Chapter Three:
Register Transfer and Microoperations

Contents to be covered

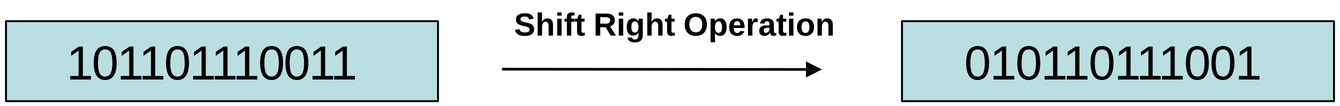
- **Register Transfer Language**
- **Register Transfer**
- **Bus and Memory Transfers**
- **Arithmetic Microoperations**
- **Logic Microoperations**
- **Shift Microoperations**

3-1 Register Transfer Language (RTL)

- Simple **digital Systems** are an interconnection of hardware modules that do a certain task on the information.
- The **modules** are constructed from such digital components as registers, decoders, arithmetic elements and control logic
- **Modules** are interconnected with common data and control paths to form a digital computer system
- **Register** is a very fast computer memory or we call it type of memory, used to store data/instruction in-execution.

3-1 Register Transfer Language cont.

- ***Microoperations***: operations executed on data stored in one or more registers.
- The result of the operation may be:
  - replace the previous binary information of a register or
  - transferred to another register



– Computer system micro-operations are of four types:
  - Register transfer micro-operations
  - Arithmetic micro-operations
  - Logic micro-operations
  - Shift micro-operations

3-1 Register Transfer Language cont.

- The internal hardware organization of a digital computer is 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 (RTL)** : a symbolic notation to describe the microoperation transfers among registers

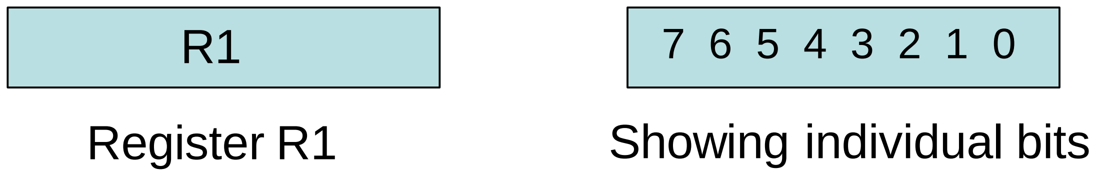
MOV AL,BL

3-2 Register Transfer (our first microoperation)

- Computer registers are designated by capital letters (sometimes followed by numerals) to denote the function of the register
  - R1: processor register holds an instruction , storage address or any data
  - MAR: Memory Address Register (holds an address for a memory unit)
  - PC: Program Counter:point to the next instruction to be executed.
  - IR: Instruction Register holds the instruction to be executed.
  - SR: Status Register also known as flag register.

3-2 Register Transfer cont.

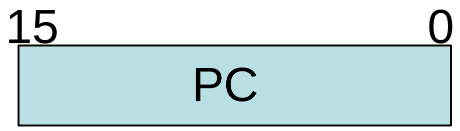
- The individual flip-flops in an n-bit register are numbered in sequence from 0 to n-1 (from the right position toward the left position)



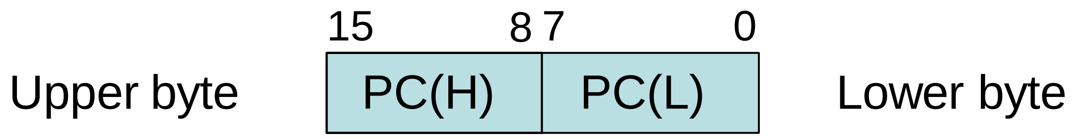
A block diagram of a register

3-2 Register Transfer cont.

Other ways of drawing the block diagram of a register:



Numbering of bits



Partitioned into two parts

3-2 Register Transfer cont.

- Information transfer from one register to another is described by a *replacement operator:* **R2 ← R1**
  - This statement denotes a transfer of the content of register R1 into register R2
  - The transfer happens in one clock cycle
  - The content of the R1 (source) does not change
  - The content of the R2 (destination) will be lost and replaced by the new data transferred from R1
- We are assuming that the 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

3-2 Register Transfer cont.

- Conditional transfer occurs only under a control condition

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

- Representation of a (conditional) transfer

  $P: \quad R2 \leftarrow R1$

- A binary condition (P equals to 0 or 1) determines when the transfer occurs
- The content of R1 is transferred into R2 only if P is 1

3-2 Register Transfer cont.

Basic Symbols for Register Transfers

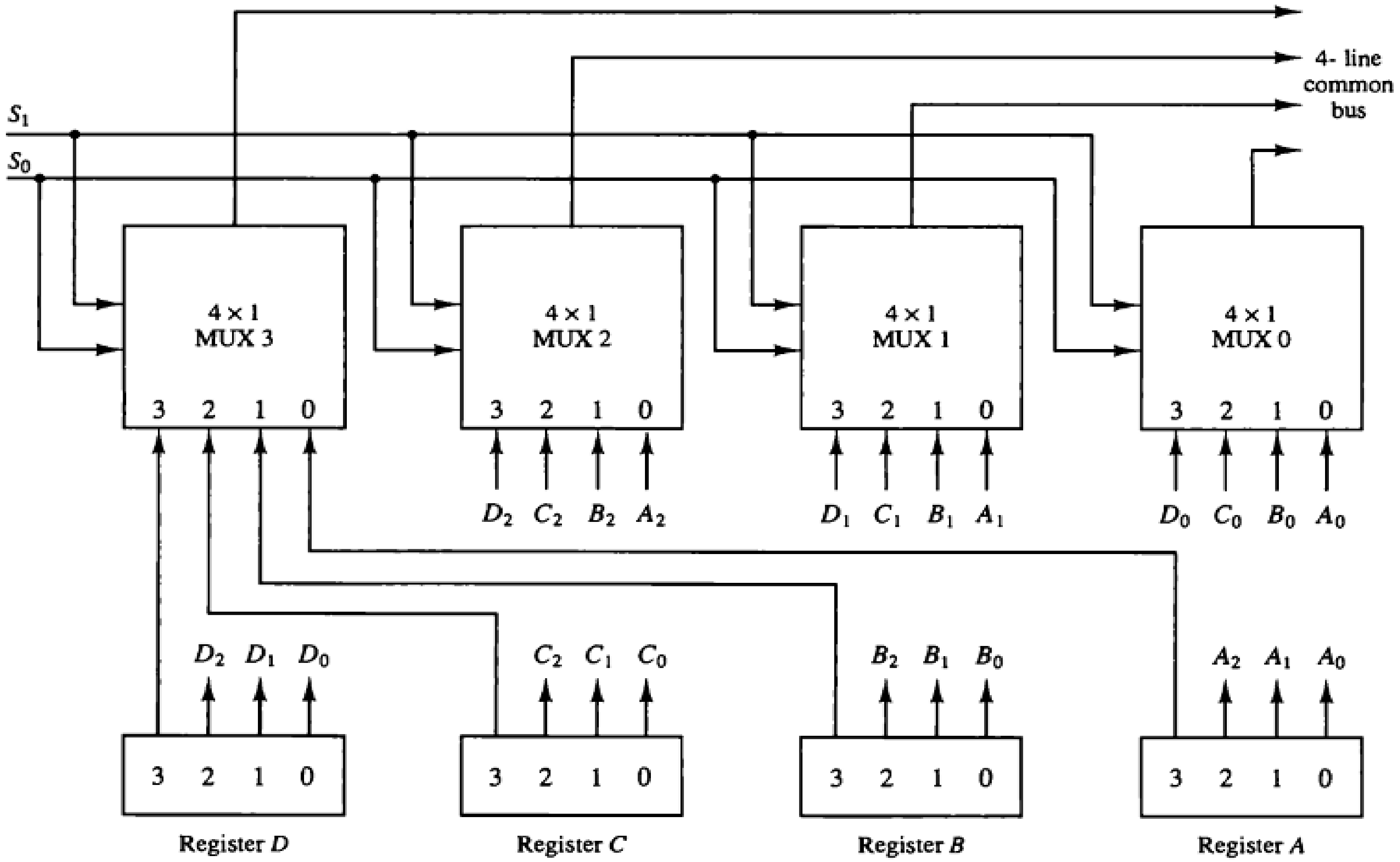
Symbol	Description	Examples
Letters & numerals	Denotes a register	MAR, R2
Parenthesis ( )	Denotes a part of a register	R2(0-7), R2(L),R2(8-bit)
Arrow $\leftarrow$	Denotes transfer of information	R2 $\leftarrow$ R1
Comma ,	Separates two microoperations	R2 $\leftarrow$ R1, R1 $\leftarrow$ R3

3-3 Bus and Memory Transfers

- Paths must be provided to transfer information from one register to another
- A ***Common Bus System*** is a scheme for transferring information between registers in a **multiple-register configuration**
- A **bus**: 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

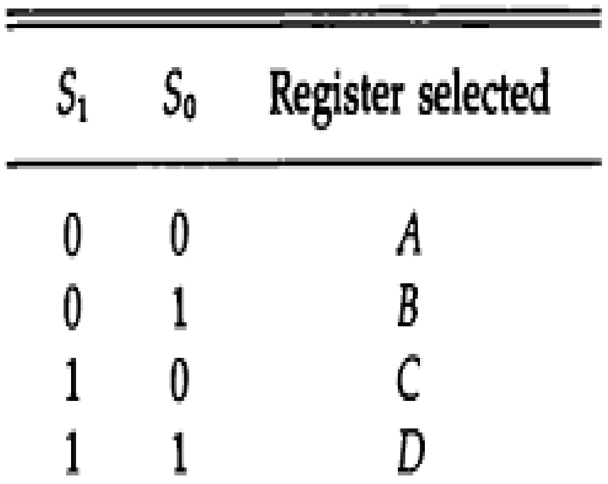
Implement the bus with the help of multiplexer.

-Number and size of mux depend on the number and size of register.



3-3 Bus and Memory Transfers

- The selection lines choose the four bits of one register and transfer them in to the four line common bus.



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

3-3 Bus and Memory Transfers

- The transfer of information from a bus into one of many destination registers is done:
  - By connecting the bus lines to the inputs of all destination registers and then:
  - activating the load control of the particular destination register selected
- We write: R2 ← C to symbolize that the content of register C is *loaded into* the register R2 using the common system bus
- It is equivalent to: BUS ←C, (select C)

  R2 ←BUS (Load R2)

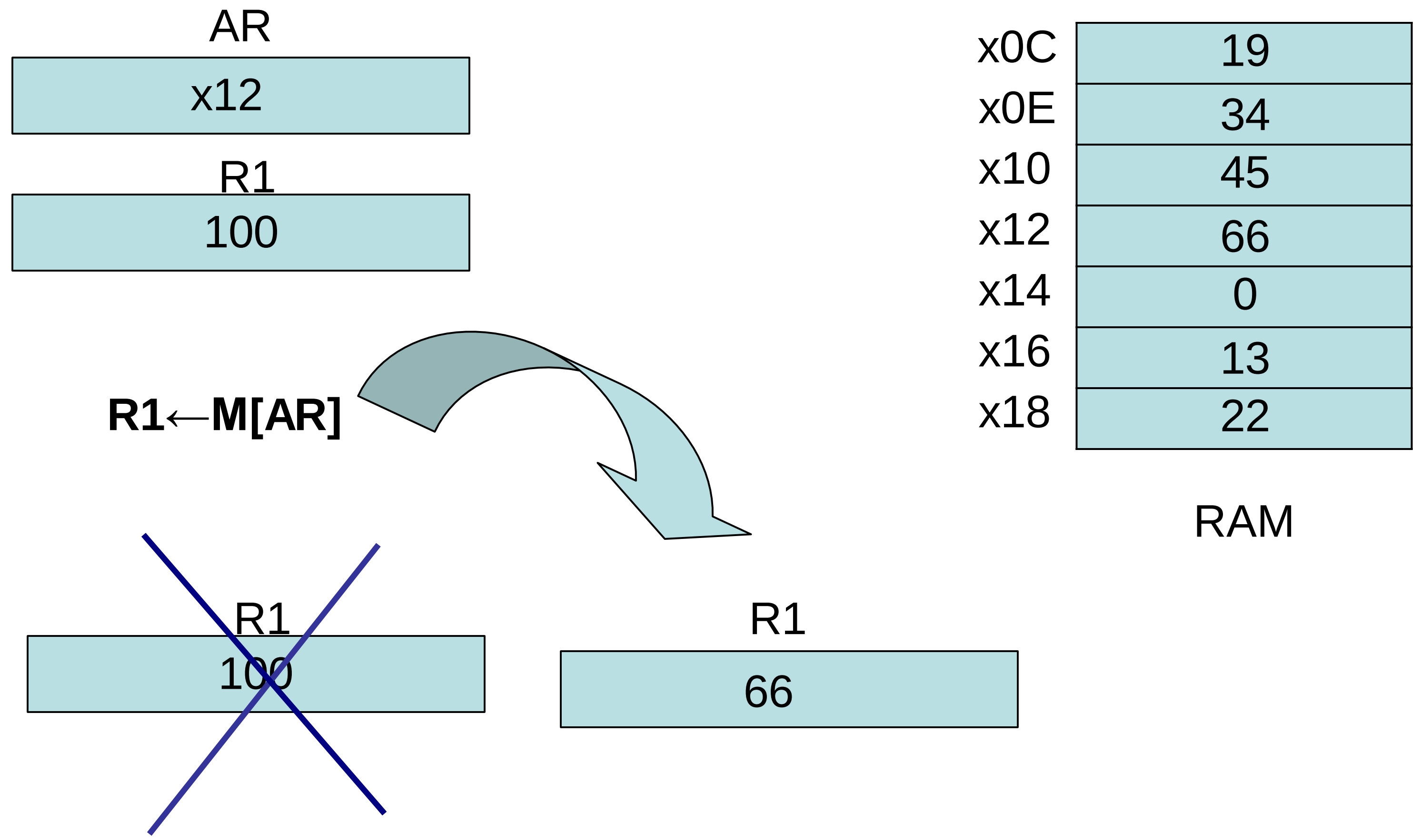
3-3 Bus and Memory Transfers:

- A memory unit is a collection of storage cells together with associated circuits needed to transfer information in and out of storage.
- The memory stores binary information in groups of bits called ***words***
  - Data being read or wrote is called a memory word (called M)
- It is necessary to specify the address of M when writing /reading memory
- This is done by enclosing the address in square brackets following the letter M
  - Example: M[0016] : the memory contents at address 0016

3-3 Bus and Memory Transfers:

- The transfer of information from a memory word to the outside environment is called a **read operation**.
  - **Memory read** : Transfer from memory
- The transfer of new information to be stored into the memory is called a **write operation**.
  - **Memory write** : Transfer to memory
- Assume that the address of a memory unit is stored in a register called the Address Register AR
- Lets represent a Data Register with DR, then:
  - Read: $DR \leftarrow M[AR]$
  - Write: $M[AR] \leftarrow DR$

3-3 Bus and Memory Transfers:

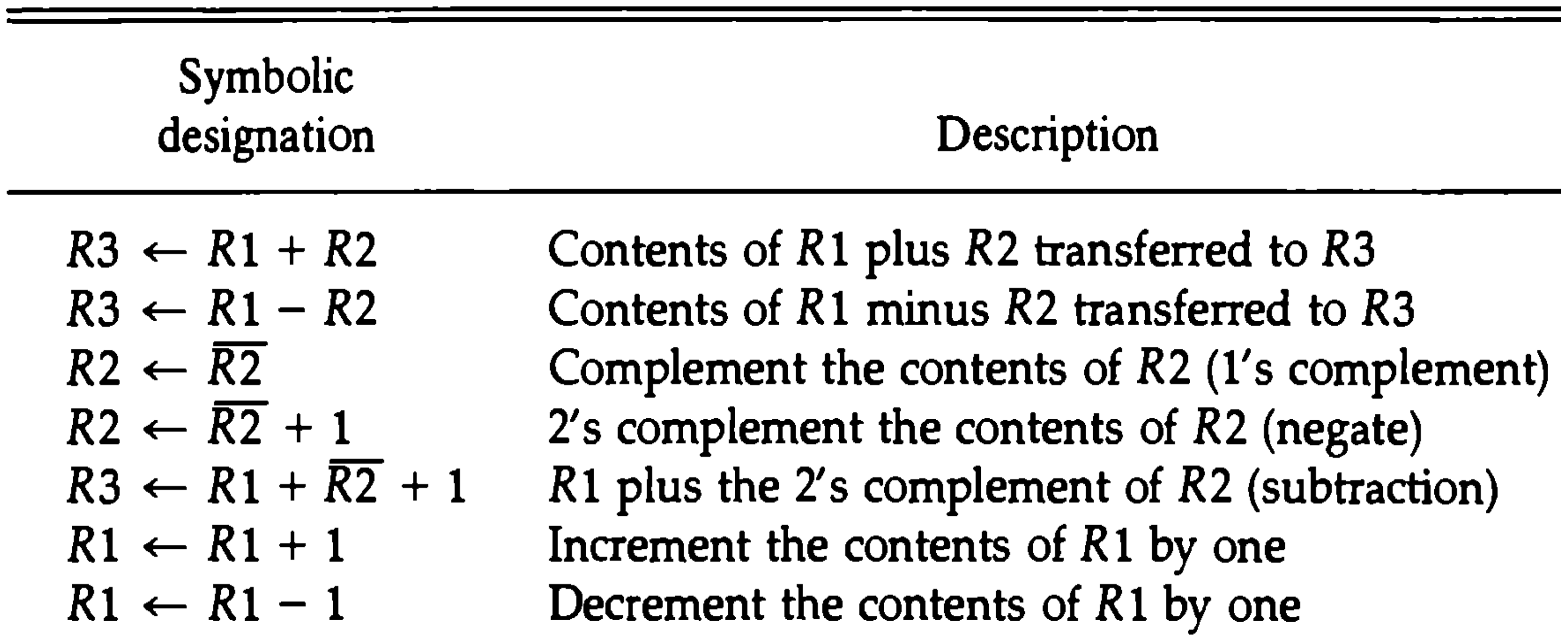


3-4 Arithmetic Microoperations

- The microoperations most often encountered in digital computers are classified into four categories:
  - Register transfer microoperations
  - Arithmetic microoperations (on numeric data stored in the registers)
  - Logic microoperations (bit manipulations on non-numeric data)
  - Shift microoperations

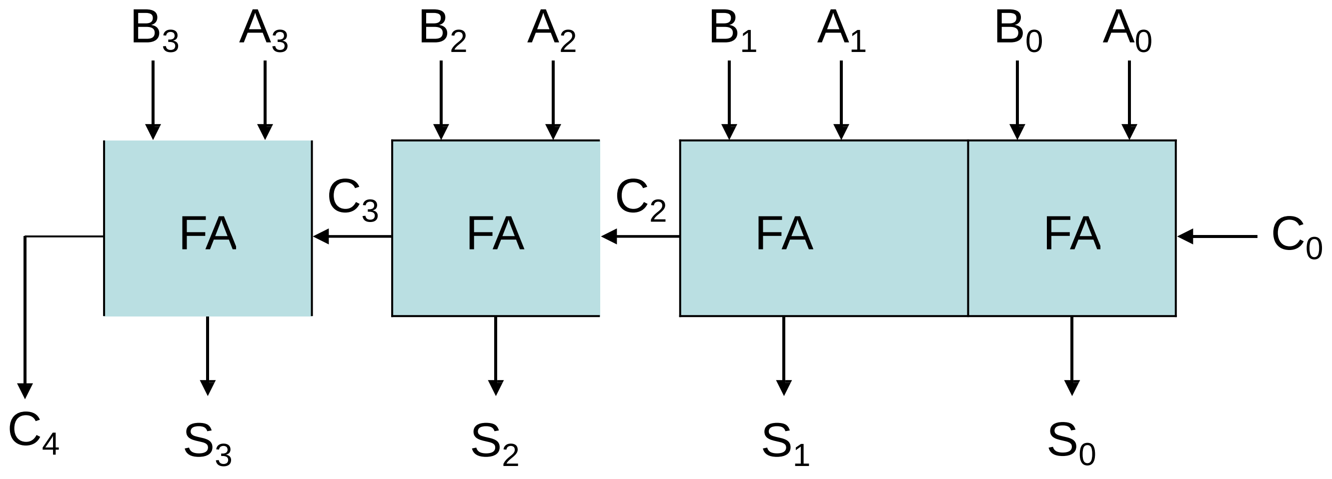
3-4 Arithmetic Microoperations cont.

- The basic arithmetic microoperations are: addition, subtraction, increment, decrement, and shift
- The following table shows the arithmetic microoperation



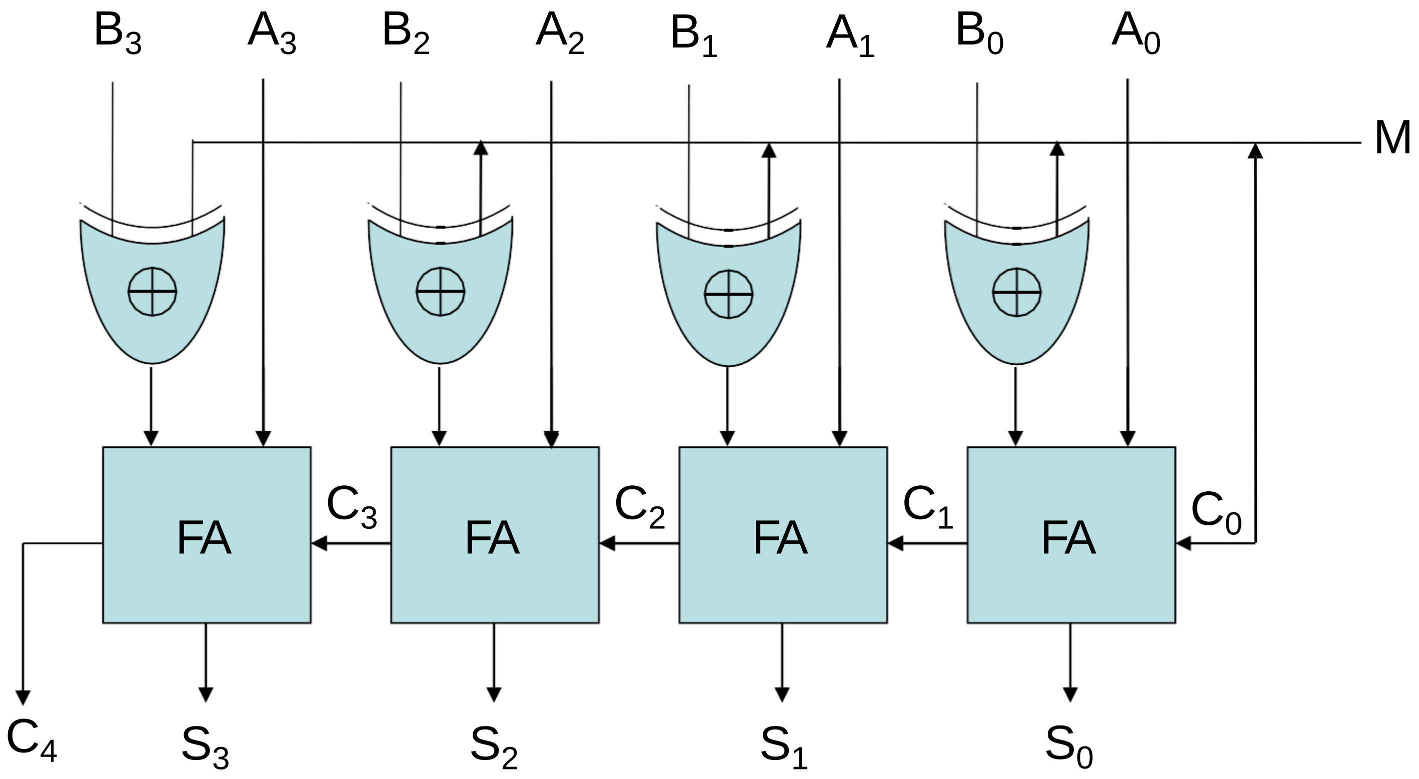
Symbolic designation	Description
$R3 \leftarrow R1 + R2$	Contents of $R1$ plus $R2$ transferred to $R3$
$R3 \leftarrow R1 - R2$	Contents of $R1$ minus $R2$ transferred to $R3$
$R2 \leftarrow \overline{R2}$	Complement the contents of $R2$ (1's complement)
$R2 \leftarrow \overline{R2} + 1$	2's complement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

3-4 Arithmetic Microoperations Binary Adder



4-bit binary adder
(connection of FAs)

3-4 Arithmetic Microoperations Binary Adder-Subtractor



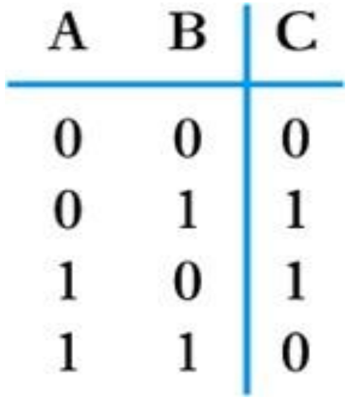
4-bit adder-subtractor

3-4 Arithmetic Microoperations Binary Adder-Subtractor

- The mode input M controls the operation
- When M=0, the circuit is an adder and

  B $\oplus$ 0=B and the input carry is 0, then A+B
- When M=1, the circuit becomes subtractor

  B $\oplus$ 1=B’ and the input carry is 1, then A plus 2’complement of B

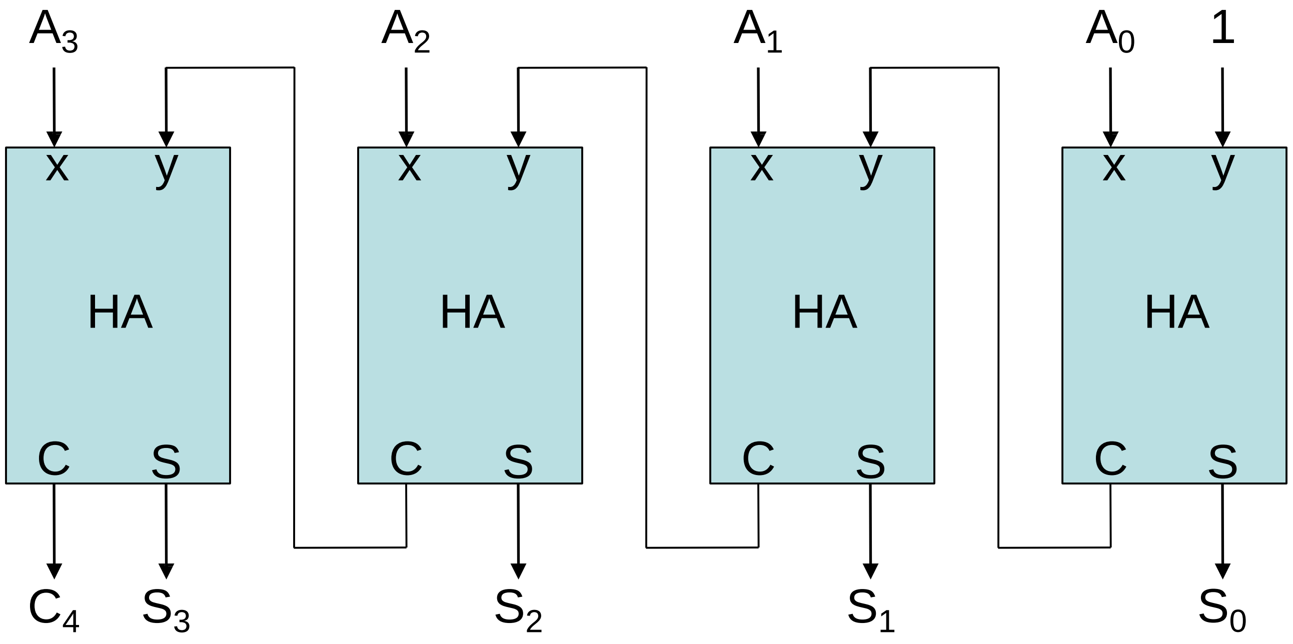


A	B	C
0	0	0
0	1	1
1	0	1
1	1	0

The subtraction A-B can be carried out by the following steps

- Take the 1 ’ complement of B
- Get the 2 ’ complement by adding 1
- Add the result to A

3-4 Arithmetic Microoperations Binary Incrementer



$X=1011 = A_3A_2A_1A_0$

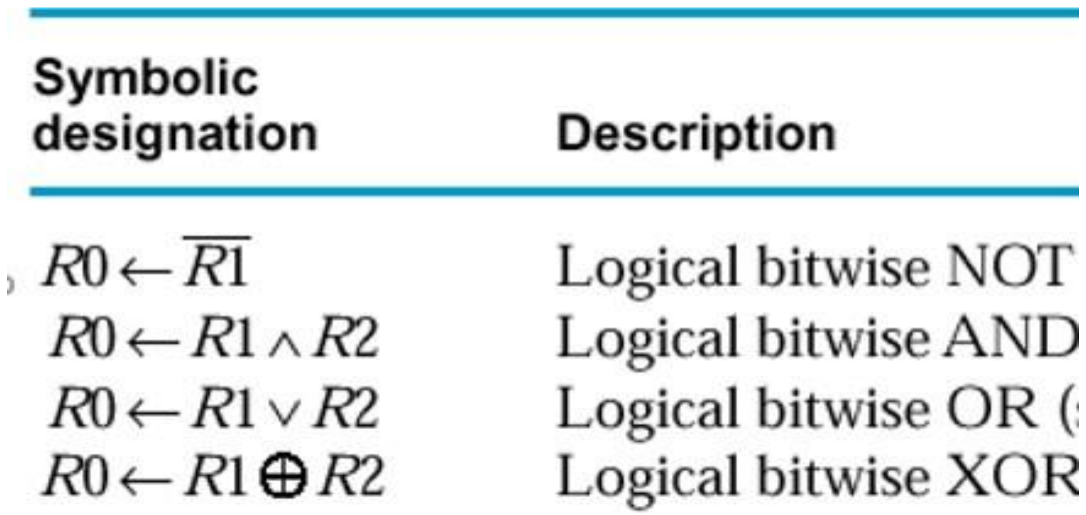
4-bit Binary Incrementer

3-4 Arithmetic Microoperations Binary Incrementer

- Binary Incrementer can also be implemented using a counter
- A binary decrementer can be implemented by adding 1111 to the desired register each time!

3-5 Logic Microoperations

- Manipulating the **bits** stored in a register (bit by bit)
- The four basic logic microoperations

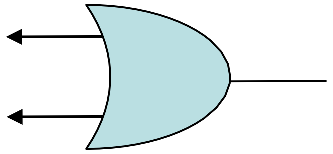


Symbolic designation	Description
$R0 \leftarrow \overline{R1}$	Logical bitwise NOT
$R0 \leftarrow R1 \wedge R2$	Logical bitwise AND
$R0 \leftarrow R1 \vee R2$	Logical bitwise OR (
$R0 \leftarrow R1 \oplus R2$	Logical bitwise XOR

3-5 Logic Microoperations The four basic microoperations

OR Microoperation

- Symbol: $\vee$, +
- Gate:



- Example: $100110_2 \vee 1010110_2 = 1110110_2$

OR

OR

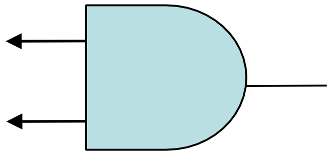
PVQ: $R1 \leftarrow R2+R3,\ R4 \leftarrow R5 \vee R6$

ADD

3-5 Logic Microoperations The four basic microoperations cont.

AND Microoperation

- Symbol: $\wedge$
- Gate:

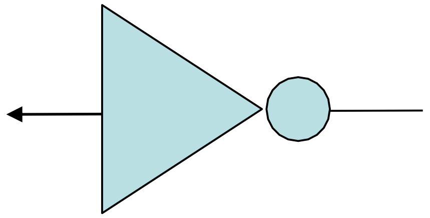


- Example: $100110_2 \wedge 1010110_2 = 0000110_2$

3-5 Logic Microoperations
The four basic microoperations $^{\text{cont.}}$

Complement (NOT) Microoperation

- Symbol: $\overline{\ \ }$



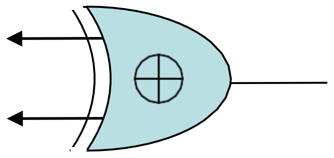
- Gate:

- Example: $\overline{1010110_2} = 0101001_2$

3-5 Logic Microoperations The four basic microoperations cont.

XOR (Exclusive-OR) Microoperation

- Symbol: ⊕
- Gate:



- Example: $100110_2 \oplus 1010110_2 = 1110000_2$

3-5 Logic Microoperations
Other Logic Microoperations

Selective-set Operation

- Used to force selected bits of a register into logic-1 by using the **OR** operation

•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.

Example: $0100_2 \vee 1000_2 = 1100_2$

In a processor register

Loaded into a register from memory to perform the selective-set operation

3-5 Logic Microoperations Other Logic Microoperations $^{\text{cont.}}$

Selective-complement (toggling) Operation

- Used to force selected bits of a register to be complemented by using the XOR operation

- Example: $0001_2 \oplus 1000_2 = 1001_2$

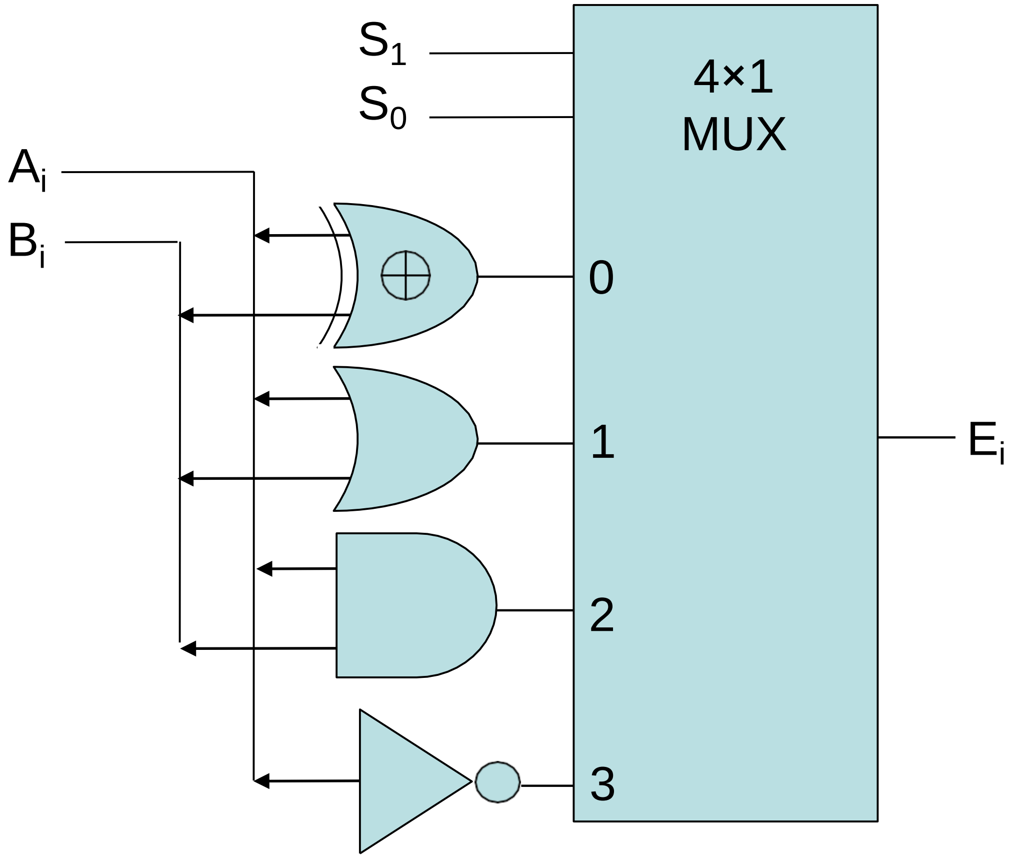
In a processor register

Loaded into a register from memory to perform the selective-complement operation

3-5 Logic Microoperations Hardware Implementation

- The hardware implementation of logic microoperations requires that logic gates be inserted for each bit or pair of bits in the registers to perform the required logic function
- Most computers use only four (AND, OR, XOR, and NOT) from which all others can be derived.

3-5 Logic Microoperations Hardware Implementation cont.

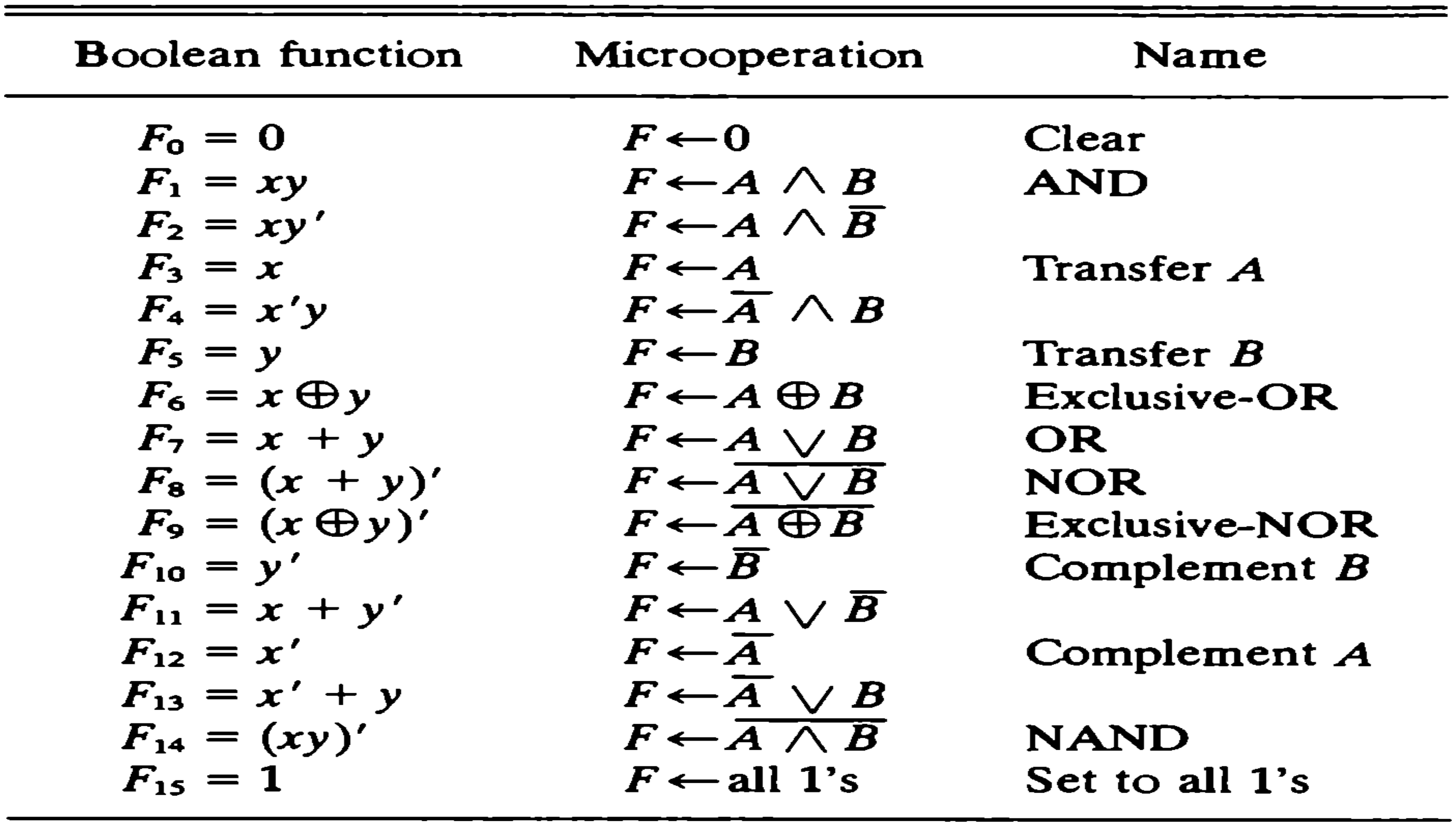


S_1	S_0	Output	Operation
0	0	$E = A \oplus B$	XOR
0	1	$E = A \vee B$	OR
1	0	$E = A \wedge B$	AND
1	1	$E = \overline{A}$	Complement

This is for one bit i

3-5 Logic Microoperations

- List of logic microoperations



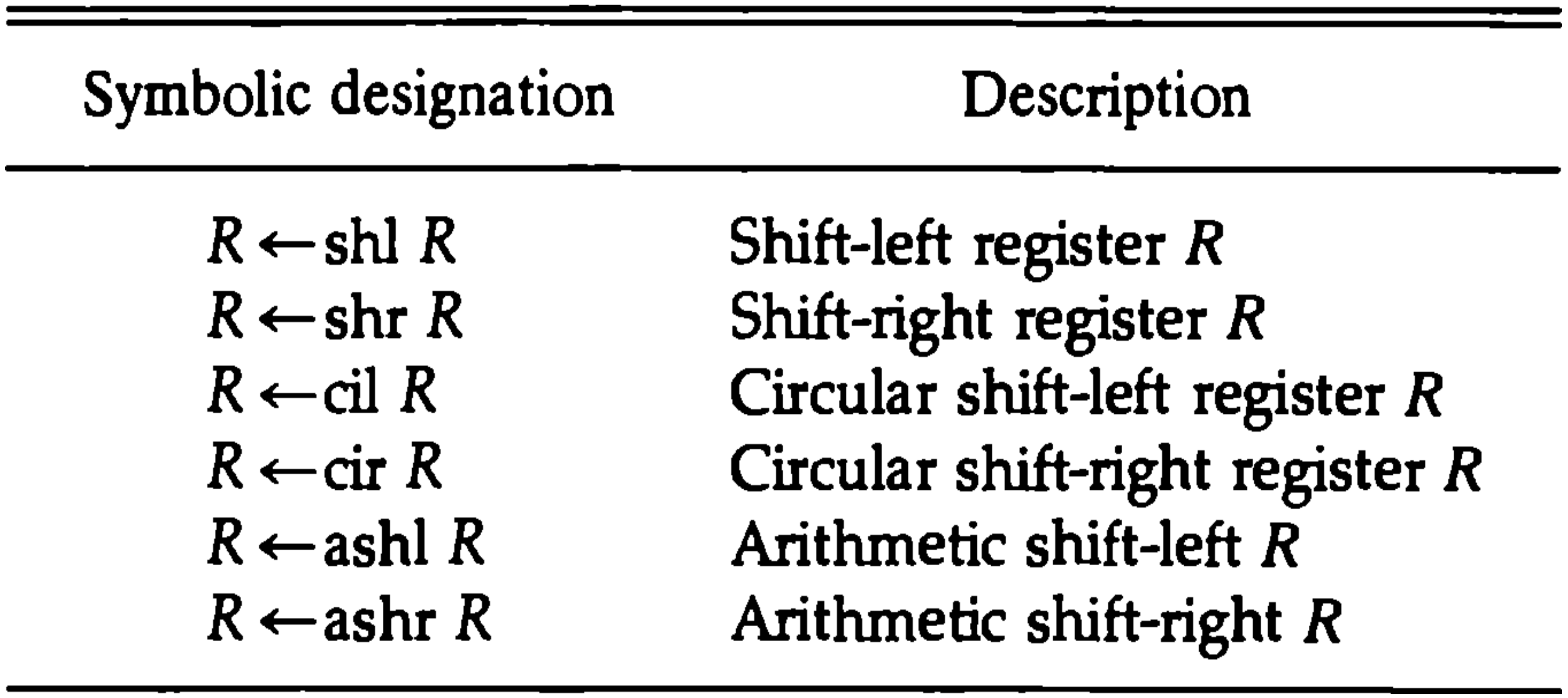
Boolean function	Microoperation	Name
$F_0 = 0$	$F \leftarrow 0$	Clear
$F_1 = xy$	$F \leftarrow A \wedge B$	AND
$F_2 = xy'$	$F \leftarrow A \wedge \overline{B}$	
$F_3 = x$	$F \leftarrow A$	Transfer A
$F_4 = x'y$	$F \leftarrow \overline{A} \wedge B$	
$F_5 = y$	$F \leftarrow B$	Transfer B
$F_6 = x \oplus y$	$F \leftarrow A \oplus B$	Exclusive-OR
$F_7 = x + y$	$F \leftarrow A \vee B$	OR
$F_8 = (x + y)'$	$F \leftarrow \overline{A \vee B}$	NOR
$F_9 = (x \oplus y)'$	$F \leftarrow \overline{A \oplus B}$	Exclusive-NOR
$F_{10} = y'$	$F \leftarrow \overline{B}$	Complement B
$F_{11} = x + y'$	$F \leftarrow A \vee \overline{B}$	
$F_{12} = x'$	$F \leftarrow \overline{A}$	Complement A
$F_{13} = x' + y$	$F \leftarrow \overline{A} \vee B$	
$F_{14} = (xy)'$	$F \leftarrow \overline{A \wedge B}$	NAND
$F_{15} = 1$	$F \leftarrow$ all 1's	Set to all 1's

3-6 Shift Microoperations

- Used for serial transfer of data
- Also used in conjunction with arithmetic, logic, and other data-processing operations
- The contents of the register can be shifted to the left or to the right
- As being shifted, the first flip-flop receives its binary information from the serial input
- There are three types of shifts:
  - **Logical shift,**
  - **Circular shift (rotate operation)**
  - **Arithmetic shift**

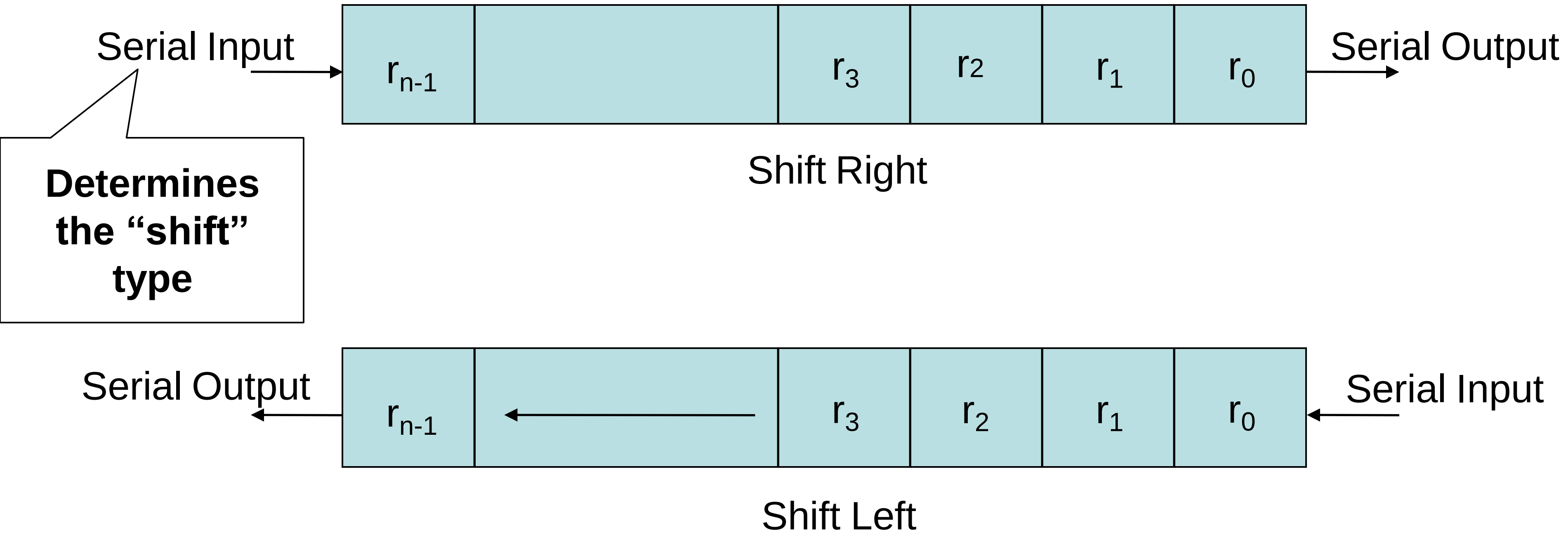
3-6 Shift Microoperations

- The following table shows shift microoperations



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

3-6 Shift Microoperations cont.



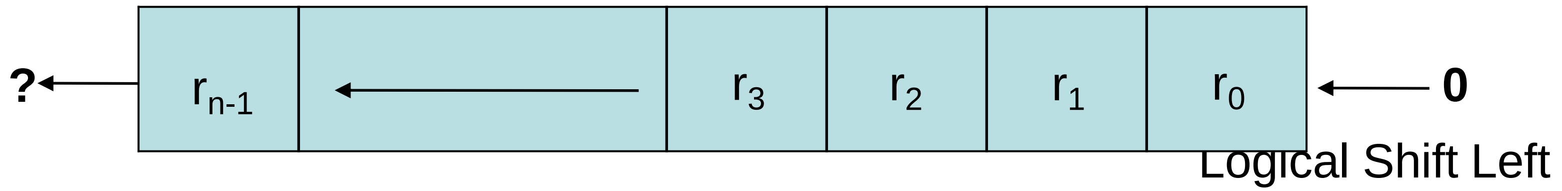
**Note that the bit ri is the bit at position (i) of the register

3-6 Shift Microoperations: Logical Shifts

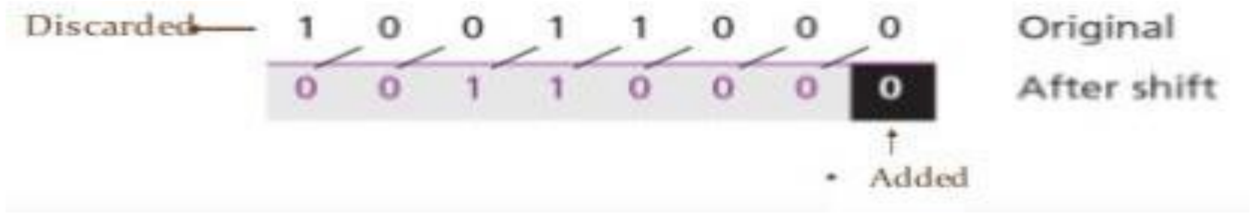
- A logical Shift transfers 0 through the serial input (***Zero inserted***)
- Logical Shift Right: R1←sh**r** R1
- Logical Shift Left: R2←sh**l** R2

The same

The same

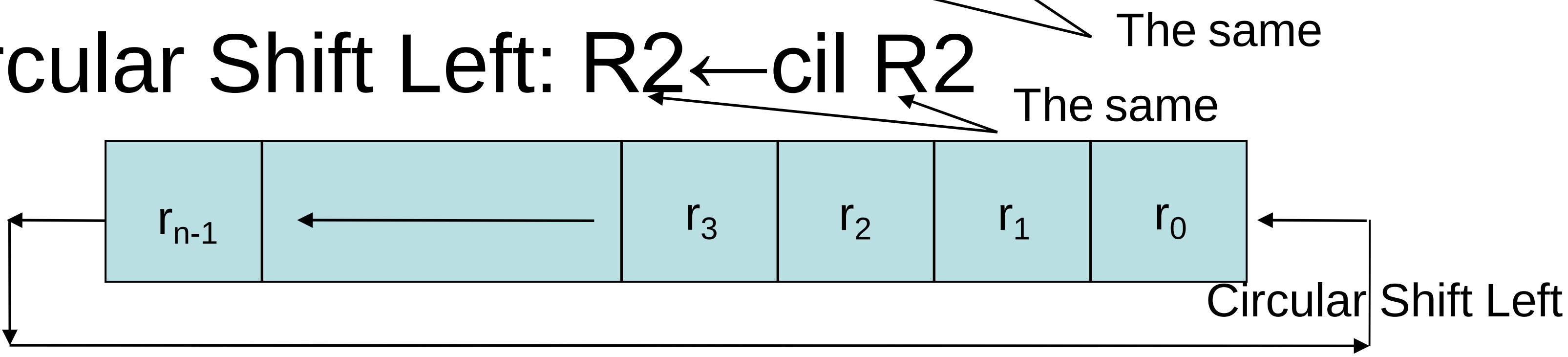


- Example: use a logical shift left on the bit pattern 10011000.
- Solution: The leftmost bit is lost and a **0 is inserted as the rightmost bit.**

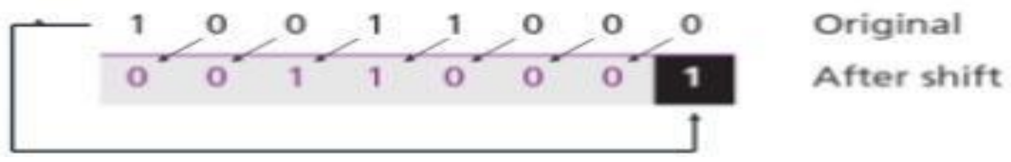


3-6 Shift Microoperations: Circular Shifts (Rotate Operation)

- Circulates the bits of the register around the two ends without loss of information
- Circular Shift Right: R1←cir R1
- Circular Shift Left: R2←cil R2



- Use a Circular Left Shift Operation on the bit pattern **10011000**.
- Solution: The leftmost bit is circulated and becomes the rightmost bit.

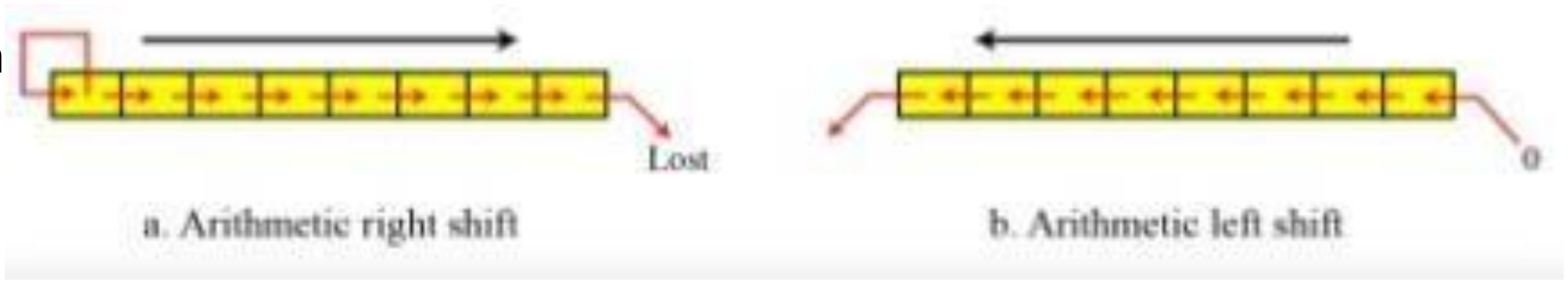


3-6 Shift Microoperations
Arithmetic Shifts

- Shifts a **signed binary number** to the left or right
- An arithmetic shift-left multiplies a signed binary number by 2: ashl (00100): 01000
- An arithmetic shift-right divides the number by 2

  ashr (00100) : 00010
- The sign bit is 0 for positive and 1 for negative.

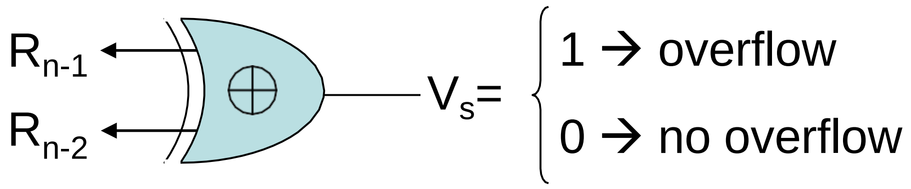
Sign Bit



3-6 Shift Microoperations Arithmetic Shifts cont.

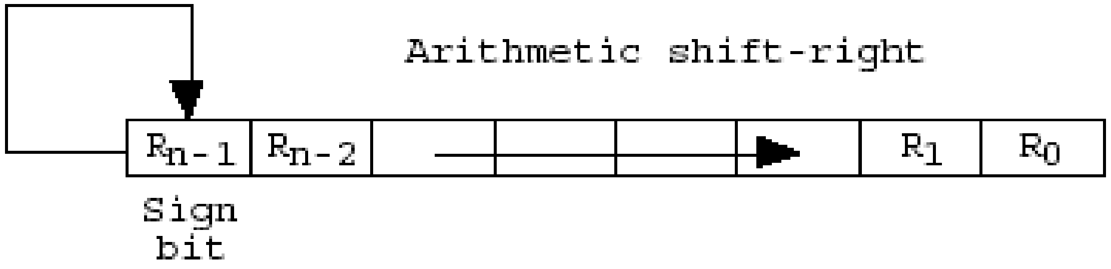
- An overflow may occur in arithmetic shift-left, and occurs when the **sign bit is changed** (sign reversal)
- An overflow flip-flop V_s can be used to detect an arithmetic shift-left overflow

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



3-6 Shift Microoperations Arithmetic Shifts

- **Arithmetic shift right:**
  - Rn-1 remains unchanged;
  - Rn-2 receives Rn-1, Rn-3 receives Rn-2, so on.
- For a negative number, 1 is shifted from the sign bit to the right.
- A negative number is represented by the **2's complement** form. The sign bit remained unchanged.



3-6 Shift Microoperations
Arithmetic Shifts

- **Arithmetic Shift Right :**
  - Example 1

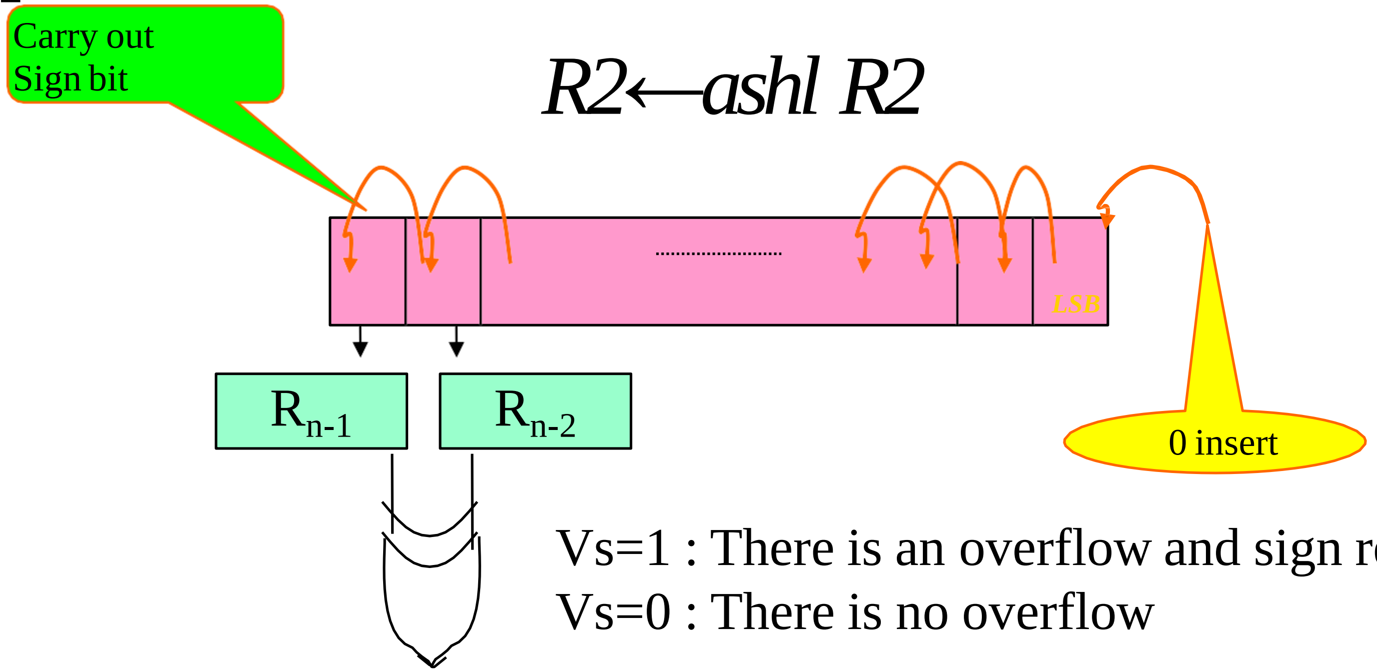
    0100 (4) → 0010 (2)
  - Example 2

    1010 (-6) → 1101 (-3)

3-6 Shift Microoperations Arithmetic Shifts

Arithmetic Shift Left

- The operation is same with **Logic shift-left**
  - transfers 0 through the serial input (***Zero inserted***)
- **The only difference is you need to check overflow problem**



Vs=1 : There is an overflow and sign reversal
Vs=0 : There is no overflow

3-6 Shift Microoperations Arithmetic Shifts

- Arithmetic Shift Left :
  - Example 1

    0010 (2) → 0100 (4)
  - Example 2

    1110 (-2) → 1100 (-4)
- Arithmetic Shift Left :
  - Example 3

    0100 (4) → 1000 (overflow)
  - Example 4

    1010 (-6) → 0100 (overflow)

3-6 Shift Microoperations cont.

- Example: Assume R1=11001110, then:
  - Arithmetic shift right once : R1 = 11100111
  - Arithmetic shift right twice : R1 = 11110011
  - Arithmetic shift left once : R1 = 10011100
  - Arithmetic shift left twice : R1 = 00111000
  - Logical shift right once : R1 = 01100111
  - Logical shift left once : R1 = 10011100
  - Circular shift right once : R1 = 01100111
  - Circular shift left once : R1 = 10011101