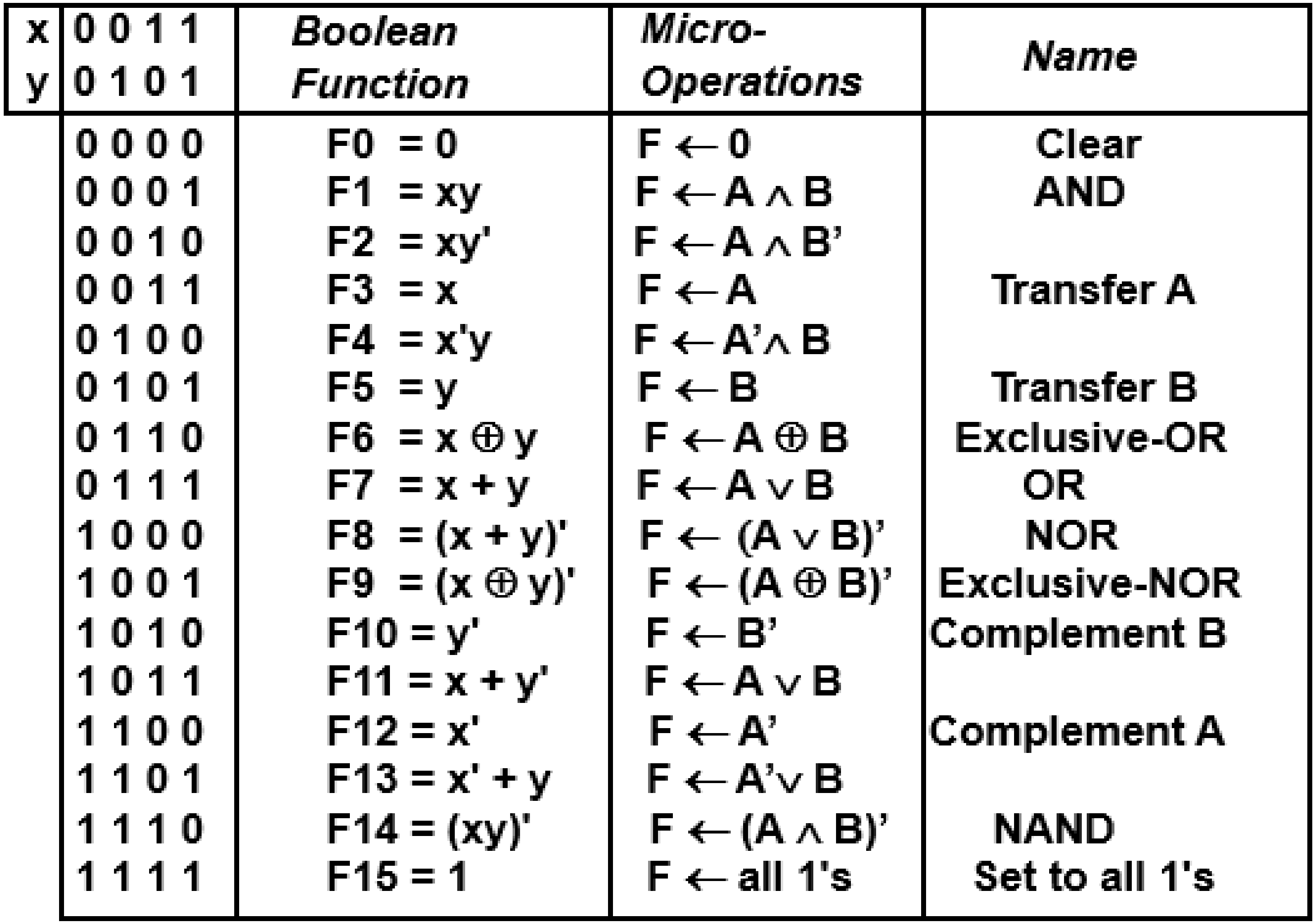
TABLE ·Truth Tables for 16 Functions of Two Variables



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

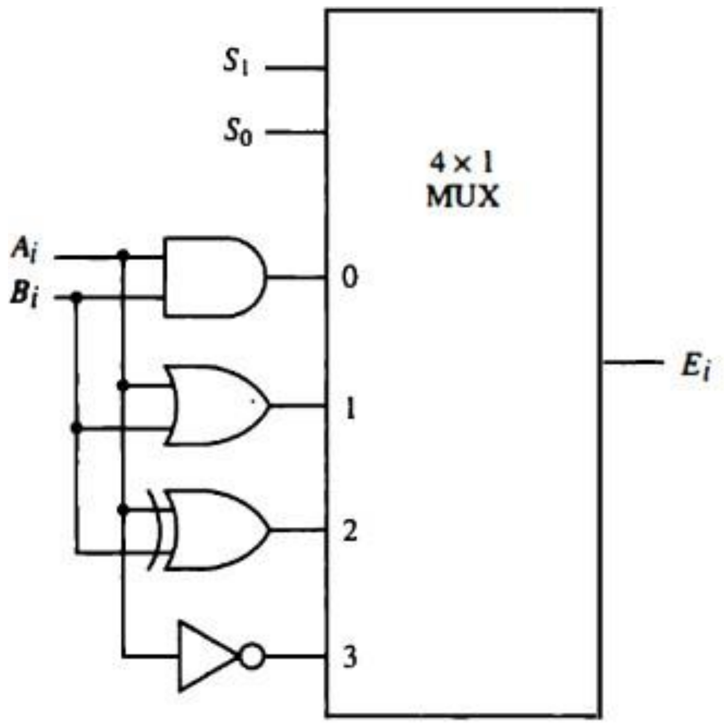
- In this table, each of the 16 columns F_0 through F_{15} represents a truth table of one possible Boolean function for the two variables x and y.
- The 16 Boolean functions of two variables x and y are expressed in algebraic form in the first column of Table below.
- The 16 logic microoperations are derived from these functions by replacing variable x by the binary content of register A and variable y by the binary content of register B.

Table : Sixteen Logic Microoperations



x y	0 0 1 1 0 1 0 1	*Boolean Function*	*Micro-Operations*	*Name*
	0 0 0 0	$F0 = 0$	$F \leftarrow 0$	Clear
	0 0 0 1	$F1 = xy$	$F \leftarrow A \wedge B$	AND
	0 0 1 0	$F2 = xy'$	$F \leftarrow A \wedge B'$	
	0 0 1 1	$F3 = x$	$F \leftarrow A$	Transfer A
	0 1 0 0	$F4 = x'y$	$F \leftarrow A' \wedge B$	
	0 1 0 1	$F5 = y$	$F \leftarrow B$	Transfer B
	0 1 1 0	$F6 = x \oplus y$	$F \leftarrow A \oplus B$	Exclusive-OR
	0 1 1 1	$F7 = x + y$	$F \leftarrow A \vee B$	OR
	1 0 0 0	$F8 = (x + y)'$	$F \leftarrow (A \vee B)'$	NOR
	1 0 0 1	$F9 = (x \oplus y)'$	$F \leftarrow (A \oplus B)'$	Exclusive-NOR
	1 0 1 0	$F10 = y'$	$F \leftarrow B'$	Complement B
	1 0 1 1	$F11 = x + y'$	$F \leftarrow A \vee B$	
	1 1 0 0	$F12 = x'$	$F \leftarrow A'$	Complement A
	1 1 0 1	$F13 = x' + y$	$F \leftarrow A' \vee B$	
	1 1 1 0	$F14 = (xy)'$	$F \leftarrow (A \wedge B)'$	NAND
	1 1 1 1	$F15 = 1$	$F \leftarrow$ all 1's	Set to all 1's

Hardware Implementation



(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	$E = \bar{A}$	Complement

(b) Function table

Figure : One stage of logic circuit.

Applications of logic microoperations

- Logic microoperations can be used to manipulate individual bits or a portions of a word in a register
- Consider the data in a register A. In another register, B, is bit data that will be used to modify the contents of A

• Selective-set	$A \leftarrow A + B$
• Selective-complement	$A \leftarrow A \oplus B$
• Selective-clear	$A \leftarrow A \bullet B'$
• Mask (Delete)	$A \leftarrow A \bullet B$
• Clear	$A \leftarrow A \oplus B$
• Insert	$A \leftarrow (A \bullet B) + C$
• Compare	$A \leftarrow A \oplus B$
• . . .	

Selective-set

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

1010	A before
$\underline{1100}$	B (logic operand)
1110	A after

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

Selective complement

- The selective-complement operation complements bits in A where there are corresponding l's in B. It does not affect bit positions that have 0's in B. For example:

  1010 A before

  $\underline{1100}$ B(logic Operand)

  0110 A after

- Again the two leftmost bits of B are 1s, so the corresponding bits of A are complemented.
- One can deduce that the selective-complement operation is just an exclusive-OR microoperation.
- Therefore the exclusive-OR microoperation can be used to selectively complement bits of a register.

Selective-clear

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

```
1010   A before
1100 B (logic operand)
0010 A after
```

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

Mask Operation

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

```
1010   A before
1100   B(logic operand )
----
1000   A after Masking
```

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

Insert Operation

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

```
0110  1010   A before
0000  1111   B(logic Operand)
0000  1010   A after Masking
```

And then insert new value:

```
0000  1010   A before
1001  0000   B(insert)
1001  1010   A after insertion
```

Clear operation

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

1010 A

$\underline{1010}$ B

0000 $A \leftarrow A \oplus B$

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

Shift microoperations

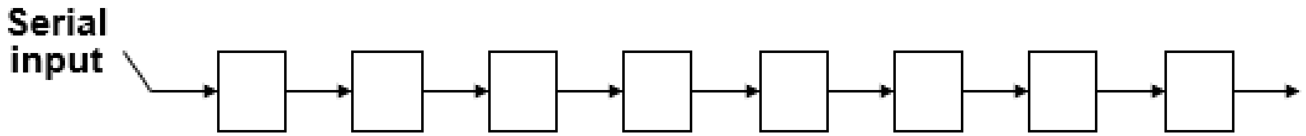
- Shift microoperations are used for serial transfer of data. They are also used in conjunction with arithmetic, logic and other data-processing operations.
- A logical shift is one that transfers 0 through the serial input. We will adopt the symbols and shr for logical shift-left and shift-right microoperations. For example:

R1←shl R1

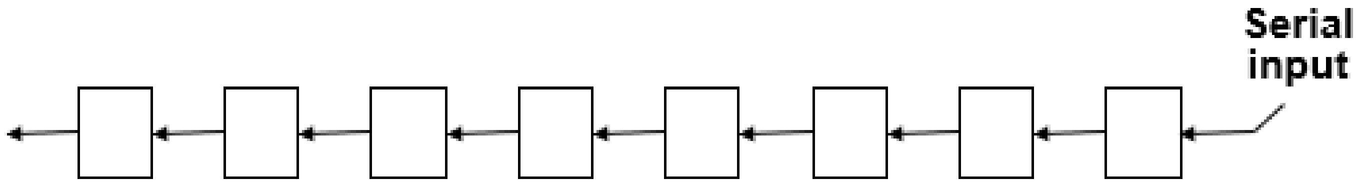
R2 ←shr R2

- are two microoperations that specify a 1-bit shift to the left of the content of register R1 and a 1-bit shift to the right of the content of register R2.
- The register symbol must be the same on both sides of the arrow. The bit transferred to the end position through the serial input is assumed to be 0 during a logical shift.

A right logical shift operation:



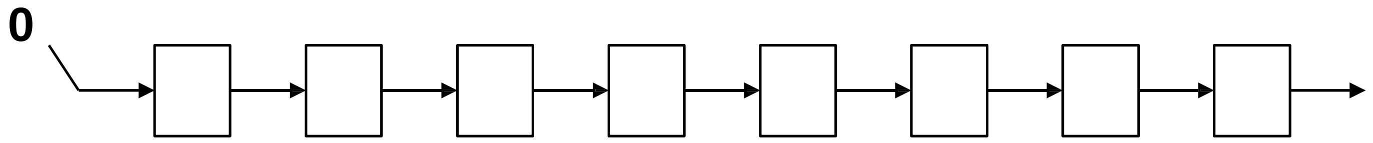
A left logical shift operation:



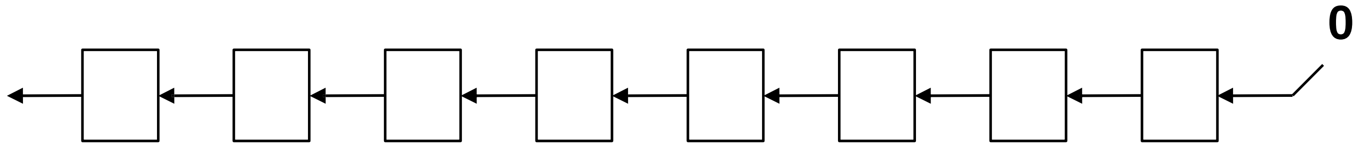
- There are three types of shifts
  1. Logical shift
  2. Circular shift
  3. Arithmetic shift
- What differentiates them is the information that goes into the serial input

Logical shift

- A logical shift is one that transfers 0 through the serial input.
- A right logical shift operation:



- A left logical shift operation:

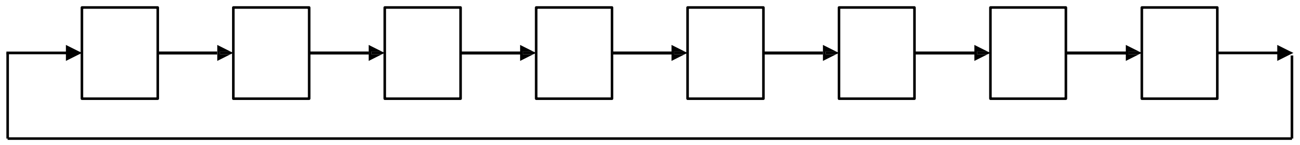


- In a Register Transfer Language, the following notation is used
  - *shl* for a logical shift left
  - *shr* for a logical shift right
  - Examples:
    - R2 ← *shr* R2
    - R3 ← *shl* R3

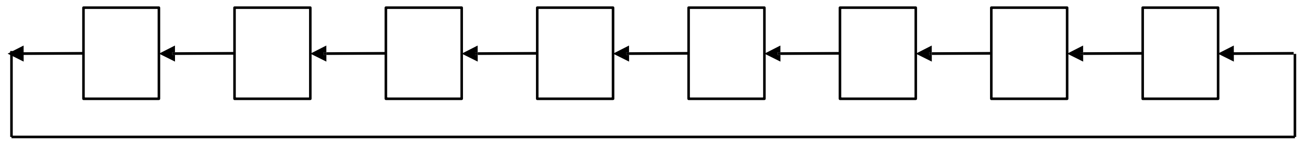
Circular shift

- In a circular shift the serial input is the bit that is shifted out of the other end of the register.

- A right circular shift operation:



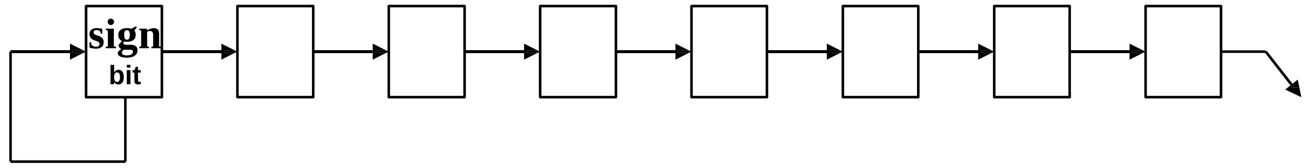
A left circular shift operation:



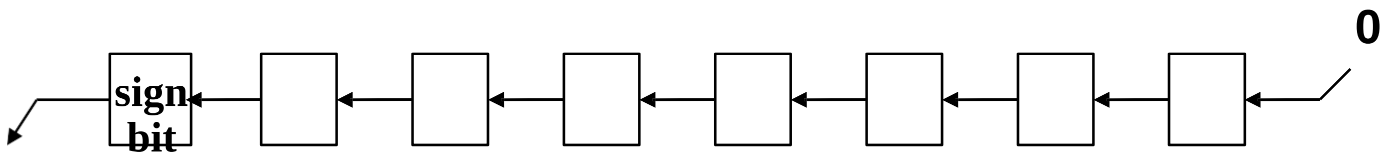
- In a RTL, the following notation is used
  - *cil* for a circular shift left
  - *cir* for a circular shift right
  - Examples: R2 ← *cir* R2 , R3 ← *cil* R3

Arithmetic shift

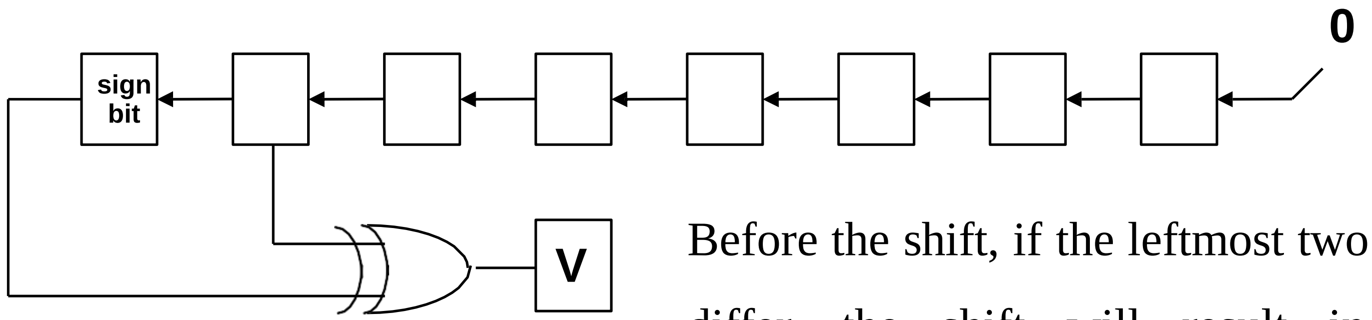
- An arithmetic shift is meant for signed binary numbers (integer)
- An arithmetic left shift multiplies a signed number by two
- An arithmetic right shift divides a signed number by two
- The main distinction of an arithmetic shift is that it must keep the sign of the number the same as it performs the multiplication or division
- A right arithmetic shift operation:



- A left arithmetic shift operation:



- An left arithmetic shift operation must be checked for the overflow, OR operation in 7th n 6th bit



Before the shift, if the leftmost two bits differ, the shift will result in an overflow

- An overflow flip-flop V can be used to detect an arithmetic shift-left overflow. $V=R_{n-1} \oplus R_{n-2}$, if V=0, there is no overflow but if V=1, overflow is detected.

Example: If the content of 8 bits register is (10100011). What is the result of the operation after executing to the register:

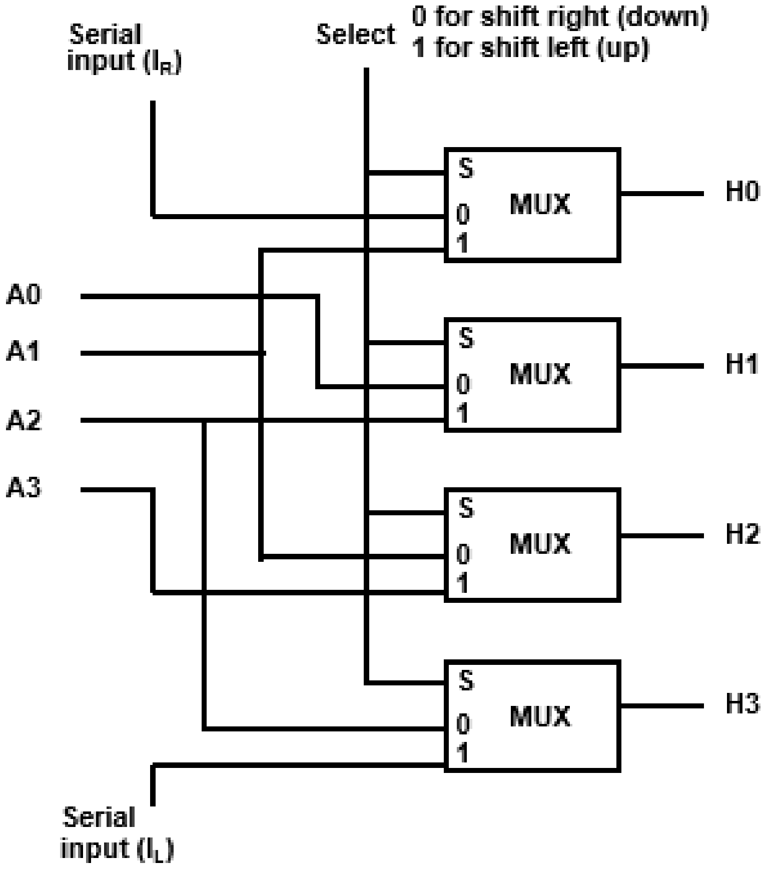
a. shl R: shift left register by 3. b. cil R : circular shift left register by 3.
c. ashl R: arithmetic shift left register by 3. d. ashr R: arithmetic shift right register by 3.

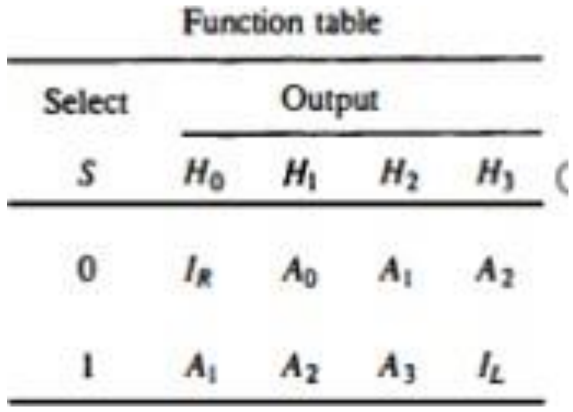
Ans:

(a) 00011000. (b) 00011101. (c) 00011000. Overflow (d) 11110100

- A combinational circuit shifter can be constructed with multiplexers as shown in figure below.
- The 4-bit shifter has four data inputs, A_0 through A_3, and four data outputs, H_0 through H3. There are two serial inputs, one for shift left (I_L) and the other for shift right (I_R).
- 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).

Hardware implementation of shift microoperations




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

Figure : 4 - bit combinational circuit shifter.

Arithmetic logic shift unit

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

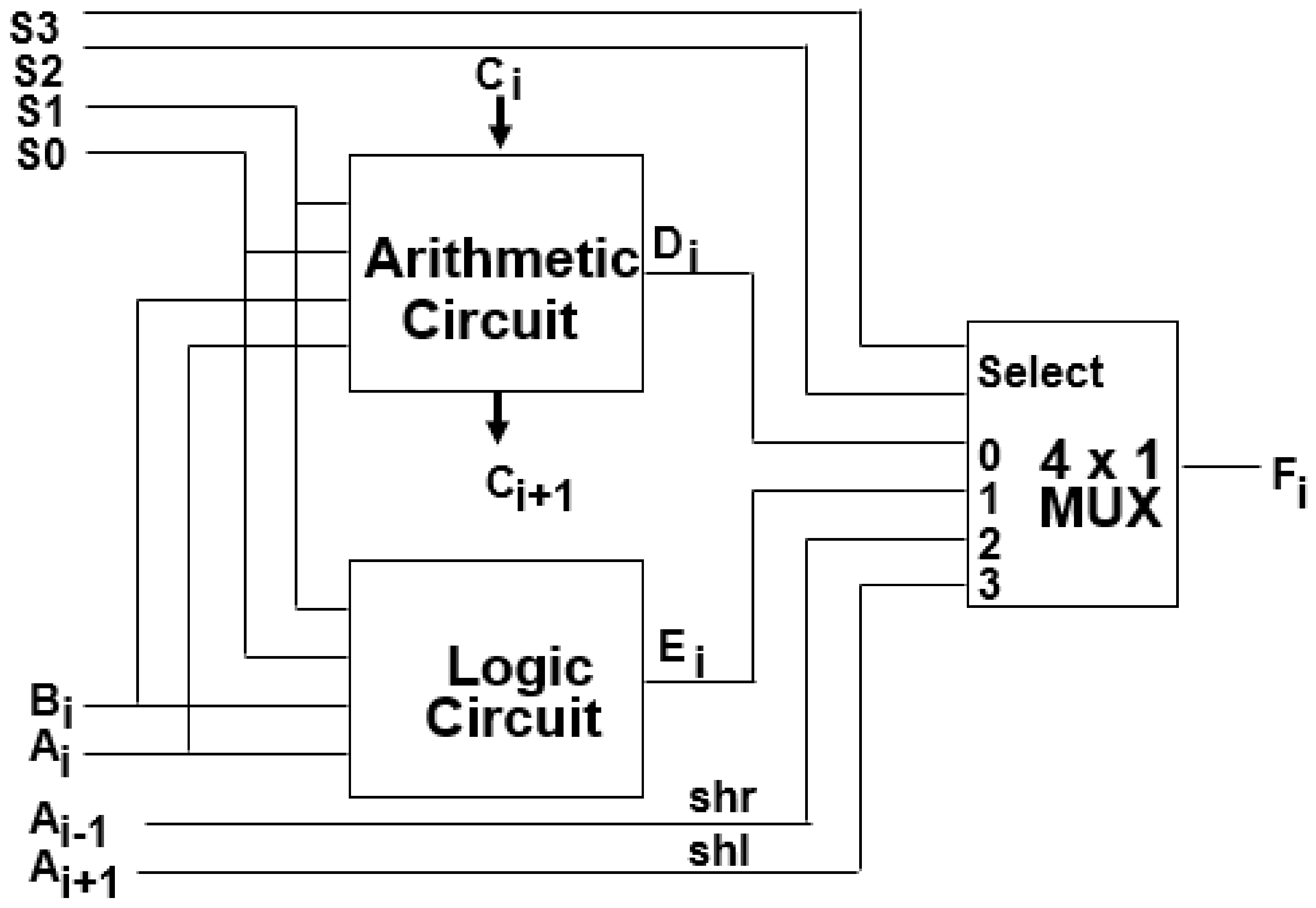


Figure : One stage of arithmetic logic shift unit.

- One stage of an arithmetic logic shift unit is shown in Fig. The subscript I designates a typical stage.
- Inputs Ai and Bi are applied to both the arithmetic and logic units. A particular microoperation is selected with inputs S1 and S0 .
- A 4 x 1 multiplexer at the output chooses between an arithmetic output in Ei and a logic output in Hi.
- The data in the multiplexer are selected with inputs S3 and S2.
- The other two data inputs to the multiplexer receive inputs Ai-1 for the shift-right operation and Ai+1for the shift-left operation.
- The diagram shows just one typical stage. The circuit of Fig. must be repeated n times for and 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.
- In every stage the circuit specified in Figure provides 8 arithmetic operation four logic operations and two shift operations. Each operation is selected with the five variables S3, S2, S1, S0 and C_{in}. The input carry C_{in} is used for selecting an arithmetic operation only.

Table : 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