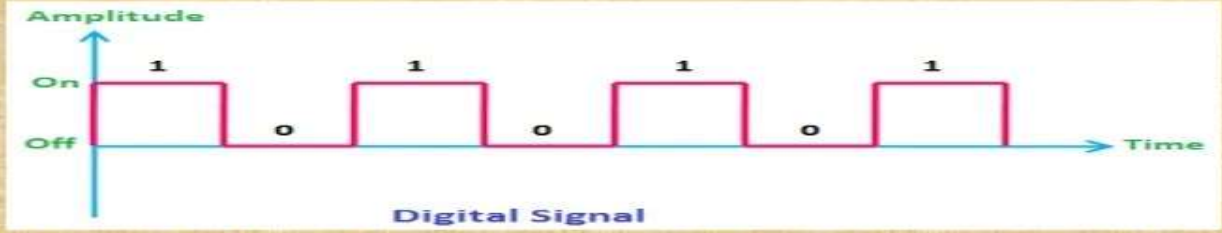
Register Transfer Language & Microoperations

Register Transfer Language (RTL)

- **Digital System**: An interconnection of hardware modules that do a certain task on the information.



- **Digital Module** = Registers + Operations performed on the data and stored

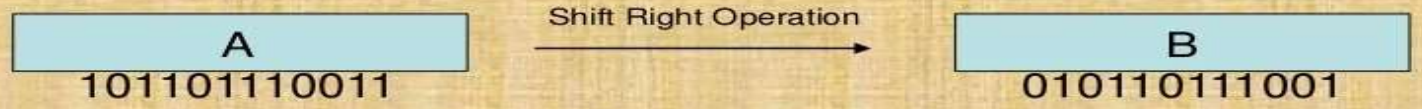
- Modules are interconnected with common data and control paths to form a digital computer system

Register

- Basically Register is collection of individual flip flops.
- Register is a temporary storage area built into a CPU.
- A special, high-speed storage area within the **CPU**. All data must be represented in a **register** before it can be processed.

Register Transfer Language

- Micro operations: operations executed on data stored in one or more registers.
- For any function of the computer, a sequence of microoperations is used to describe it
- The result of the operation may be:
  - replace the previous binary information of a register or
  - transferred to another register



RTL

example

- A=B+C
- A-B
- A=A-C
- A==1
- A++
- A--

Register Transfer Language

- The internal hardware organization of a digital computer is defined by specifying:
  - The set of registers it contains and their function
  - The sequence of micro operations performed on the binary information stored in the registers
- Registers + Microoperations Hardware + Control Functions = Digital Computer

Register Transfer Language

- Register Transfer Language (RTL) :
- A symbolic notation to describe the microoperation transfers among registers

Next steps:

- Define symbols for various types of micro operations
- Describe the hardware that implements these microoperations

Register Transfer

- 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

Register Transfer

- 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 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

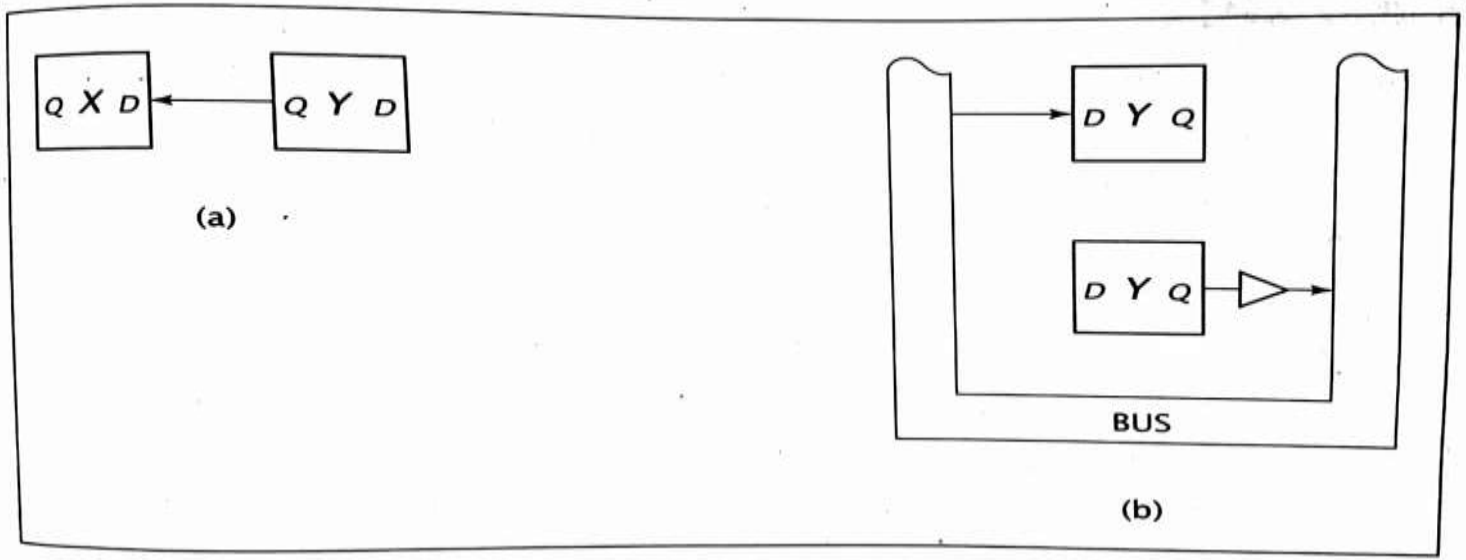
Register Transfer with Control Function

- Conditional transfer occurs only under a control condition

- Representation of a (conditional) transfer
  P: $R2 \leftarrow R1$
- 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

Consider a digital system with two 1-bit registers, X and Y. The µop that copies the contents of register Y to register X can be expressed as X←Y.

Implementations of the micro-operation $X \leftarrow Y$ using (a) a direct connection and (b) a bus connection



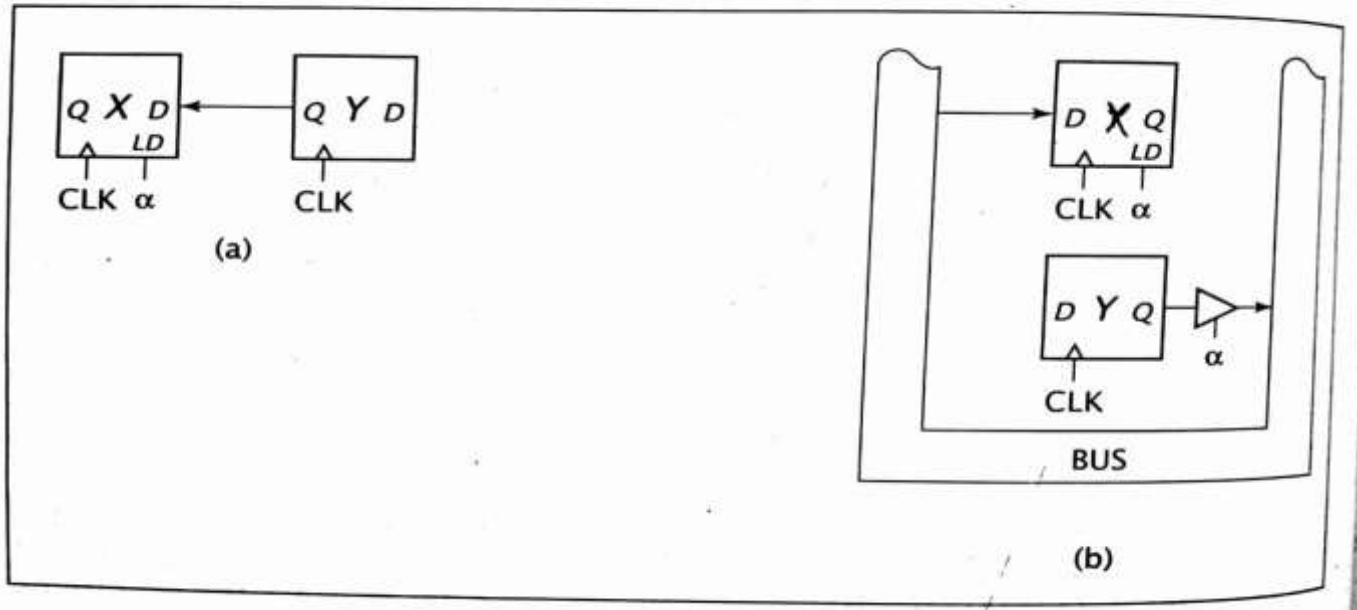
Contd…

- Both designs provide a path for data to flow from register Y to register X, but neither specifies when X should load this data.
- Assume that the transfer should occur when control input α is high.

  $\alpha: X \leftarrow Y$
- When all conditions to the left of the colon are asserted, the data transfers specified by the μops are performed.
- α is used to load register X and, in the bus-based implementation, to enable the tri-state buffer so that the contents of register Y are placed on the bus.

Contd…

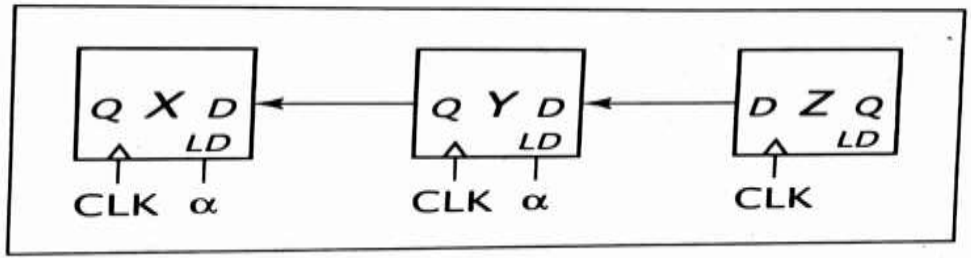
Implementations of the data transfer α: $X \leftarrow Y$ with control signals: (a) with direct path, and (b) using a bus



Contd…

- One way to improve system performance is to perform two or more µops simultaneously.
- The µops are separated by commas; the order in which they are written is unimportant because they are performed concurrently.
- Consider: α: X←Y, Y←Z or α: Y←Z, X←Y
  If X=0, Y=1 and Z=0 just before α becomes 1, these µops set X=1(the original value of Y) and Y=0.
- Note that a single bus cannot be used here because a bus can hold only one value at a time.
- When α=1, both Y and Z must travel on the data paths simultaneously.

Implementation of the data transfer α: X←Y, Y←Z

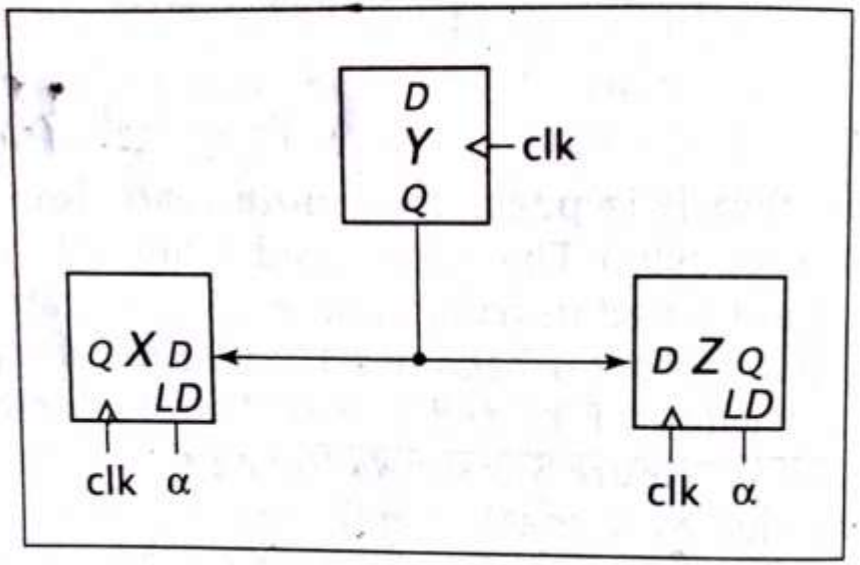


Consider the transfers that occur when α = 1.

α: X←Y, Z←Y

Register Y can be read by many other registers simultaneously; both micro operations can be performed concurrently.

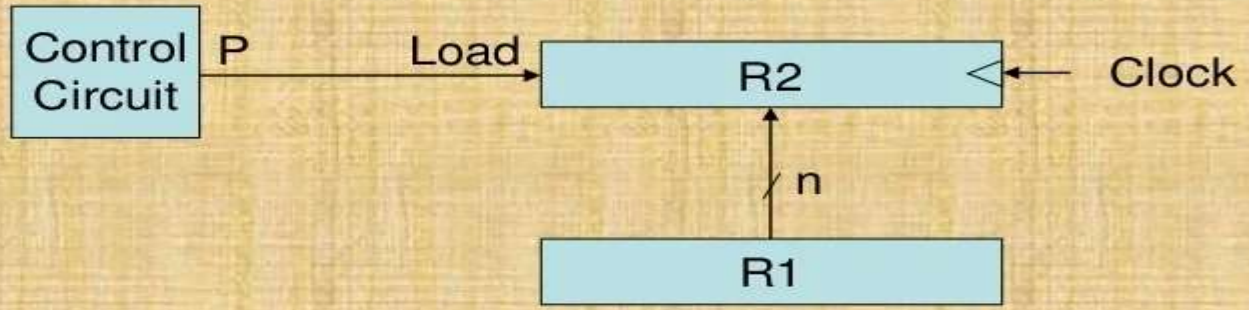
Implementation of the data transfer α: X←Y, Z←Y



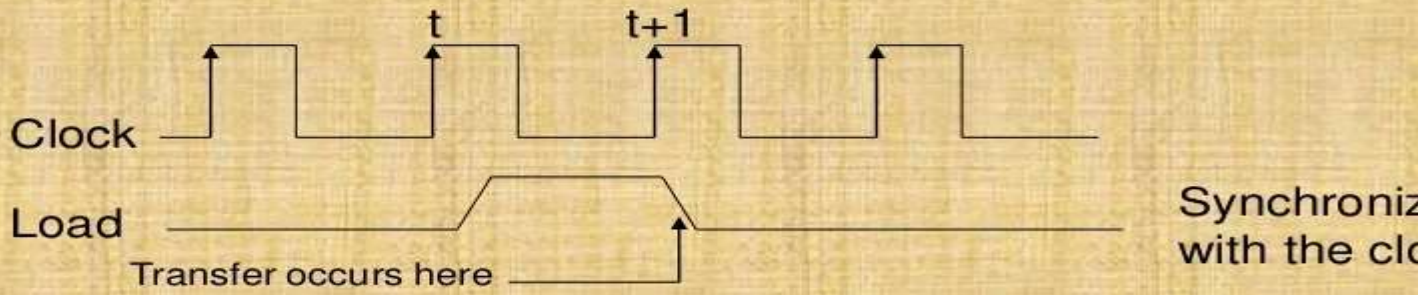
Register Transfer

Hardware implementation of a controlled transfer: P: R2 ← R

Block diagram:



Timing diagram



Register Transfer

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)
Arrow ←	Denotes transfer of information	R2 ← R1
Comma ,	Separates two microoperations	R2 ← R1, R1 ← R

Bus and Memory Transfers

- Paths must be provided to transfer information from one register to another
- A bus: set of common lines, one for each bit of a register, through which binary information transferred one at a time..

Memory Operations

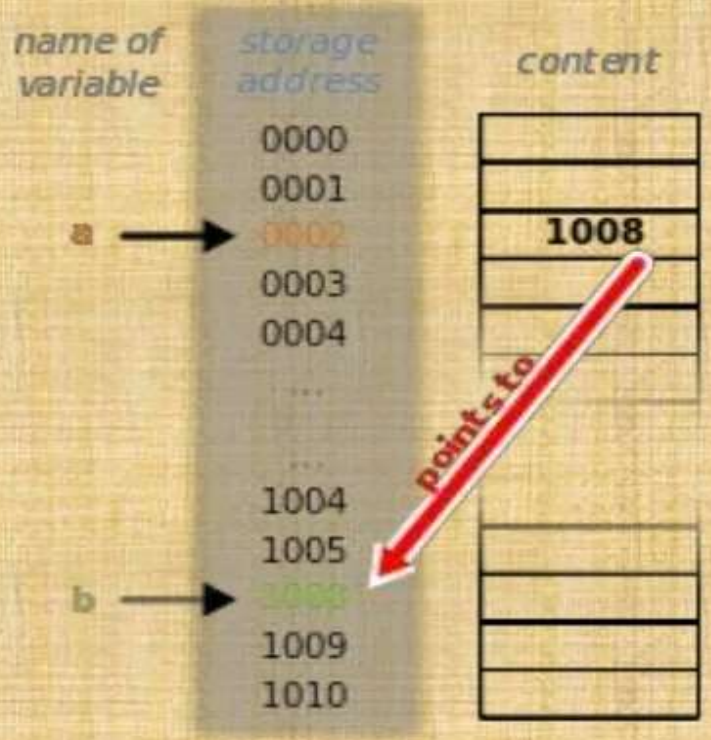
- **READ operation**
- Helps to retrieve data from memory.
- **WRITE operation**
- Storing data into a memory.

- Memory read : Transfer from memory
- Memory write : Transfer to memory

Bus and Memory Transfers: Memory Transfer

- Data being read or write 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 0x0016

Note: 0x written in front of a number is an alternative way to say “this is hexadecimal.



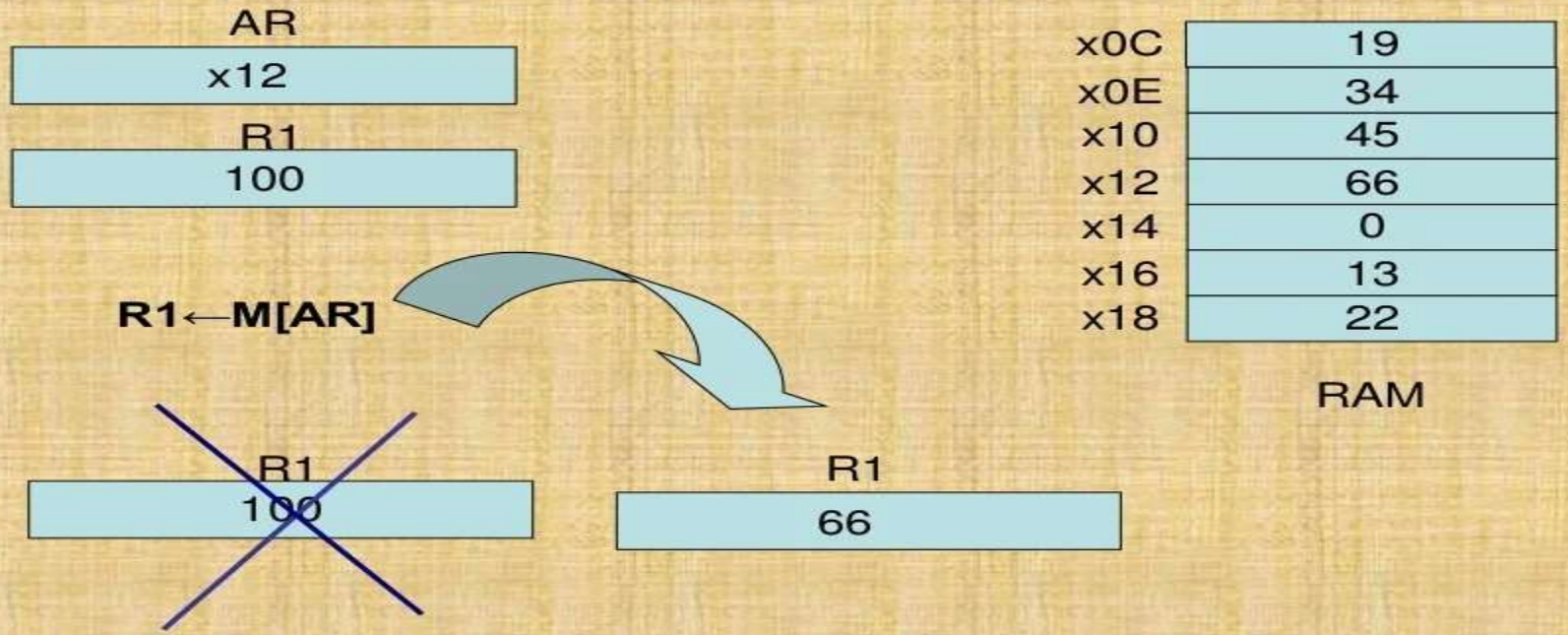
Address	Value
0x00	01001010
0x01	10111010
0x02	01011111
0x03	00100100
0x04	01000100
0x05	10100000
0x06	01110100
0x07	01101111
0x08	10111011
...	...
0xFE	11011110
0xFF	10111011

Memory related registers

1. Memory Address Register (MAR)
2. Memory Data Register (MDR)

- **MAR** is the CPU register that either stores the memory address from which data will be fetched to the CPU or the address to which data will be sent and stored.
- **MDR** is the register in a computer's processor that stores the data being transferred to and from the immediate access storage.

Bus and Memory Transfers: Memory Transfer



Arithmetic Microoperations

- The micro operations in digital computers are classified into four categories:
  - Register transfer microoperations
  - Arithmetic microoperations
  - Logic microoperations
  - Shift microoperations

Arithmetic Microoperations

- The basic arithmetic microoperations are: addition, subtraction, increment, decrement, and shift
- Addition Microoperation:

$$R3 \leftarrow R1+R2$$

- Subtraction Microoperation:

$$R3 \leftarrow R1-R2 \text{ or :}$$

1's complement

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

Arithmetic Microoperations
cont.

- One's Complement Microoperation:

$$R2 \leftarrow \overline{R2}$$

- Two's Complement Microoperation:

$$R2 \leftarrow R2+1$$

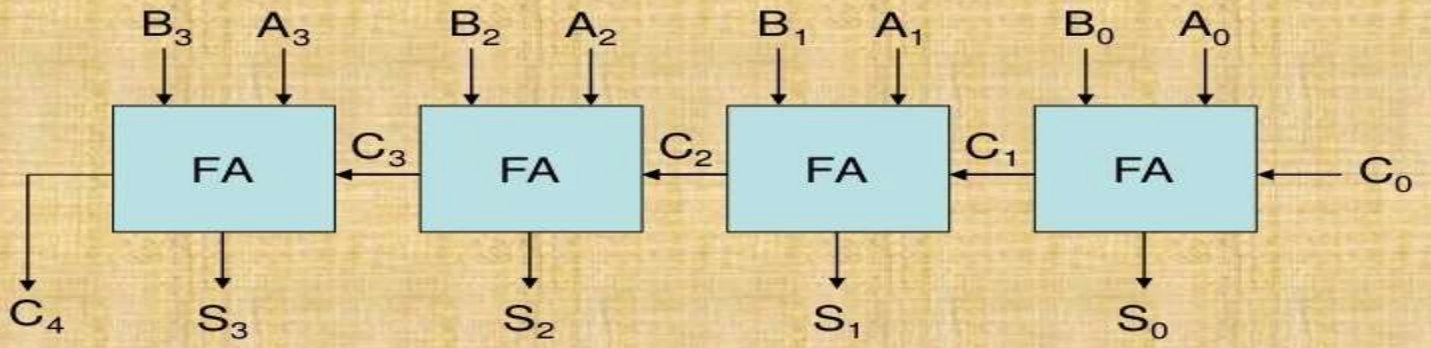
- Increment Microoperation:

$$R2 \leftarrow R2+1$$

- Decrement Microoperation:

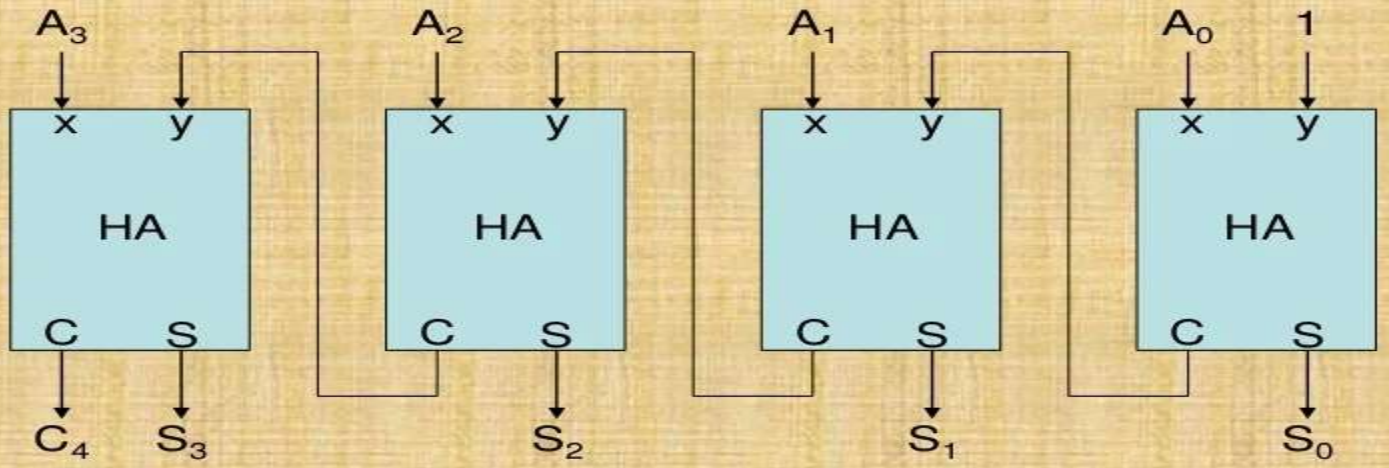
$$R2 \leftarrow R2-1$$

Arithmetic Microoperations
Binary Adder



4-bit binary adder
(connection of FAs)

Arithmetic Microoperations
Binary Incrementer

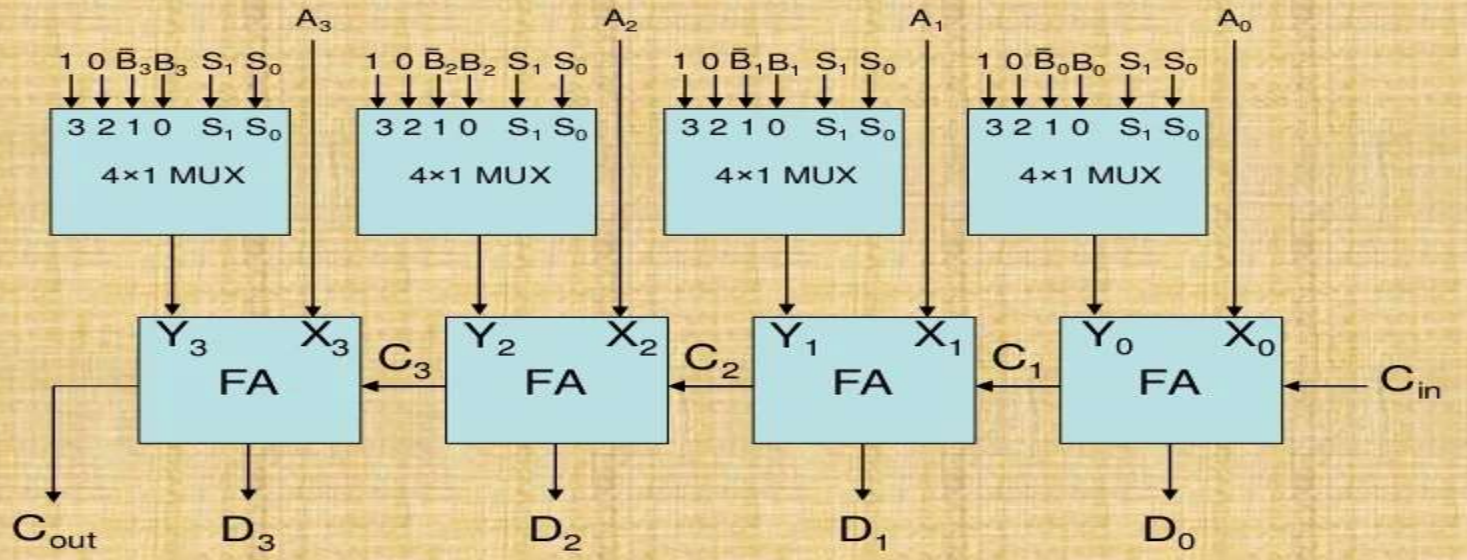


4-bit Binary Incrementer

Arithmetic Microoperations
Binary Incrementer

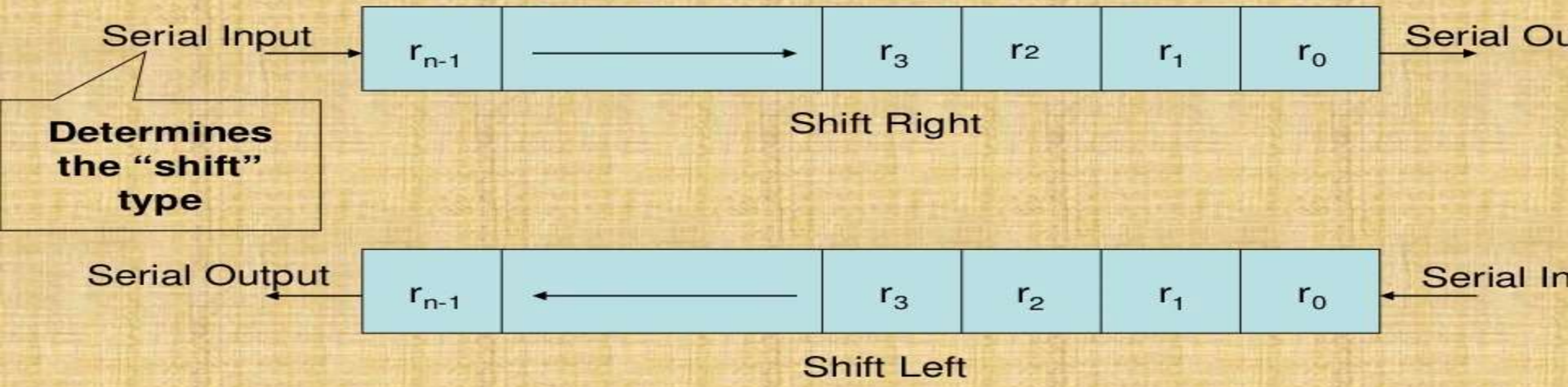
- Binary Incrementer can also be implemented using a counter
- A binary decrementer can be implemented by subtracting 1111 to the register each time!

Arithmetic Microoperations
Arithmetic Circuit



4-bit Arithmetic Circuit

Shift Microoperations cont.



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

Shift Microoperations: Logical Shifts

- Transfers 0 through the serial input
- Logical Shift Right: R1 ← shr R1

The same

- Logical Shift Left: R2 ← shl R2

The same

Logical Shift Left

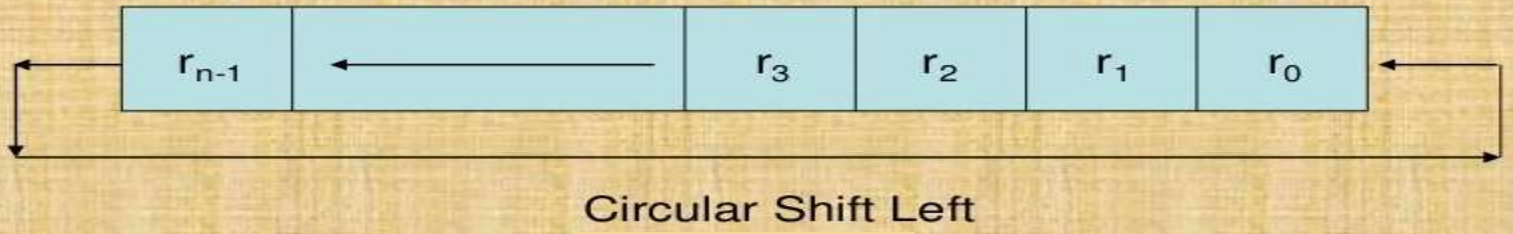
Shift Microoperations: Circular Shifts (Rotate Operation)

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

The same

- Circular Shift Left: R2 ← cil R2

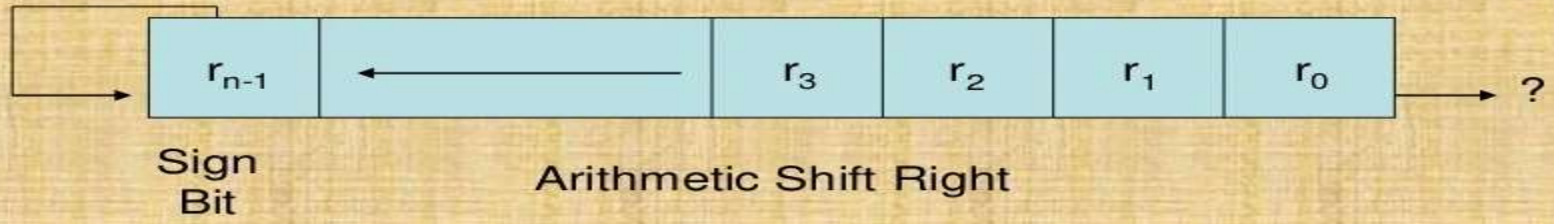
The same

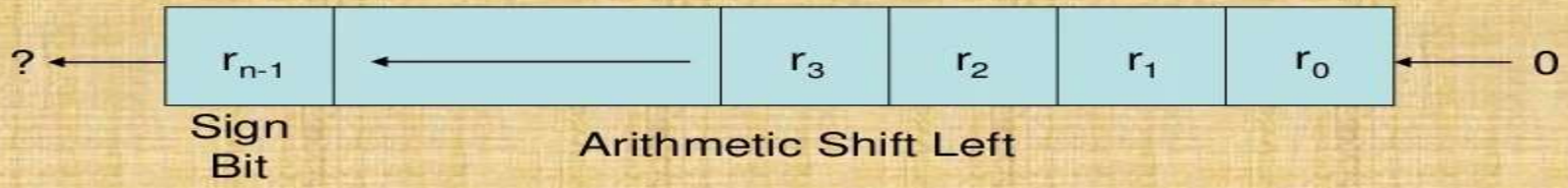


Shift Microoperations
Arithmetic Shifts

- 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
- An overflow may occur in arithmetic shift-left, and occurs when the sign bit is changed (sign reversal)

Shift Microoperations
Arithmetic Shifts cont.

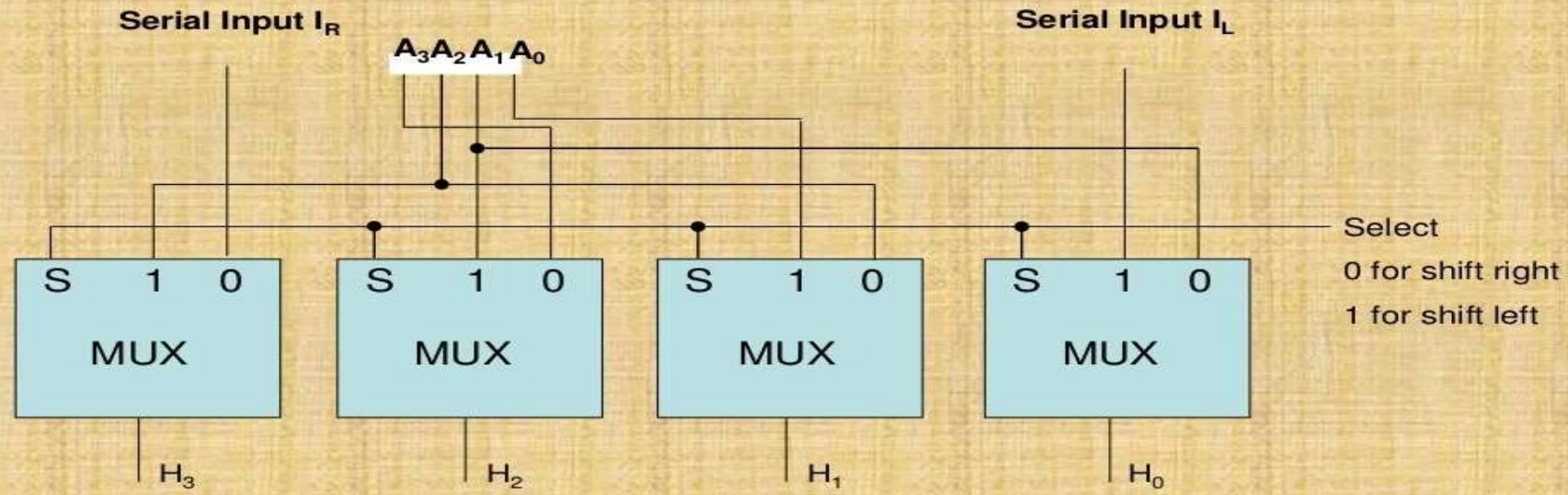




Shift Microoperations cont.

- 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

Shift Microoperations Hardware Implementation



4-bit Combinational Circuit Shifter

Logical micro-operations

- Specify binary operations on the strings of bits in registers
  - Logic micro-operations are bit-wise operations, i.e., they work on the individual bits of data
- There are, in principle, 16 different logic functions that can be defined over two binary input variables

A	B	F_0	F_1	F_2	…	F_{13}	F_{14}	F_{15}
0	0	0	0	0	…	1	1	1
0	1	0	0	0	…	1	1	1
1	0	0	0	1	…	0	1	1
1	1	0	1	0	…	1	0	1

- 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

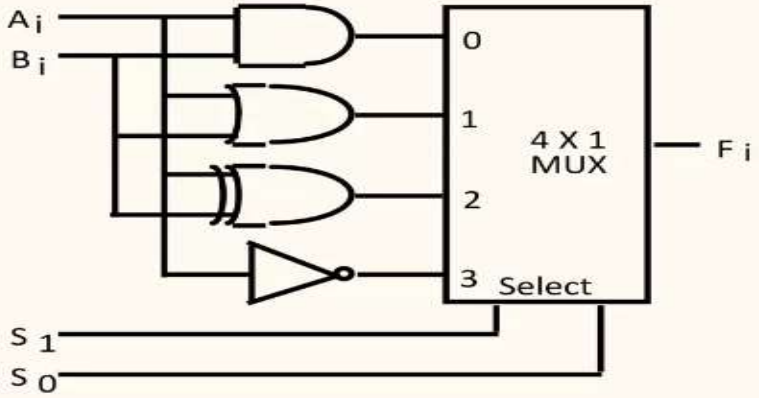
Logical Micro-operation

- List of Logic Micro-operations
  - 16 different logic operations with 2 binary vars.
  - n binary vars $\rightarrow 2^{2^n}$ functions

- Truth tables for 16 functions of 2 variables and the corresponding 16 logic micro-operations

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

Hardware implementation of logic Micro-operation



Function table

S_1	S_0	Output	μ-operation
0	0	F = A ∧ B	AND
0	1	F = A ∨ B	OR
1	0	F = A ⊕ B	XOR
1	1	F = A’	Complement