CSE211

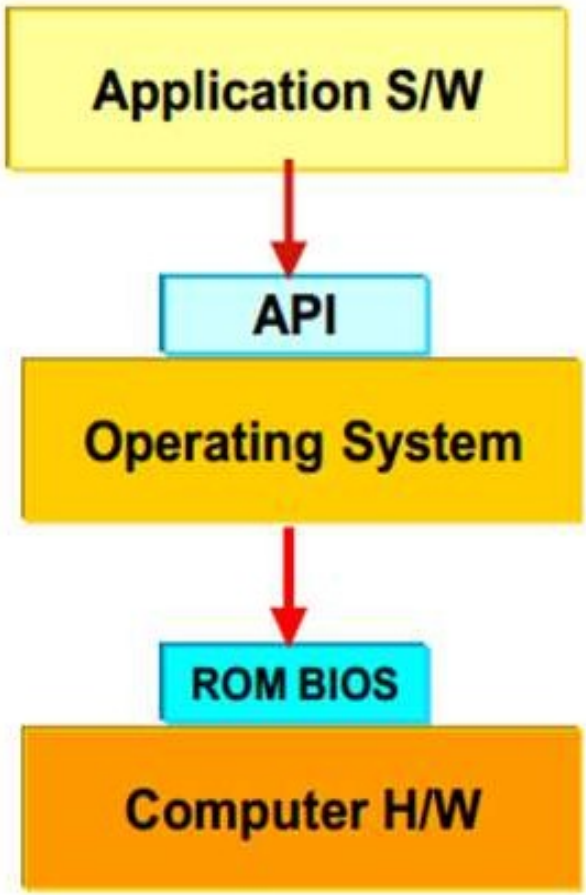
Computer Organization and Design

Lecture : 3 Tutorial: 1 Practical: 0 Credit: 4

1-1 Digital Computers

- Digital – A limited number of discrete value
- Bit – A Binary Digit
- Program – A Sequence of instructions

- Computer = H/W + S/W
- Program(S/W)
  - A sequence of instruction
  - S/W = Program + Data
    - The data that are manipulated by the program constitute the data base
  - Application S/W
    - DB, word processor, Spread Sheet
  - System S/W
    - OS, Firmware, Compiler, Device Driver



1-1 Digital Computers

- Computer Hardware
  - CPU
  - Memory
    - Program Memory(ROM)
    - Data Memory(RAM)
  - I/O Device
    - Interface
    - Input Device: Keyboard, Mouse, Scanner
    - Output Device: Printer, Plotter, Display
    - Storage Device(I/O): FDD, HDD, MOD

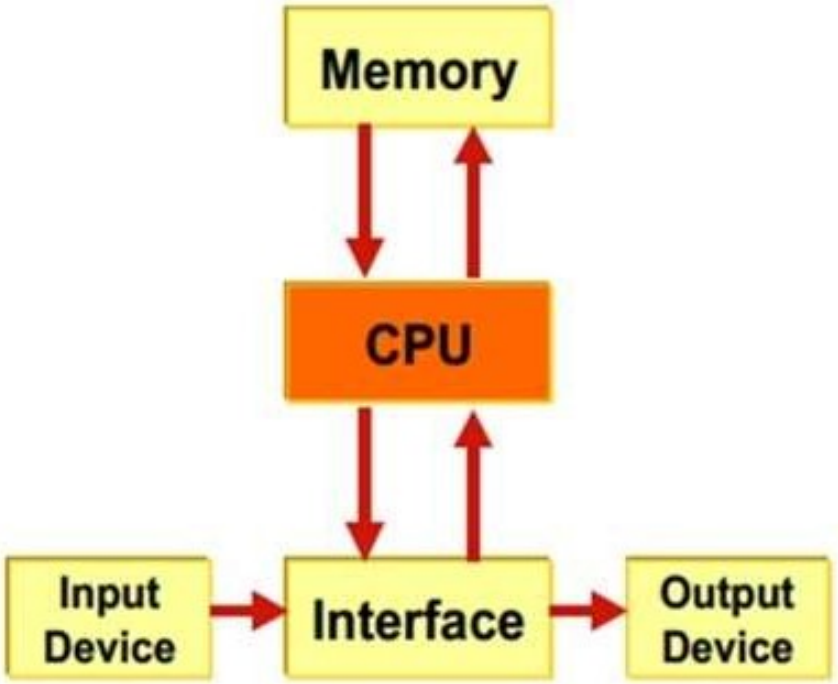


Figure ***Block Diagram of a digital Computer***

1-1 Digital Computers

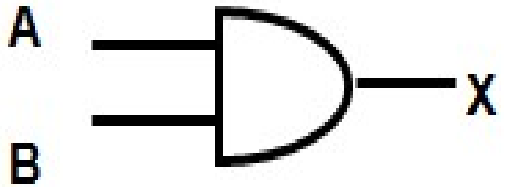
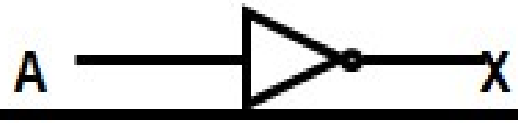
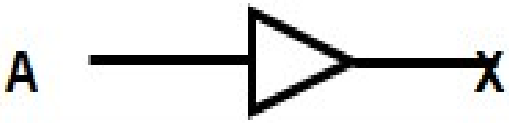
- 3 different point of view(Computer Hardware)
  - Computer Organization
    - H/W components operation/connection
  - Computer Design
    - H/W Design/Implementation
  - Computer Architecture
    - Structure and behavior of the computer as seen by the user
    - Information format, Instruction set, memory addressing, CPU, I/O, Memory
- ISA(Instruction Set Architecture)
  - the attributes of a system as seen by the programmer, i.e., the conceptual structure and functional behavior, as distinct from the organization of the data flows and controls, the logic design, and the physical implementation.

    - Amdahl, Blaaw, and Brooks(1964)

1-2 Logic Gates

- ADC(Analog to Digital Conversion)
  - Signal → Physical Quantity → Binary Information
    Discrete Value
    { 0 : 0.5 ↓
    { 1 : 3 ↑
- Gate
  - The manipulation of binary information is done by logic circuit called "gate".
- *Fig.* *Digital Logic Gates*
  - AND, OR, INVERTER, BUFFER, NAND, NOR, XOR, XNOR

1-2 Logic Gates

Name	Symbol	Function	Truth Table
AND		X = A • B or X = AB	A B | X 0 0 | 0 0 1 | 0 1 0 | 0 1 1 | 1
OR		X = A + B	A B | X 0 0 | 0 0 1 | 1 1 0 | 1 1 1 | 1
I		X = A’	A | X 0 | 1 1 | 0
Buffer		X = A	A | X 0 | 0 1 | 1

1-2 Logic Gates

Name	Symbol	Function	Truth Table
NAND		X = (AB)'	A B | X 0 0 | 1 0 1 | 1 1 0 | 1 1 1 | 0
NOR		X = (A + B)'	A B | X 0 0 | 1 0 1 | 0 1 0 | 0 1 1 | 0
XOR Exclusive OR		X = A ⊕ B or X = A'B + AB'	A B | X 0 0 | 0 0 1 | 1 1 0 | 1 1 1 | 0
XNOR Exclusive NOR or Equivalence		X = (A ⊕ B)' or X = A'B'+ AB	A B | X 0 0 | 1 0 1 | 0 1 0 | 0 1 1 | 1

Combinational Circuits

- In this output depends only upon present input.
- Speed is fast.
- It is designed easy.
- There is no feedback between input and output.
- This is time independent.
- Elementary building blocks: Logic gates
- Used for arithmetic as well as boolean operations.
- Combinational circuits don't have capability to store any state.
- As combinational circuits don't have clock, they don't require triggering.
- These circuits do not have any memory element.

Examples – Encoder, Decoder, Multiplexer, Demultiplexer

Sequential Circuits

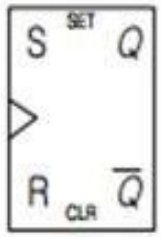
- In this output depends upon present as well as past input.
- Speed is slow comparative to combinational circuits.
- It is designed tough as compared to combinational circuits.
- There exists a feedback path between input and output.
- This is time dependent.
- Elementary building blocks: Flip-flops Mainly used for storing data. Sequential circuits have capability to store any state or to retain earlier state.
- As sequential circuits are clock dependent they need triggering.
- These circuits have memory element.

Examples – Flip-flops, Counters

1-6 Flip-Flops

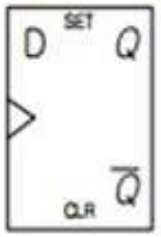
Combinational Circuit = Gate
Sequential Circuit = Gate + F/F

- Flip-Flop
  - The *storage elements* employed in clocked *sequential circuit*
  - A binary cell capable of storing one bit of information
- SR*(Set/Reset)* F/F



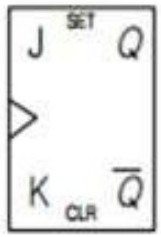
S	R	Q(t+1)	
0	0	Q(t)	no change
0	1	0	clear to 0
1	0	1	set to 1
1	1	?	Indeterminate

- D*(Data)* F/F



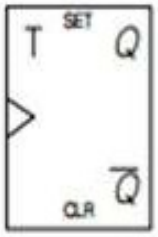
D	Q(t+1)	
0	0	clear to 0
1	1	set to 1

  - "no change" condition
    1) Disable Clock
    2) Feedback output into input
- JK*(Jack/King)* F/F



J	K	Q(t+1)	
0	0	Q(t)	no change
0	1	0	clear to 0
1	0	1	set to 1
1	1	Q(t)'	Complement

  - JK F/F is a refinement of the SR F/F
  - The indeterminate condition of the SR type is defined in complement
- T*(Toggle)* F/F

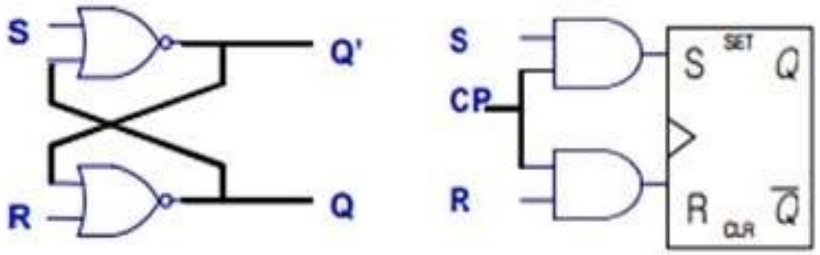


T	Q(t+1)	
0	Q(t)	no change
1	Q'(t)	Complement

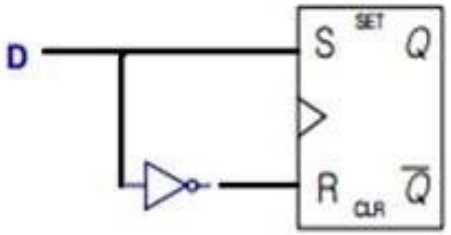
  - T=1(J=K=1), T=0(J=K=0)

1-6 Flip-Flops

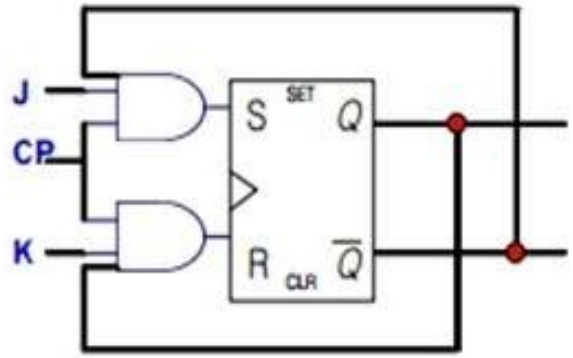
- SR*(Set/Reset)* F/F



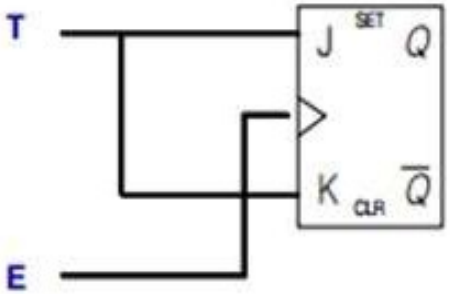
- D*(Data)* F/F



- JK*(Jack/King)* F/F

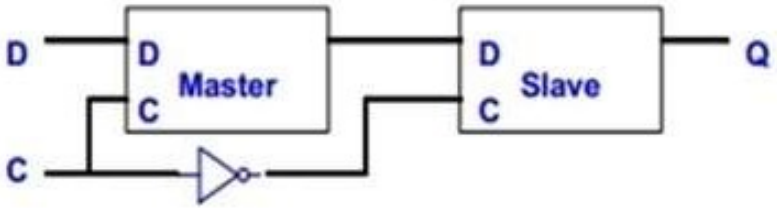


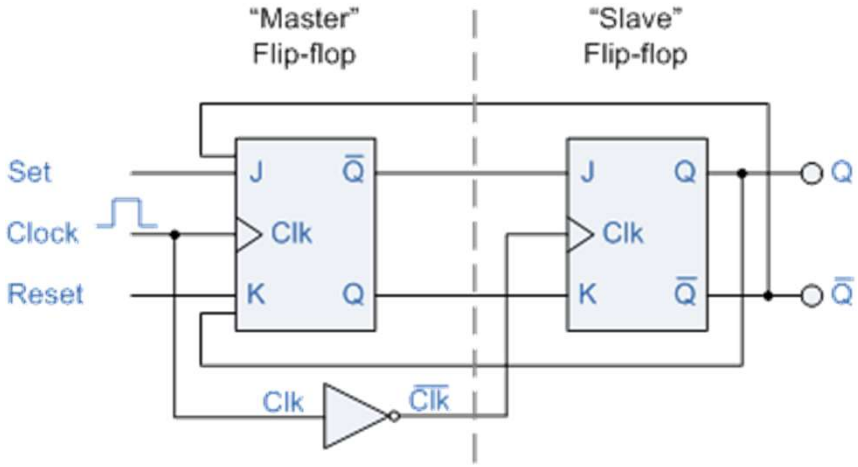
- T*(Toggle)* F/F

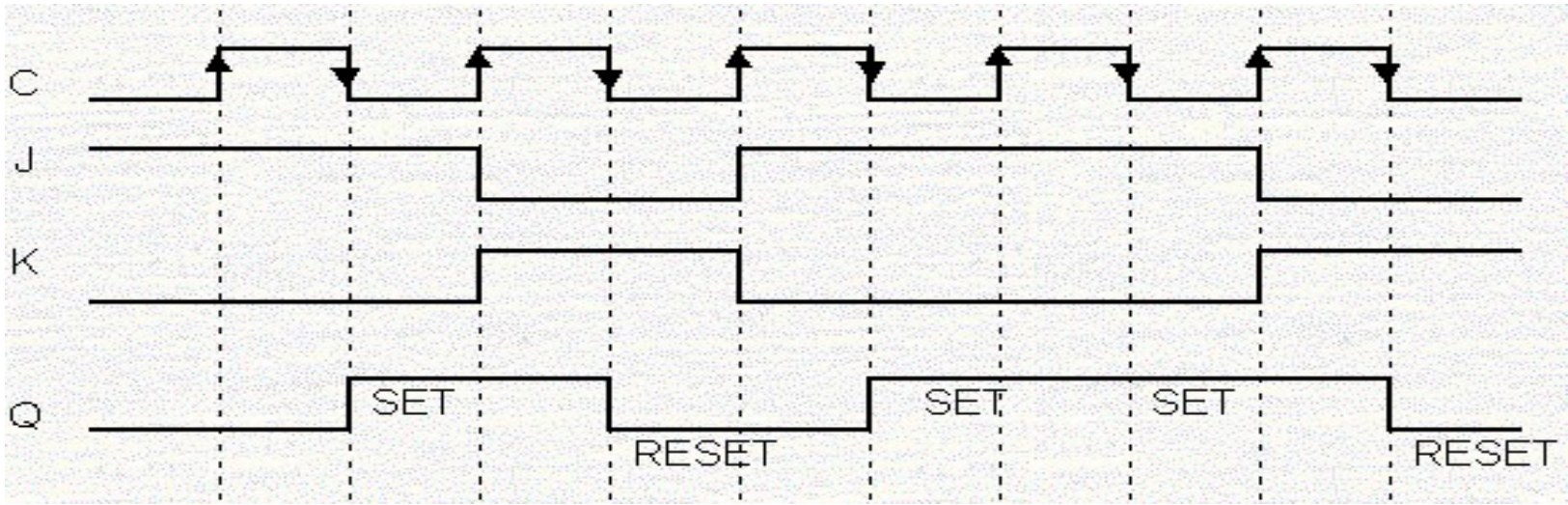


1-6 Flip-Flops

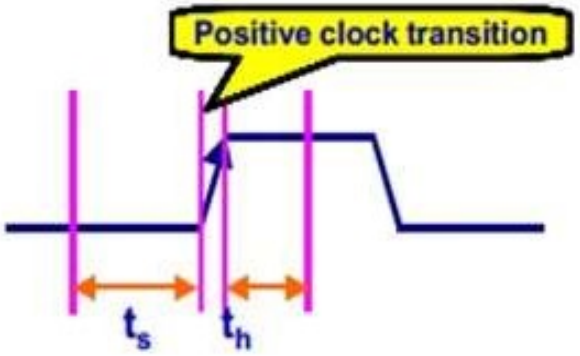
- Master – Slave D*(Data)* F/F







1-6 Flip-Flops



- Edge-Triggered F/F
  - State Change : *Clock Pulse*
    - Rising Edge(positive-edge transition)
    - Falling Edge(negative-edge transition)
  - Setup time(20ns)
    - minimum time that D input must remain at constant value before the transition.
  - Hold time(5ns)
    - minimum time that D input must not change after the positive transition.
  - Propagation delay(max 50ns)
    - time between the clock input and the response in Q
  - Master-Slave F/F

Integrated Circuits

An IC is a small silicon semiconductors crystal called chip containing the electronic components for digital gates.

- Various gates are interconnected inside chip to form required circuit.
- Chip is mounted in ceramic/plastic container connected to external pin

Small scale Integration (SSI) : less than 10 gates

Medium Scale Integration(MSI) : between 10 to 200 gates
(decoders, adders, registers)

Large Scale Integration(LSI) : between 200 and few thousands gates
(Processors, Memory Chips)

Very Large Scale Integration (VLSI) : Thousands of gate within single package (Large Memory Arrays, Complex Microcomputer Chips)

CSE211

Computer Organization and Design

Lecture : 3 Tutorial: 2 Practical: 0 Credit: 4

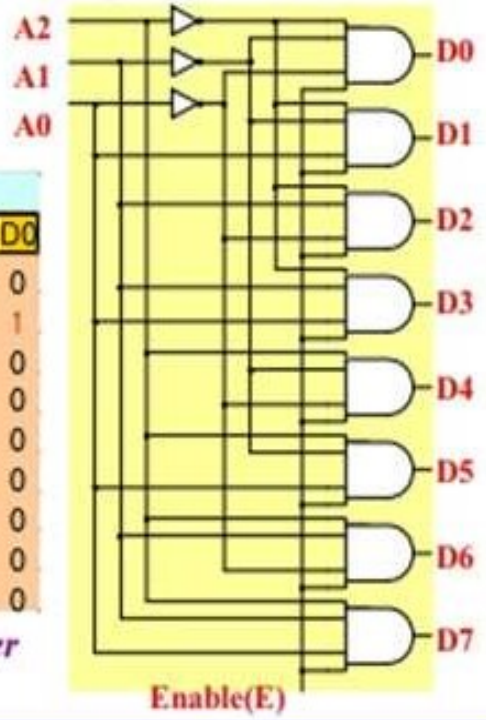
Unit 1 : Basics of Digital Electronics

- Introduction
- Decoder
- Encoder
- Multiplexers
- Demultiplexer
- Registers

2-2 Decoder/Encoder

- Decoder
  - A combinational circuit that converts binary information from the ***n*** coded inputs to a maximum of 2^n unique outputs
  - ***n*-to-*m*** line decoder = ***n x m*** decoder
    - ***n*** inputs, ***m*** outputs
  - If the n-bit coded information has unused bit combinations, the decoder may have less than 2^n outputs
    - $m \leq 2^n$
- 3-to-8 Decoder
  - A Binary-to-octal conversion
  - Logic Diagram : *Fig. 2-1*
  - Truth Table : *Tab. 2-1*
  - Commercial decoders include one or more Enable Input(E)

Fig. 2-1 3-to-8 Decoder



Enable	Inputs			Outputs							
E	A2	A1	A0	D7	D6	D5	D4	D3	D2	D1	D0
0	x	x	x	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	1
1	0	0	1	0	0	0	0	0	0	1	0
1	0	1	0	0	0	0	0	0	1	0	0
1	0	1	1	0	0	0	0	1	0	0	0
1	1	0	0	0	0	0	1	0	0	0	0
1	1	0	1	0	0	1	0	0	0	0	0
1	1	1	0	0	1	0	0	0	0	0	0
1	1	1	1	1	0	0	0	0	0	0	0

Tab. 2-1 Truth table for 3-to-8 Decoder

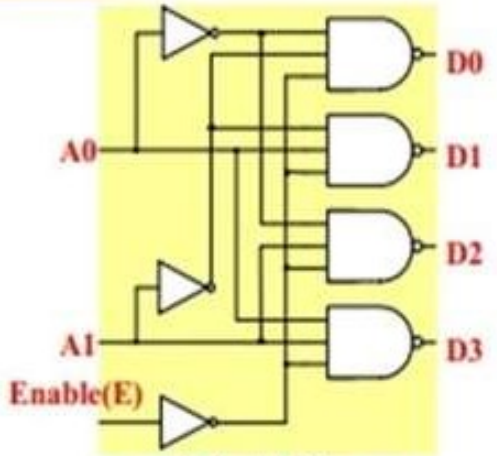
2-2 Decoder/Encoder

- NAND Gate Decoder
  - Constructed with NAND instead of AND gates
  - Logic Diagram/Truth Table : *Fig. 2-2*

* Active Low Output
* Fig. 2-1 3-to-8 Decoder ≒ Active High Output

Enable	Input		Output			
E	A1	A0	D0	D1	D2	D3
0	0	0	0	1	1	1
0	0	1	1	0	1	1
0	1	0	1	1	0	1
0	1	1	1	1	1	0
1	x	x	1	1	1	1

(a) Truth Table



(b) Logic Diagram

Fig. 2-2 2-to-4 Decoder with NAND gates

- Decoder Expansion
  - Constructed decoder : *Fig. 2-3*
  - 3 X 8 Decoder constructed with two 2 X 4 Decoder
- Encoder
  - Inverse Operation of a decoder
  - 2^n input, n output
  - Truth Table : *Tab. 2-2*
    - 3 OR Gates Implementation
      - » A0 = D1 + D3 + D5 + D7
      - » A1 = D2 + D3 + D6 + D7
      - » A2 = D4 + D5 + D6 + D7

Tab. 2-2 Truth Table for Encoder

Inputs								Outputs		
D7	D6	D5	D4	D3	D2	D1	D0	A2	A1	A0
0	0	0	0	0	0	0	1	0	0	0
0	0	0	0	0	0	1	0	0	0	1
0	0	0	0	0	1	0	0	0	1	0
0	0	0	0	1	0	0	0	0	1	1
0	0	0	1	0	0	0	0	1	0	0
0	0	1	0	0	0	0	0	1	0	1
0	1	0	0	0	0	0	0	1	1	0
1	0	0	0	0	0	0	0	1	1	1

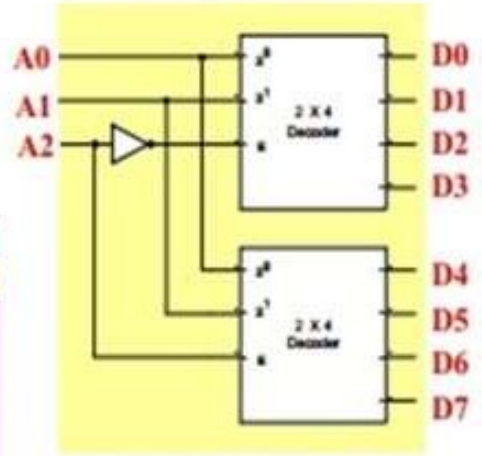


Fig. 2-3 A 3-to-8 Decoder constructed with two with 2-to-4 Decoder

2-3 Multiplexers

- Multiplexer(Mux)
  - A combinational circuit that receives binary information from one of 2^n input data lines and directs it to a single output line
  - A 2^n -to 1 multiplexer has *2^n input data lines* and *n input selection lines*(Data Selector)
  - 4-to-1 multiplexer Diagram : ***Fig. 2-4***
  - 4-to-1 multiplexer Function Table : ***Tab. 2-3***

Select		Output
S1	S0	Y
0	0	I_0
0	1	I_1
1	0	I_2
1	1	I_3

Tab. 2-3 Function Table for 4-to-1 line Multiplexter

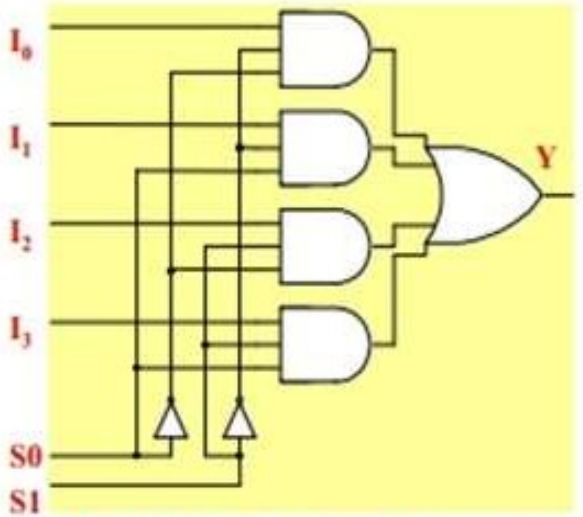


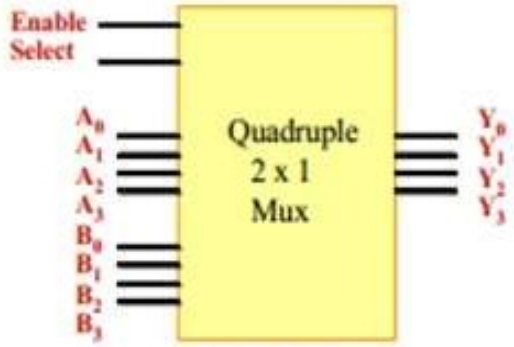
Fig. 2-4 4-to-1 Line Multiplexer

- Quadruple 2-to-1 Multiplexer
  - Quadruple 2-to-1 Multiplexer : ***Fig. 2-5***

Select		Output
E	S	Y
0	0	All 0's
1	0	A
1	1	B

(a) Function Table

Fig. 2-5 Quadruple 2-to-1 line Multiplexter



(b) Block Diagram

2-3 Multiplexers

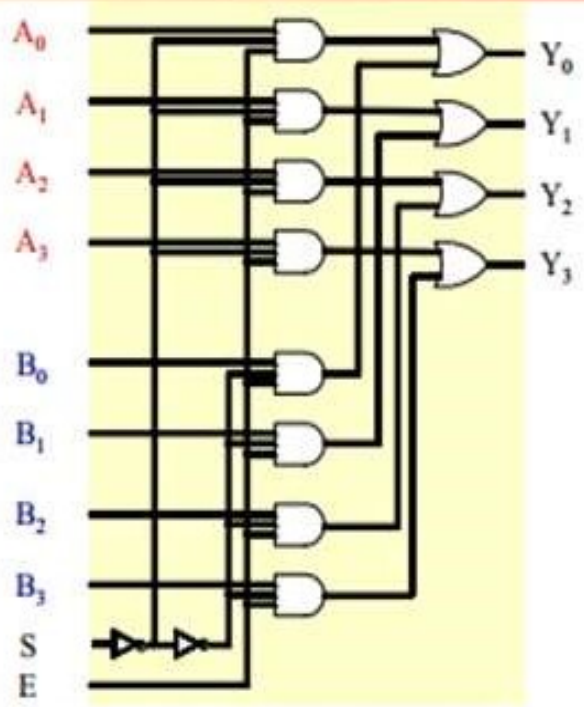
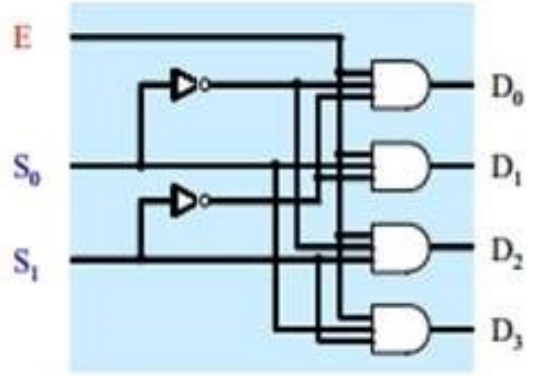


Fig A. Combinational logic diagram with four 2×1 multiplexer



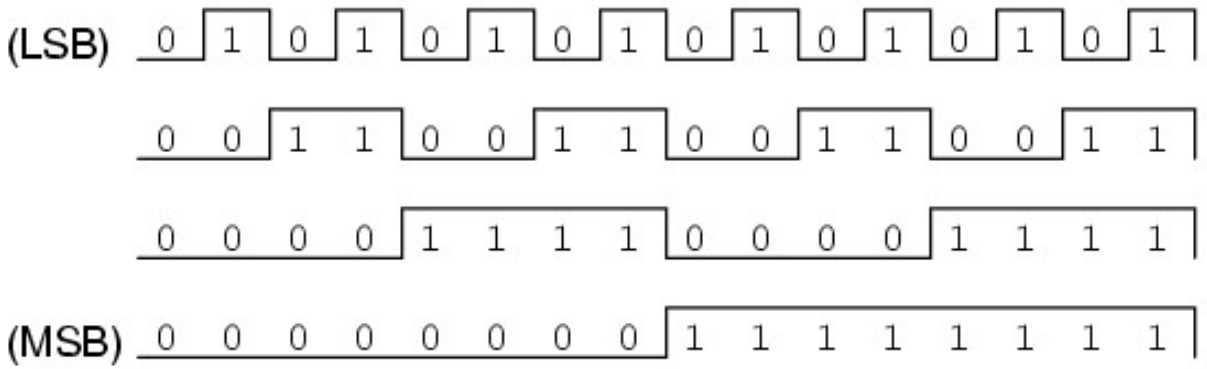
E → Data input
S_0, S_1 → Select Data

Fig B. Demultiplexer

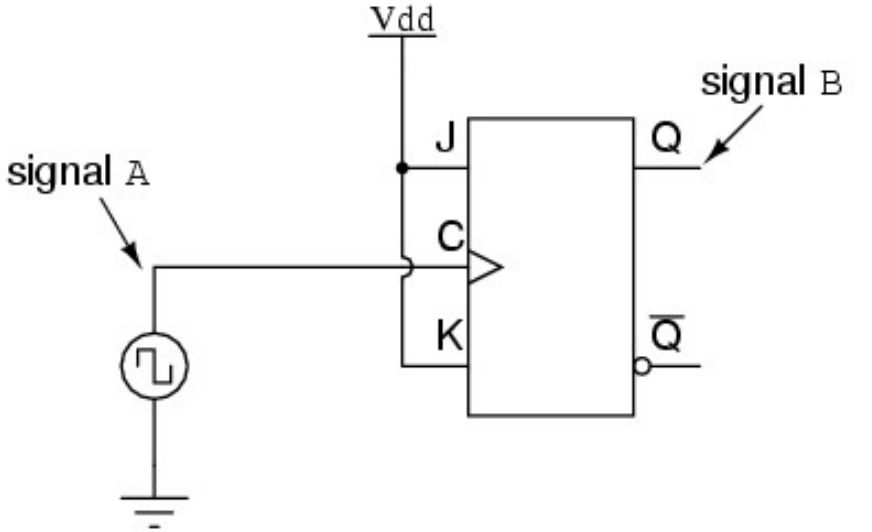
A **Demultiplexer,** sometimes abbreviated **DMUX** is a circuit that has one input and more than one output. It is used when a circuit wishes to send a signal to one of many devices

Binary counter

- A register that goes through a predetermined sequence of states upon the application of input pulses is called a counter. The input pulses may be clock pulses or may originate from an external source. They may occur at uniform intervals of time or at random.
- A counter that follows the binary number sequence is called a binary counter. An n-bit binary counter is a register of n flip-flops and associated gates that follows a sequence of states according to the binary count of n bits, from 0 to 2" - 1.



•J-K flip-flops are ideally suited for this task, because they have the ability to "toggle" their output state at the command of a clock pulse when both J and K inputs are made "high" (1)



2-4 Registers

- Register
  - A group of flip-flops with each flip-flop capable of storing one bit of information
  - An n-bit register has a group of n flip-flops and is capable of storing any binary information of n bits
  - The simplest register consists only of flip-flops, with no external gate : *Fig. 2-6*
  - A clock input C will load all four inputs in parallel
    - The clock must be *inhibited* if the content of the register must be left unchanged
- Register with Parallel Load
  - A 4-bit register with a load control input : *Fig. 2-7*
  - The clock inputs receive clock pulses at all times
  - The buffer gate in the clock input will increase "fan-out"
  - Load Input
    - 1 : Four input transfer
    - 0 : Input inhibited, Feedback from output to input(**no change**)

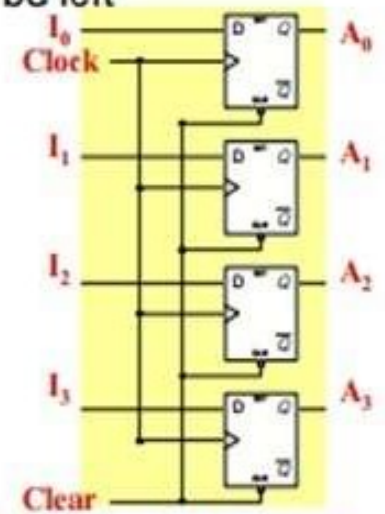


Fig. 2-6 4-bit register

2-4 Registers

- When the load input is 1 , the data in the four inputs are transferred into the register with the next positive transition of a clock pulse

- When the load input is 0, the data inputs are inhibited and the D-output of flip flop are connected to their inputs.

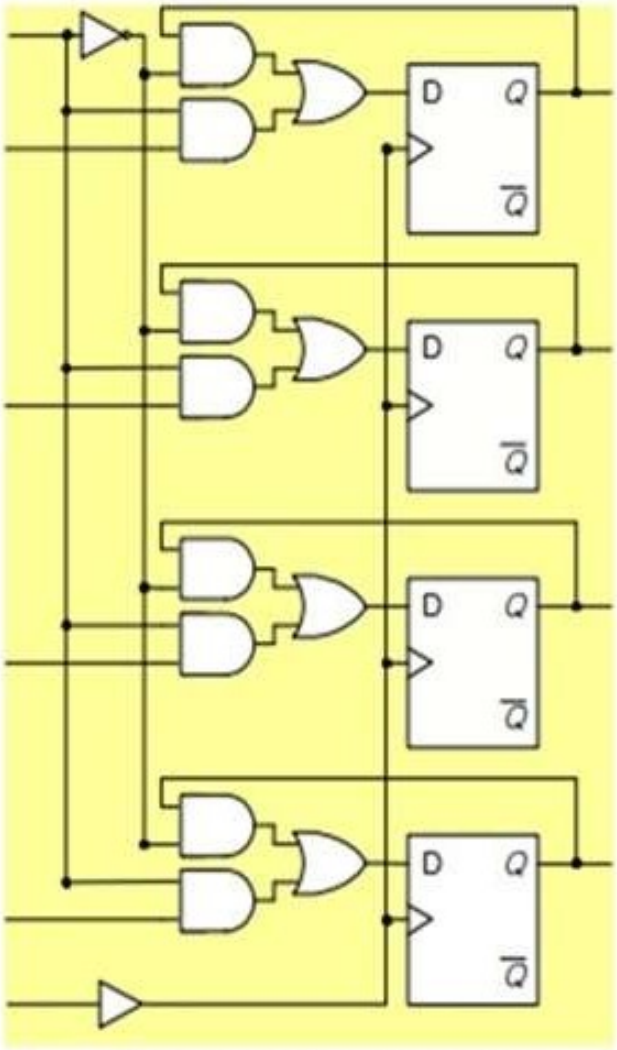


Fig. 2-7 4-bit register with parallel load

2-5 Shift Registers

- Shift Register
  - A register capable of shifting its binary information in one or both directions
  - The logical configuration of a shift register consists of a chain of flip-flops in cascade
  - The simplest possible shift register uses only flip-flops : *Fig. 2-8*
  - The ***serial input*** determines what goes into the leftmost position during the shift
  - The ***serial output*** is taken from the output of the rightmost flip-flop

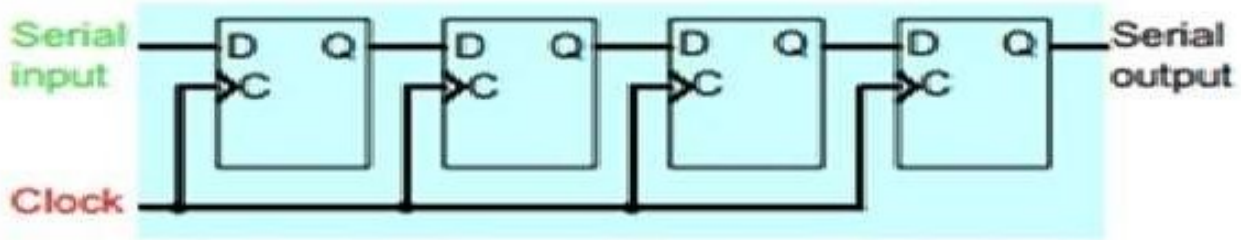


Fig. 2-8 4-bit shift register

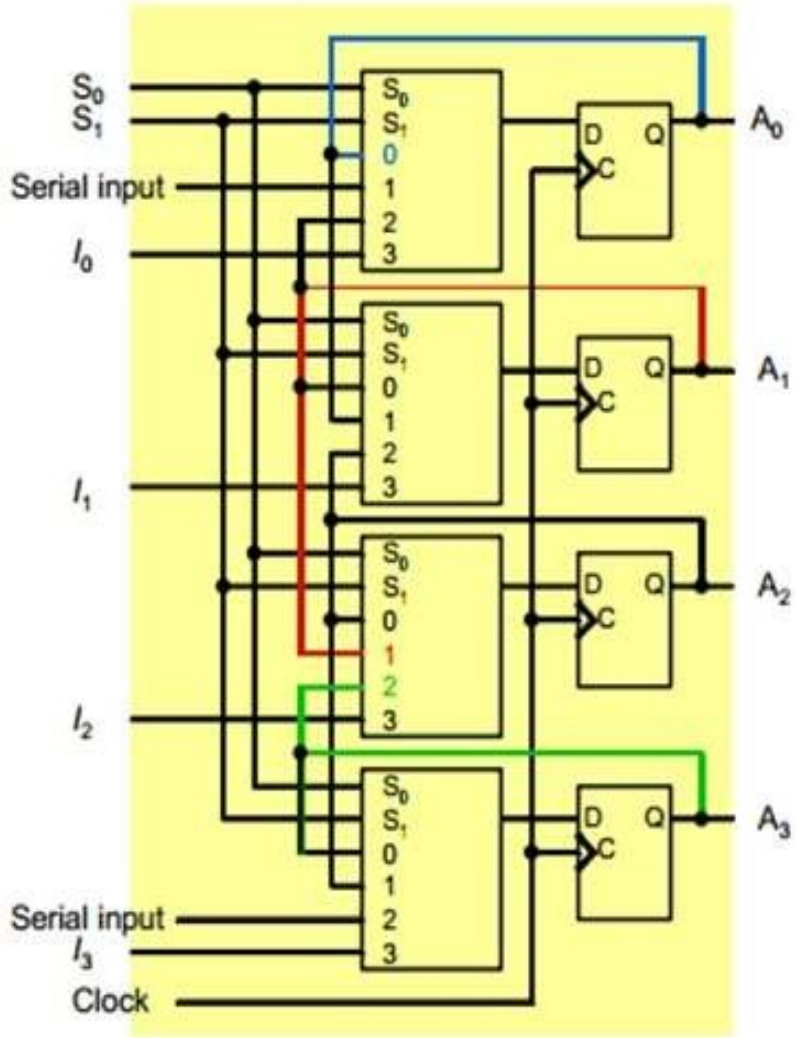
2-5 Shift Registers

- Bidirectional Shift Register with Parallel Load
  - A register capable of shifting in *one direction only* is called a ***unidirectional shift register***
  - A register that can shift in *both directions* is called a ***bidirectional shift register***
  - The most general shift register has all the capabilities listed below:
    - An input clock pulse to synchronize all operations
    - A shift-right /left (serial output/input)
    - A parallel load, n parallel output lines
    - The register unchanged even though clock pulses are applied continuously
  - 4-bit bidirectional shift register with parallel load : *Fig. 2-9*
    - 4 X 1 Mux = 4 , D F/F = 4

Tab. 2-4 Function Table for Register of Fig. 2-9

Mode		Operation
S1	S0	
0	0	No chage
0	1	Shift right(down)
1	0	shift left(up)
1	1	Parallel load

2-5 Shift Registers



$S_1S_0 = 00$: $A_i \rightarrow A_i$ (No change)
$S_1S_0 = 01$: $A_{i-1} \rightarrow A_i$ (Shift)
$S_1S_0 = 10$: $A_{i+1} \rightarrow A_i$ (Shift)
$S_1S_0 = 11$: Parallel load

- Shift Register

Interface digital systems situated remotely from each other

Fig. 2-9 Bidirectional shift register

Overview

- **Register Transfer Language**
- **Register Transfer**
- Bus and Memory Transfers
- Logic Micro-operations
- Shift Micro-operations
- Arithmetic Logic Shift Unit

Register Transfer Language

- **Combinational and sequential circuits can be used to create simple digital systems.**
- **These are the low-level building blocks of a digital computer.**
- **Simple digital systems are frequently characterized in terms of**
  - **the registers they contain, and**
  - **the operations that are performed on data stored in them**
- **The operations executed on the data in registers are called micro-operations e.g. shift, count, clear and load**

Register Transfer Language

Internal hardware organization of a digital computer :

- **Set of registers and their functions**
- **Sequence of microoperations performed on binary information stored in registers**
- **Control signals that initiate the sequence of micro-operations (to perform the functions)**

Register Transfer Language

- **Rather than specifying a digital system in words, a specific notation is used, Register Transfer Language**
- **The symbolic notation used to describe the micro operation transfer among register is called a register transfer language**
- **For any function of the computer, the register transfer language can be used to describe the (sequence of) micro-operations**
- **Register transfer language**
  - **A symbolic language**
  - **A convenient tool for describing the internal organization of digital computers in concise/precise manner.**

Register Transfer Language

- **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**

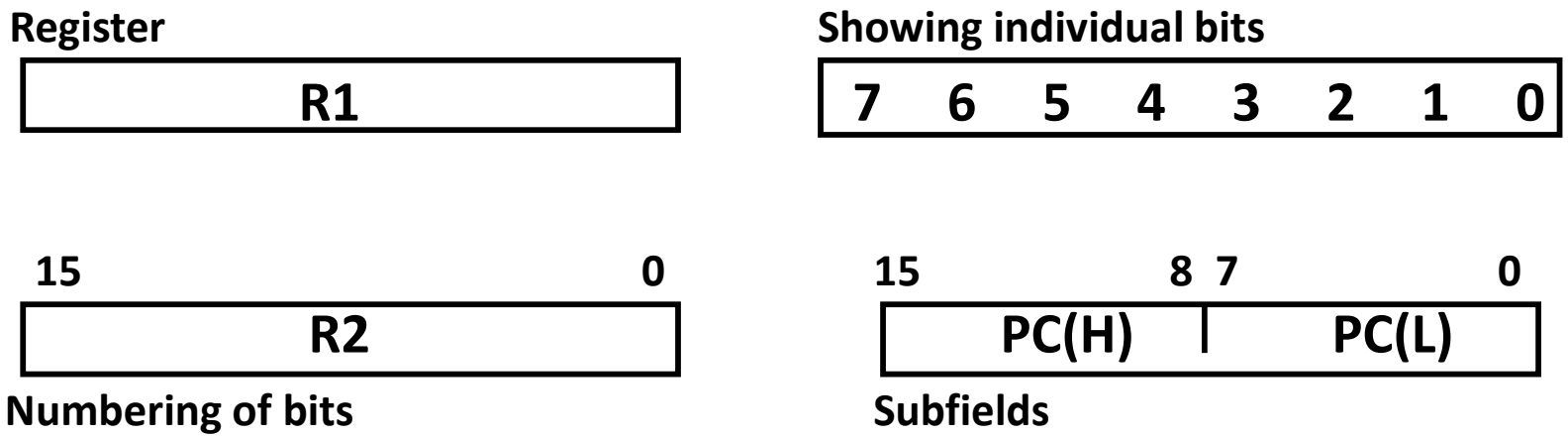
- **Registers and their contents can be viewed and represented in *various ways***
  - **A register can be viewed as a single entity:**

Register Transfer Language

- **Designation of a register**

  - a register
  - portion of a register
  - a bit of a register

- **Common ways of drawing the block diagram of a register**



Register Transfer Language

- **Copying the contents of one register to another is a register transfer**
- **A register transfer is indicated as**

R2 ← R1

  - **In this case the contents of register R1 are copied (loaded) into register R2**
  - **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**

Register Transfer Language

- **A register transfer such as**

  R3 ← 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**
- **This is represented as:**

P: R2 ← R1

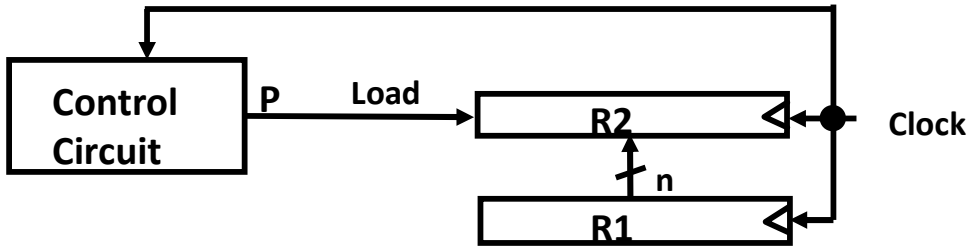
Which means "if P = 1, then load the contents of register R1 into register R2", i.e., if (P = 1) then (R2 ← R1)

Hardware Implementation of Controlled Transfers

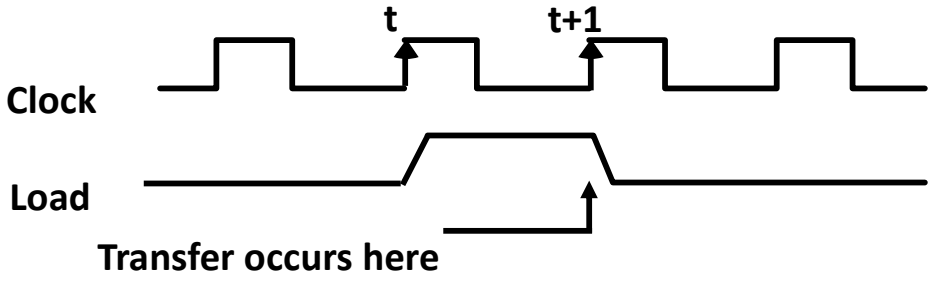
Implementation of controlled transfer

P: R2 ← R1

Block diagram



Timing diagram



- **The same clock controls the circuits that generate the control function and the destination register**
- **Registers are assumed to use *positive-edge-triggered* flip-flops**

Basic Symbols in Register Transfer

Symbols	Description	Examples
Capital letters & Numerals	Denotes a register	MAR, R2
Parentheses ()	Denotes a part of a register	R2(0-7), R2(L)
Arrow ←	Denotes transfer of information	R2 ← R1
Colon :	Denotes termination of control function	P:
Comma ,	Separates two micro-operations	A ← B, B ← A

Overview

- Register Transfer Language
- Register Transfer
- **Bus and Memory Transfers**
- Logic Micro-operations
- Shift Micro-operations
- Arithmetic Logic Shift Unit

Connecting Registers - Bus Transfer

- **In a digital system with many registers, it is impractical to have data and control lines to directly allow each register to be loaded with the contents of every possible other registers**

- **To completely connect n registers → n(n-1) lines**
- **$O(n^2)$ cost**
  - **This is not a realistic approach to use in a large digital system**

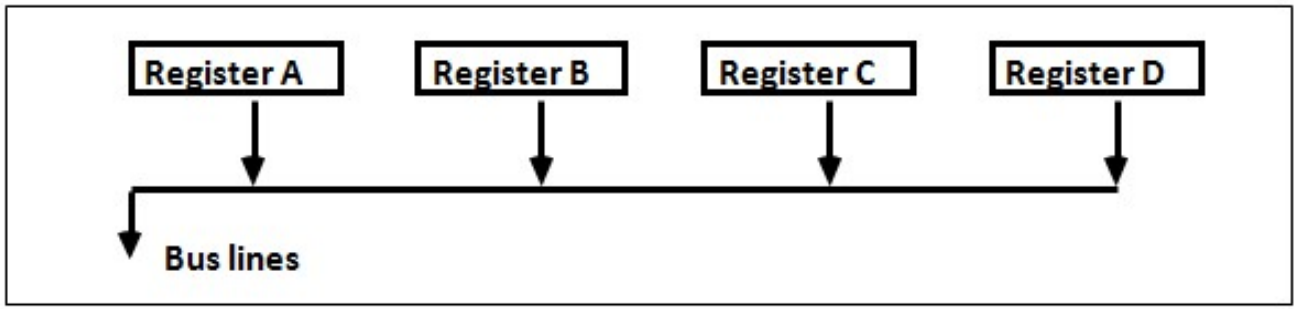
- **Instead, take a different approach**
- **Have one centralized set of circuits for data transfer – the bus**

- **BUS STRUCTURE CONSISTS OF SET OF COMMON LINES, ONE FOR EACH BIT OF A REGISTER THROUGH WHICH BINARY INFORMATION IS TRANSFERRED ONE AT A TIME**

- **Have control circuits to select which register is the source, and which is the destination**

Connecting Registers - Bus Transfer

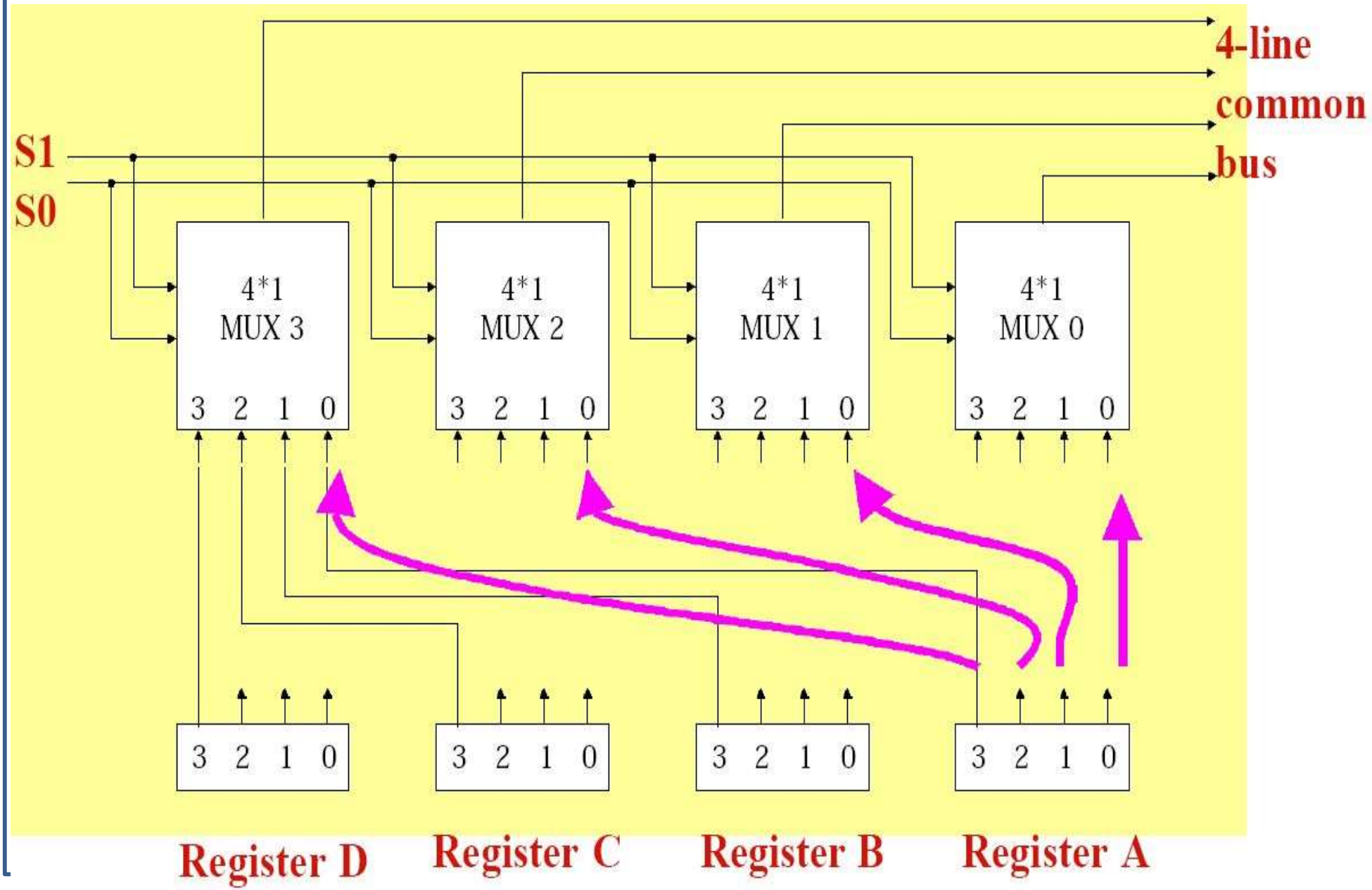
From a register to bus: BUS ← R



- One way of constructing common bus system is with multiplexers
- Multiplexer selects the source register whose binary information is kept on the bus.

- Construction of bus system for 4 register (Next Fig)
  - 4 bit register X 4
  - four 4X1 multiplexer
  - Bus selection S0, S1

Connecting Registers - Bus Transfer



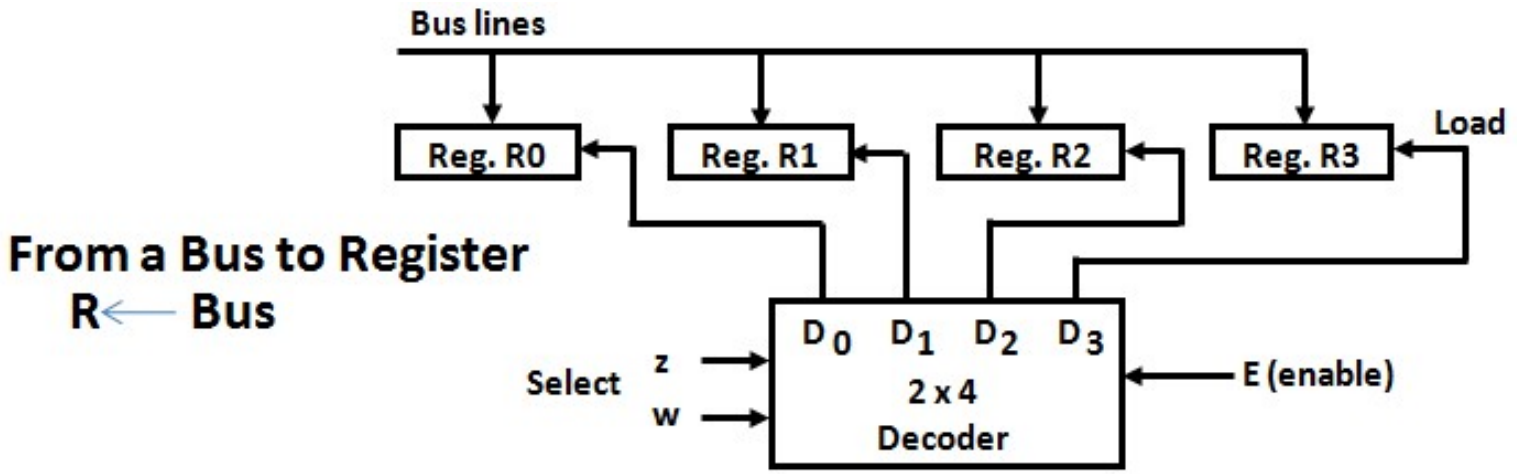
Connecting Registers - Bus Transfer

- For a bus system to multiplex k registers of n bits each
  - No. of multiplexer = n = No. of bits
  - Size of each multiplexer = k x 1, k data lines in each

  MUX

- **Construction of bus system for 8 register with 16 bits**
  - **16 bit register X 8**
  - **Eight 16X1 multiplexer**
  - **Bus selection S0, S1, S2**

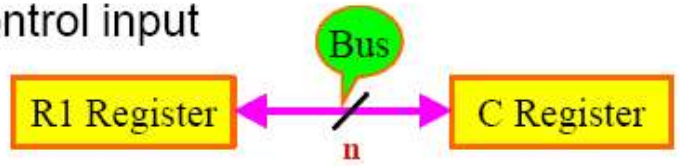
Connecting Registers - Bus Transfer



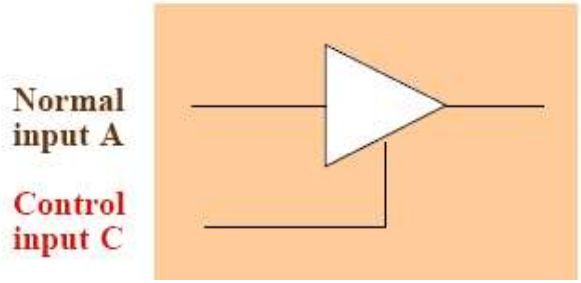
Connecting Registers - Bus Transfer

- Bus Transfer
  - The content of register C is placed on the bus, and the content of the bus is loaded into register R1 by activating its load control input

$$\left.\begin{array}{l} Bus \leftarrow C,\ \ R1 \leftarrow Bus \\ R1 \leftarrow C \end{array}\right\} =$$

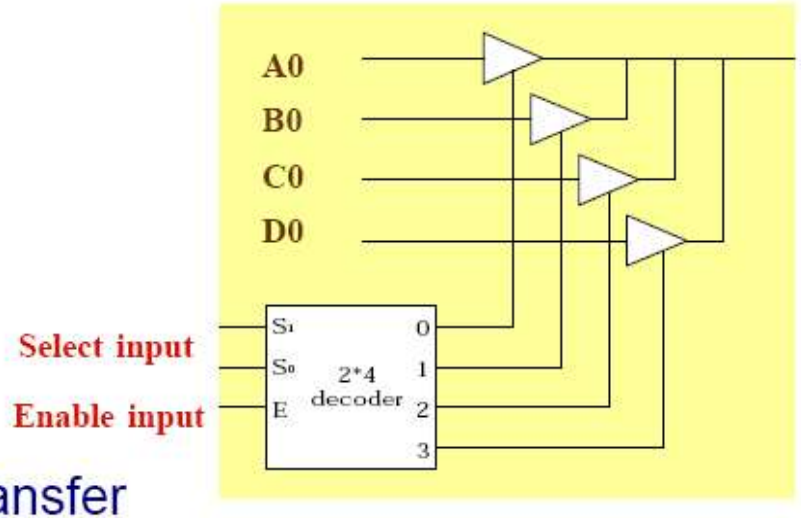


- Three-State Bus Buffers
  - A bus system can be constructed with ***three-state gates*** *instead of* ***multiplexers***
  - Tri-State : 0, 1, High-impedance(***Open circuit***)
  - Buffer
    - » A device designed to be inserted between other devices to match impedance, to prevent mixed interactions, and ***to supply additional drive or relay capability***
    - » Buffer types are classified as inverting or noninverting
  - Tri-state buffer gate : Fig. 4-4
    - » When control input =1 : The output is enabled(output Y = input A)
    - » When control input =0 : The output is disabled(output Y = high-impedance)



Connecting Registers - Bus Transfer

- The construction of a bus system with tri-state buffer : ***Fig.***
  - The outputs of four buffer are connected together to form a single bus line(Tri-state buffer
  - No more than one buffer may be in the active state at any given time(2 X 4 Decoder
  - To construct a common bus for 4 register with 4 bit : Fig.



AR: Address Reg.
DR: Data Reg.
M : Memory Word(Data)

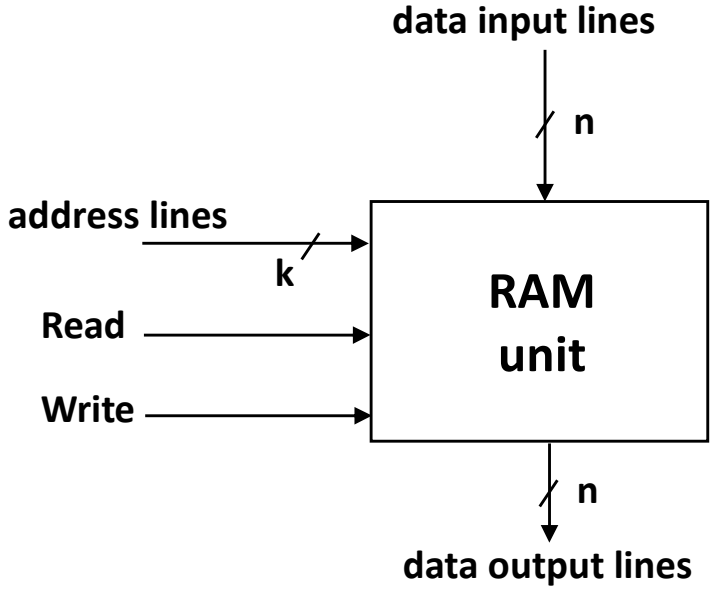
$READ: \quad DR \leftarrow M[AR]$

$WRITE: \quad M[AR] \leftarrow R1$

- Memory Transfer
  - Memory read : A transfer information into DR from the memory word M selected by the address in AR
  - Memory Write : A transfer information from R1 into the memory word M selected by the address in AR

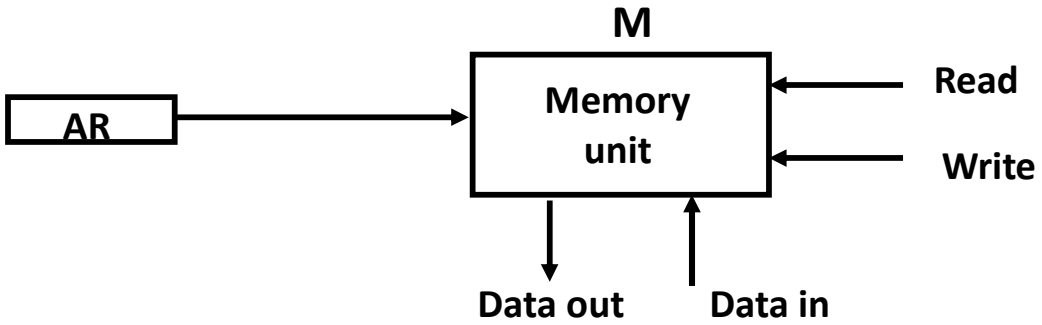
Memory - RAM

- **Memory (RAM) can be thought as a sequential circuits containing some number of registers**
- **Memory stores binary information in groups of bits called words**
- **These registers hold the *words* of memory**
- **Each of the r registers is indicated by an *address***
- **These addresses range from 0 to r-1**
- **Each register (word) can hold n bits of data**
- **Assume the RAM contains r = 2^k words. It needs the following**
  1. **n data input lines**
  2. **n data output lines**
  3. **k address lines**
  4. **A Read control line**
  5. **A Write control line**



Memory Transfer

Memory is usually accessed in computer systems by putting the desired address in a special register, the Memory Address Register (MAR, or AR)



Memory Read

- **To read a value from a location in memory and load it into a register, the register transfer language notation looks like this:**

$$R1 \leftarrow M[MAR]$$

- **This causes the following to occur**
  1. **The contents of the MAR get sent to the memory address lines**
  2. **A Read (= 1) gets sent to the memory unit**
  3. **The contents of the specified address are put on the memory's output data lines**
  4. **These get sent over the bus to be loaded into register R1**

Memory Write

- **To write a value from a register to a location in memory looks like this in register transfer language:**

$$M[MAR] \leftarrow R1$$

- **This causes the following to occur**
  1. **The contents of the MAR get sent to the memory address lines**
  2. **A Write (= 1) gets sent to the memory unit**
  3. **The values in register R1 get sent over the bus to the data input lines of the memory**
  4. **The values get loaded into the specified address in the memory**

SUMMARY OF R. TRANSFER MICROOPERATIONS

A ← B	1.Transfer content of reg. B into reg. A
A ← constant	3.Transfer a binary constant into reg. A
ABUS ← R1, R2 ← ABUS	4.Transfer content of R1 into bus A and, at the same time, transfer content of bus A into R2
AR	5.Address register
DR	6.Data register
M[R]	7.Memory word specified by reg. R
M	8.Equivalent to M[AR]
DR ← M	9.Memory *read* operation: transfers content of memory word specified by AR into DR
M ← DR	10.Memory *write* operation: transfers content of DR into memory word specified by AR

MICROOPERATIONS

Computer system microoperations are of four types:

- **Register transfer microoperations**
- **Arithmetic microoperations**
- **Logic microoperations**
- **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. …**

Summary of Typical Arithmetic Micro-Operations

R3 ← R1 + R2	Contents of R1 plus R2 transferred to R3
R3 ← R1 - R2	Contents of R1 minus R2 transferred to R3
R2 ← R2’	Complement the contents of R2
R2 ← R2’+ 1	2's complement the contents of R2 (negate)
R3 ← R1 + R2’+ 1	subtraction
R1 ← R1 + 1	Increment
R1 ← R1 - 1	Decrement

Overview

- Register Transfer Language
- Register Transfer
- Bus and Memory Transfers
- **Arithmetic Micro-operations**
- Logic Micro-operations
- Shift Micro-operations
- Arithmetic Logic Shift Unit

MICROOPERATIONS

Computer system microoperations are of four types:

- **Register transfer microoperations**
- **Arithmetic microoperations**
- **Logic microoperations**
- **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. …**

Summary of Typical Arithmetic Micro-Operations

R3 ← R1 + R2	Contents of R1 plus R2 transferred to R3
R3 ← R1 - R2	Contents of R1 minus R2 transferred to R3
R2 ← R2’	Complement the contents of R2
R2 ← R2’+ 1	2's complement the contents of R2 (negate)
R3 ← R1 + R2’+ 1	subtraction
R1 ← R1 + 1	Increment
R1 ← R1 - 1	Decrement

Binary Adder

- 4-bit Binary Adder : ***Fig. 4-6***
  - Full adder = 2-bits sum + previous carry
  - Binary adder = the arithmetic sum of two binary numbers of any length
  - c_0(input carry), c_4(output carry)

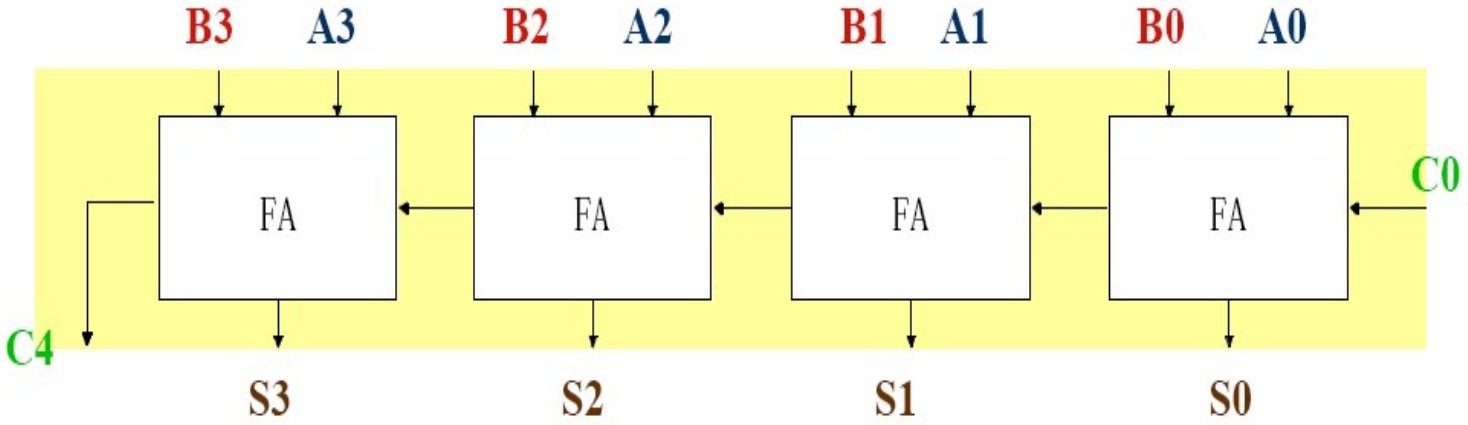
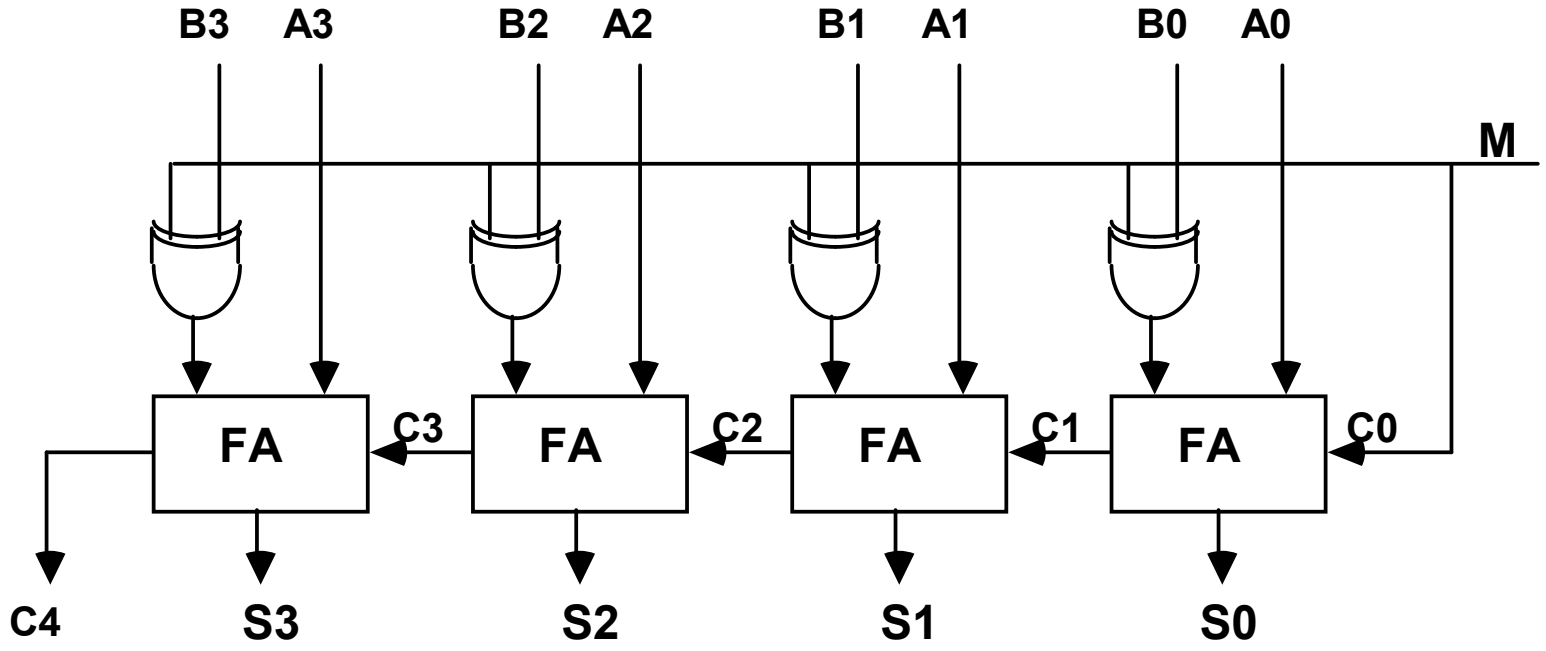


Figure 4-6. 4-bit binary adder

Binary Adder-Subtractor

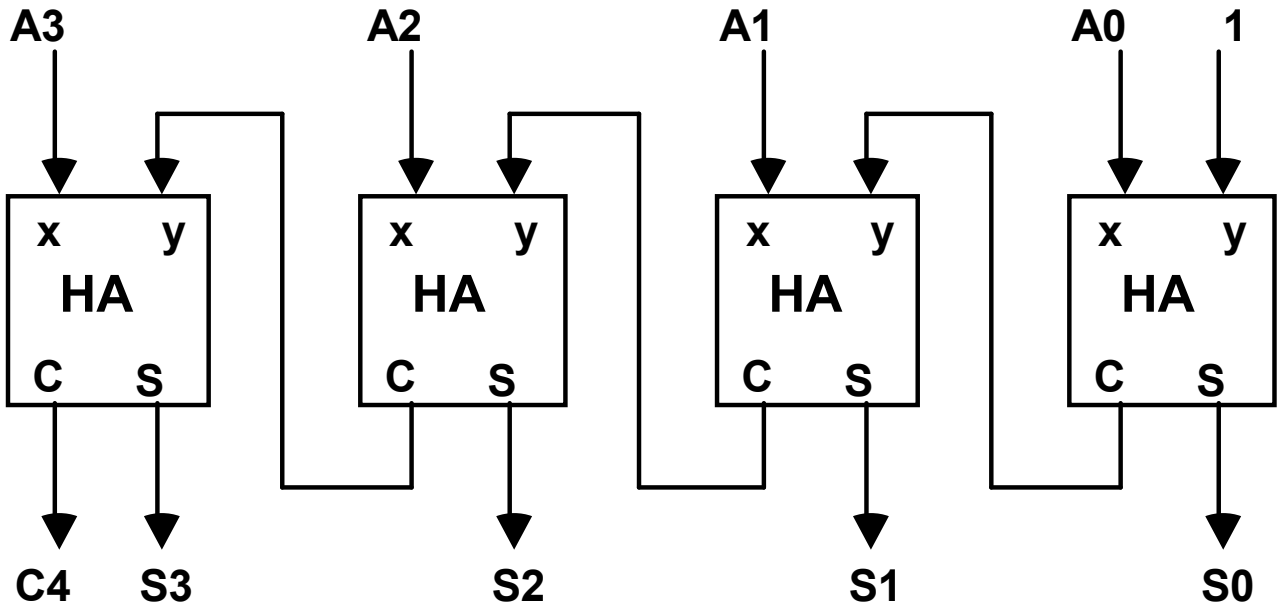
Binary Adder-Subtractor



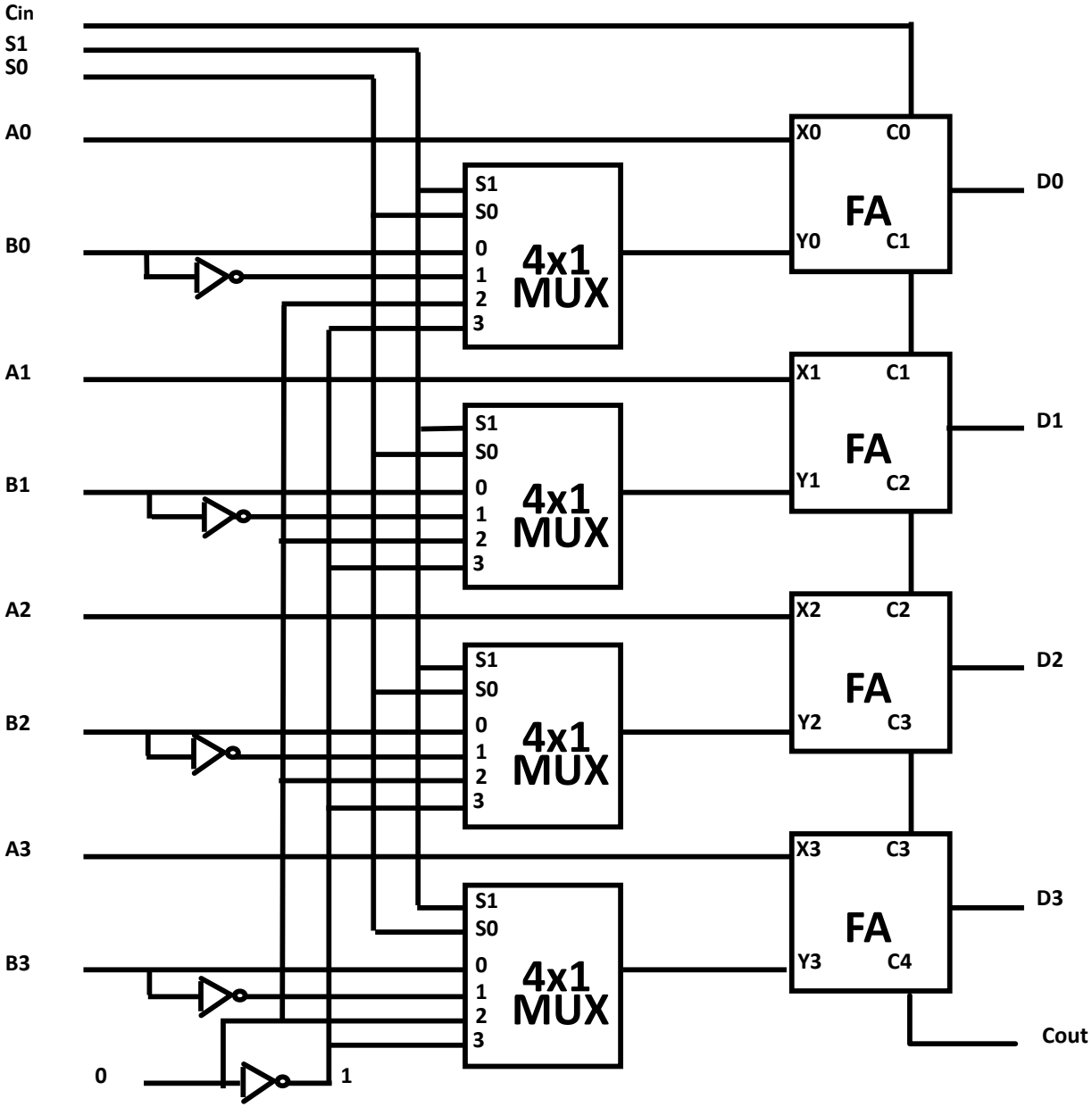
- Mode input M controls the operation
  - M=0 ---- adder
  - M=1 ---- subtractor

Binary Incrementer

Binary Incrementer



Arithmetic Circuits



Select			Input	Output
S1	S0	C_{in}	Y	$D=A+Y+C_{in}$
0	0	0	B	D=A+B
0	0	1	B	D=A+B+1
0	1	0	B'	D=A+B'
0	1	1	B'	D=A+B'+1
1	0	0	0	D=A
1	0	1	0	D=A+1
1	1	0	1	D=A-1
1	1	1	1	D=A

Overview

- Register Transfer Language
- Register Transfer
- Bus and Memory Transfers
- Arithmetic Micro-operations
- **Logic Micro-operations**
- **Shift Micro-operations**
- **Arithmetic Logic Shift Unit**

Logic Micro operations

- Logic microoperation
  - Logic microoperations consider ***each bit of the register separately*** and treat them as binary variables
    - » *exam)*

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

```
  1010  Content of R1
+ 1100  Content of R2
  0110  Content of R1 after P=1
```

  - Special Symbols
    - » Special symbols will be adopted for the logic microoperations ***OR(∨), AND(∧),*** and ***complement(a bar on top)***, to distinguish them from the corresponding symbols used to express Boolean functions
    - » *exam)*

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

Logic OR

Arithmetic ADD

- List of Logic Microoperation
  - Truth Table for 16 functions for 2 variables : ***Tab. 4-5***
  - 16 Logic Microoperation : ***Tab. 4-6***
- Hardware Implementation
  - 16 microoperation → Use only 4(AND, OR, XOR, Complement)
  - One stage of logic circuit

∵All other Operation can be derived

Logic Microoperations

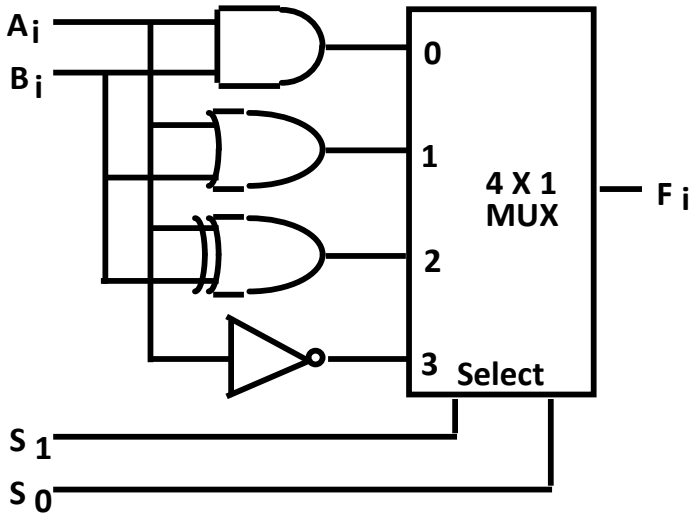
X	Y	F_0	F_1	F_2	F_3	F_4	F_5	F_6	F_7	F_8	F_9	F_{10}	F_{11}	F_{12}	F_{13}	F_{14}	F_{15}
0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
0	1	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
1	0	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
1	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1

TABLE 4-5. Truth Table for 16 Functions of Two Variables

Boolean function	Microoperation	Name
$F_0 = 0$	$F \leftarrow 0$	Clear
$F_1 = xy$	$F \leftarrow A \wedge B$	AND
$F_2 = xy'$	$F \leftarrow A \wedge \overline{B}$	
$F_3 = x$	$F \leftarrow A$	Transfer A
$F_4 = x'y$	$F \leftarrow \overline{A} \wedge B$	
$F_5 = y$	$F \leftarrow B$	Transfer B
$F_6 = x \oplus y$	$F \leftarrow A \oplus B$	Ex-OR
$F_7 = x+y$	$F \leftarrow A \vee B$	OR
$F_8 = (x+y)'$	$F \leftarrow \overline{A \vee B}$	NOR
$F_9 = (x \oplus y)'$	$F \leftarrow \overline{A \oplus B}$	Ex-NOR
$F_{10} = y'$	$F \leftarrow \overline{B}$	Compl-B
$F_{11} = x+y'$	$F \leftarrow A \vee \overline{B}$	
$F_{12} = x'$	$F \leftarrow \overline{A}$	Compl-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

TABLE 4-6. Sixteen Logic Microoperations

Hardware Implementation



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

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	**$A \leftarrow A \oplus B$**

Applications of Logic Microoperations

1. In a selective set operation, the bit pattern in B is used to *set* certain bits in A

1 1 0 0 A_t
1 0 1 0 B
1 1 1 0 A_{t+1} $(A \leftarrow A + B)$

If a bit in B is set to 1, that same position in A gets set to 1, otherwise that bit in A keeps its previous value

2. In a selective complement operation, the bit pattern in B is used to *complement* certain bits in A

1 1 0 0 A_t
1 0 1 0 B

0 1 1 0 A_{t+1} $(A \leftarrow A \oplus B)$

If a bit in B is set to 1, that same position in A gets complemented from its original value, otherwise it is unchanged

Applications of Logic Microoperations

3. In a selective clear operation, the bit pattern in B is used to *clear* certain bits in A

1 1 0 0 A_t

1 0 1 0 B

0 1 0 0 A_{t+1} $(A \leftarrow A \cdot B')$

If a bit in B is set to 1, that same position in A gets set to 0, otherwise it is unchanged

4. In a mask operation, the bit pattern in B is used to *clear* certain bits in A

1 1 0 0 A_t

1 0 1 0 B

1 0 0 0 A_{t+1} $(A \leftarrow A \cdot B)$

If a bit in B is set to 0, that same position in A gets set to 0, otherwise it is unchanged

Applications of Logic Microoperations

5. In a clear operation, if the bits in the same position in A and B are the same, they are cleared in A, otherwise they are set in A

1 1 0 0 A_t

1 0 1 0 B

0 1 1 0 A_{t+1} $(A \leftarrow A \oplus B)$

Applications of Logic Microoperations

6. An insert operation is used to introduce a specific bit pattern into A register, leaving the other bit positions unchanged

This is done as

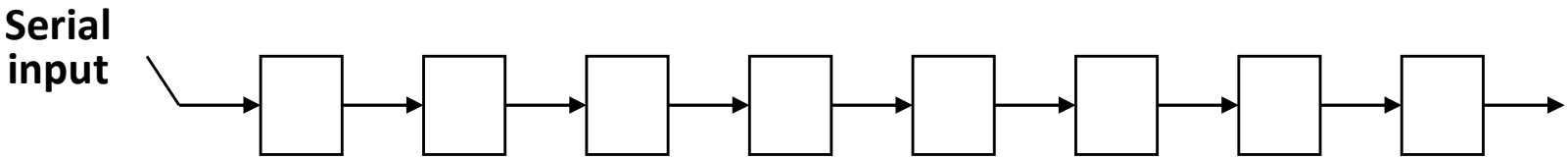
- **A mask operation to clear the desired bit positions, followed by**
- **An OR operation to introduce the new bits into the desired positions**
- **Example**
  - **Suppose you wanted to introduce 1010 into the low order four bits of A:**
  - **1101 1000 1011 0001 A (Original)**
    1101 1000 1011 1010 A (Desired)

  - `1101 1000 1011 0001  A (Original)`
    `1111 1111 1111 0000  Mask`
    `1101 1000 1011 0000  A (Intermediate)`
    `0000 0000 0000 1010  Added bits`
    `1101 1000 1011 1010  A (Desired)`

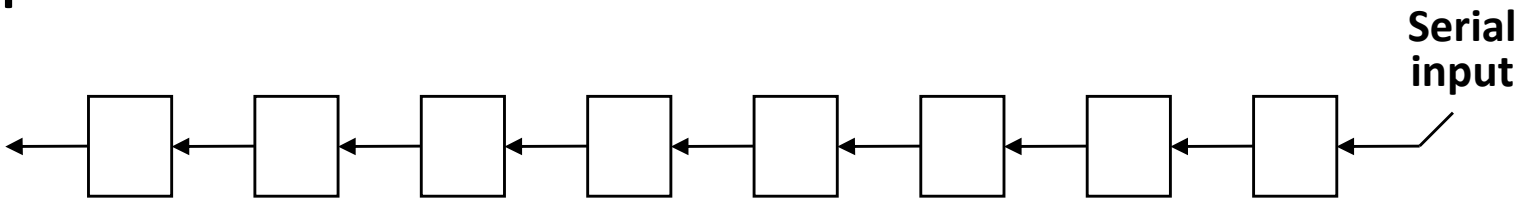
Shift Microoperations

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

- **A right shift operation**

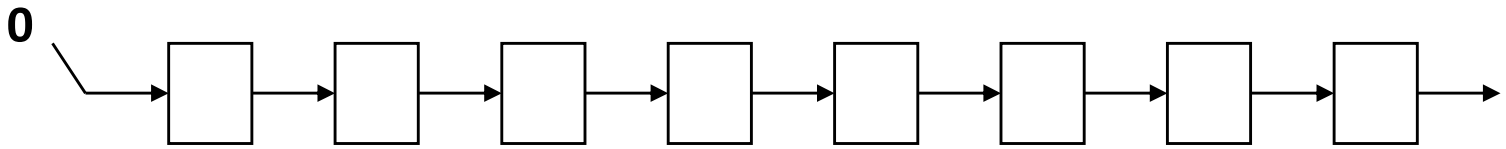


- **A left shift operation**

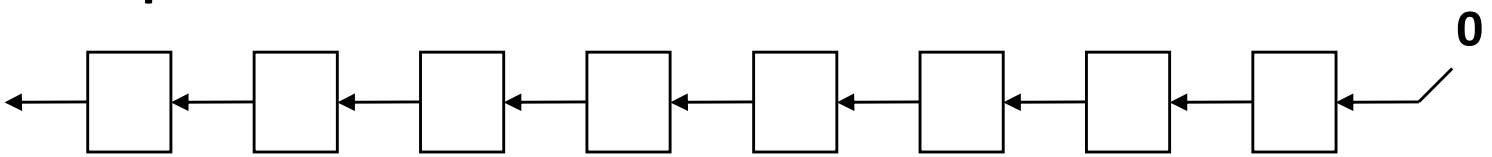


Logical Shift

- **In a logical shift the serial input to the shift is a 0.**
- **A right logical shift operation:**



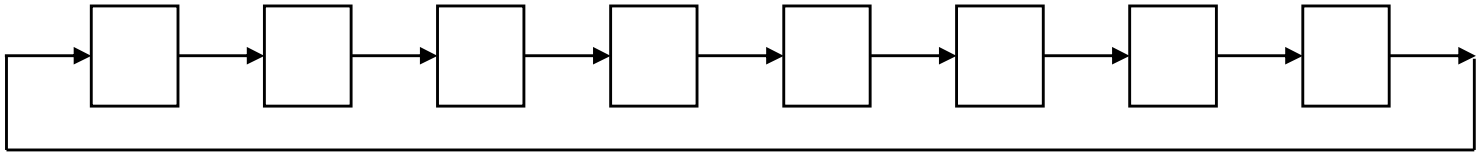
- **A left logical shift operation:**



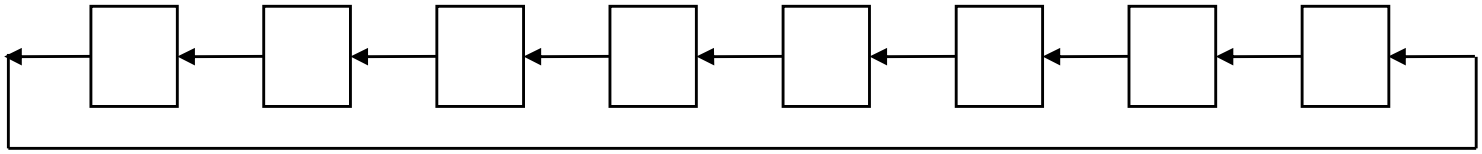
- **In a Register Transfer Language, the following notation is used**
  - ***shl*** **for a logical shift left**
  - ***shr*** **for a logical shift right**
  - **Examples:**
    - **R2 ← *shr* R2**
    - **R3 ← *shl* R3**

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:**

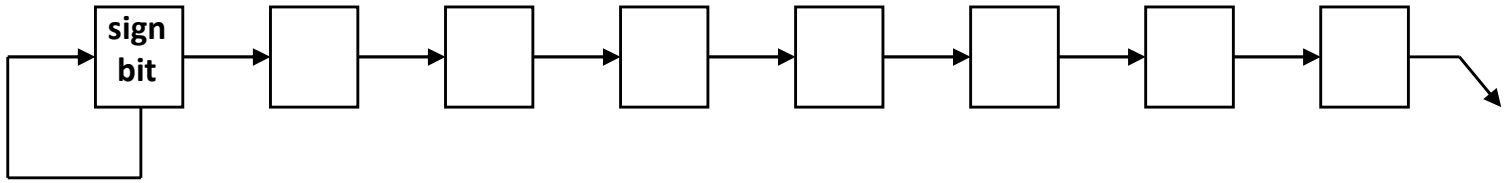


- **In a RTL, the following notation is used**
  - ***cil*** **for a circular shift left**
  - ***cir*** **for a circular shift right**
  - **Examples:**
    - **R2 ← *cir* R2**
    - **R3 ← *cil* R3**

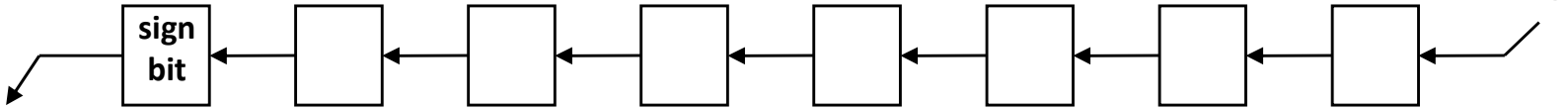
Arithmetic Shift

- **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**
- **Sign bit : 0 for positive and 1 for negative**
- **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:**

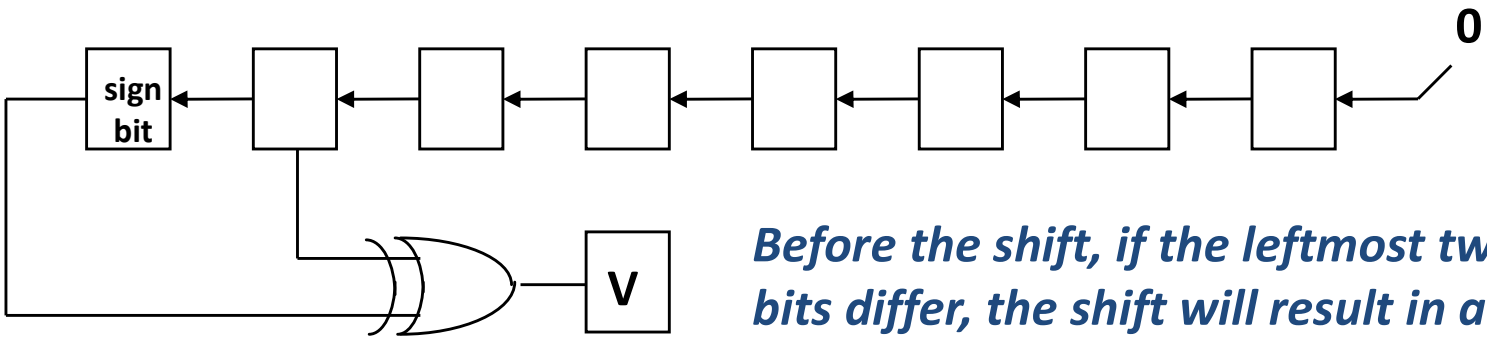


- **A left arithmetic shift operation:**



Arithmetic Shift

- **An left arithmetic shift operation must be checked for the overflow**

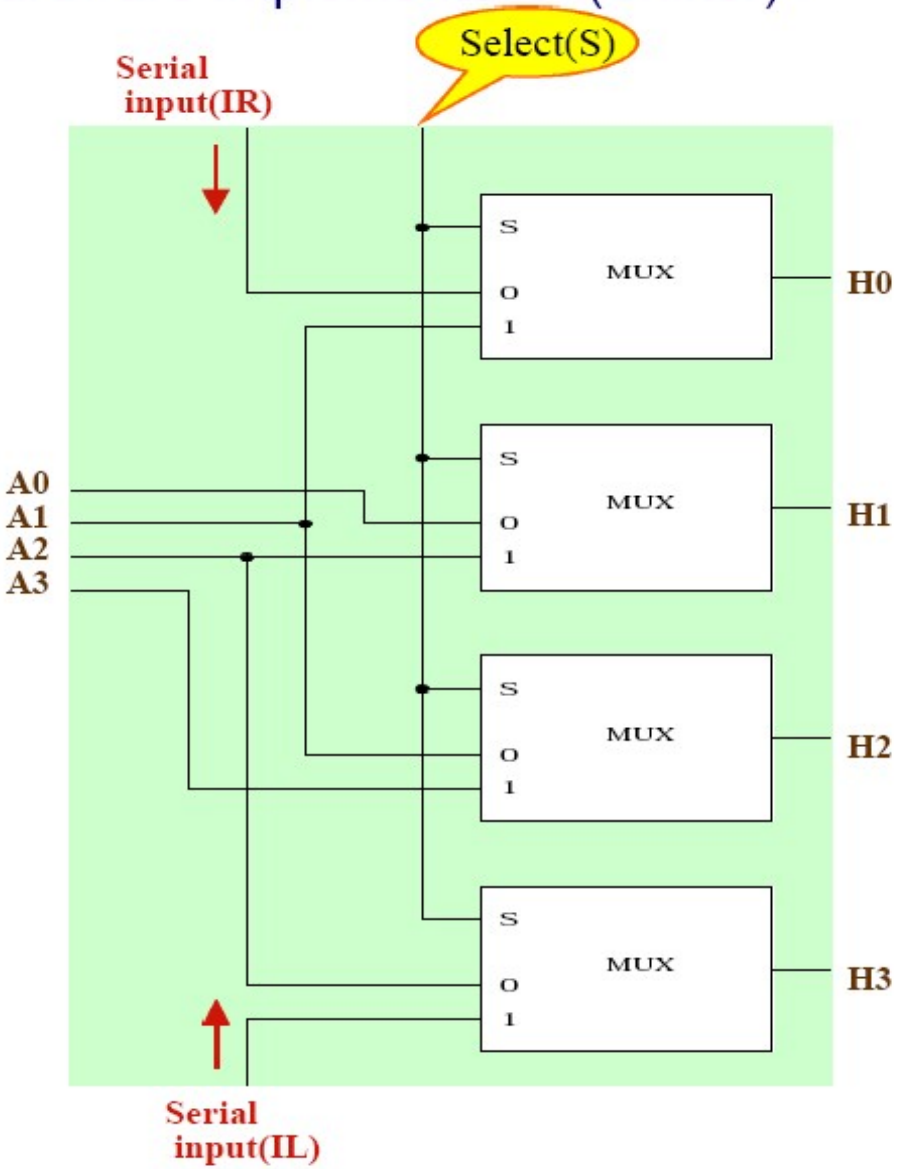


Before the shift, if the leftmost two bits differ, the shift will result in an overflow

- **In a RTL, the following notation is used**
  - ***ashl*** **for an arithmetic shift left**
  - ***ashr*** **for an arithmetic shift right**
  - **Examples:**
    - » **R2 ← *ashr* R2**
    - » **R3 ← *ashl* R3**

Hardware Implementation of Shift Microoperation

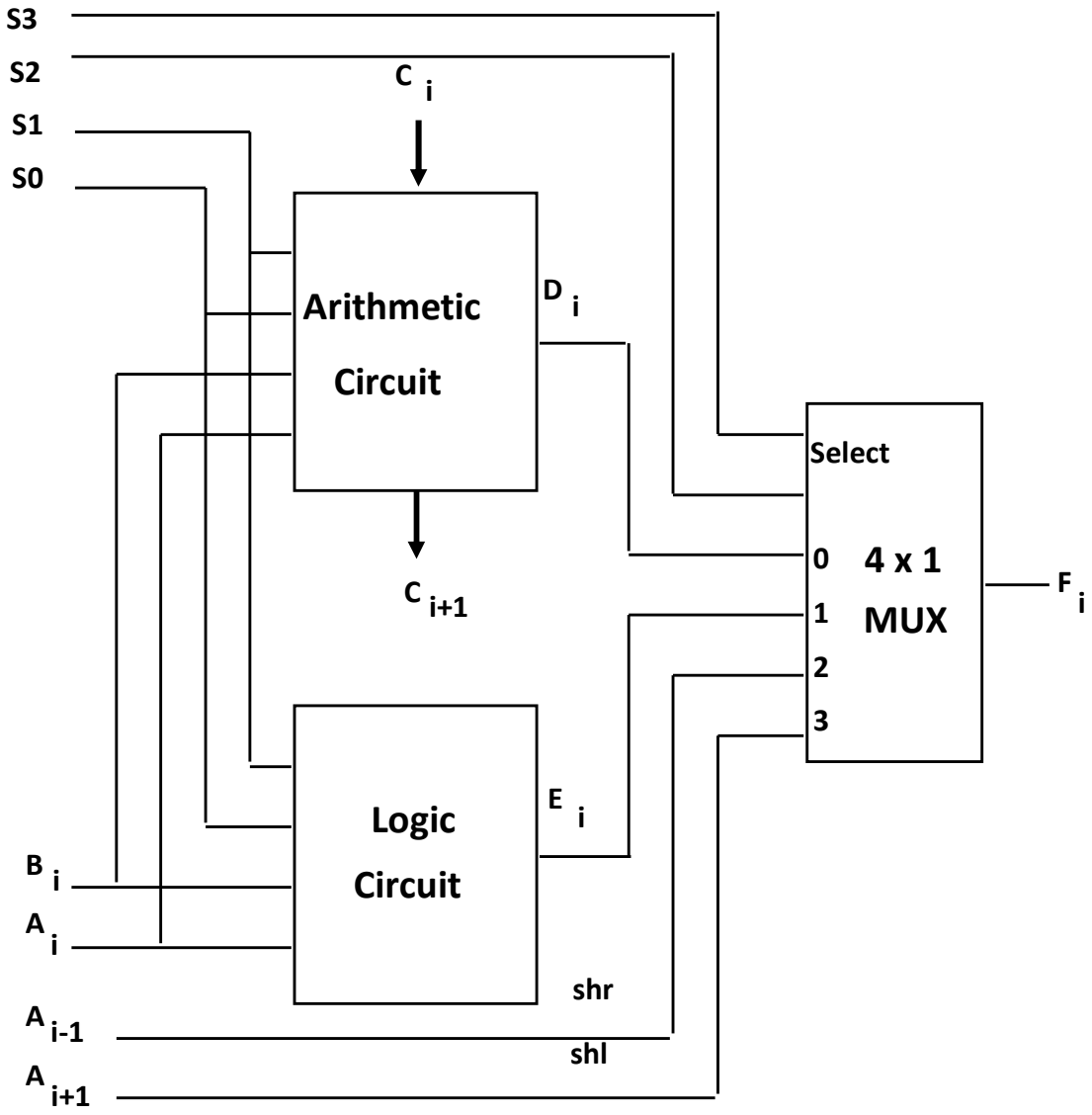
- Hardware Implementation(Shifter) :



Function Table

Select	output			
S	H0	H1	H2	H3
0	IR	A0	A1	A2
1	A1	A2	A3	IL

Arithmetic Logic and Shift Unit



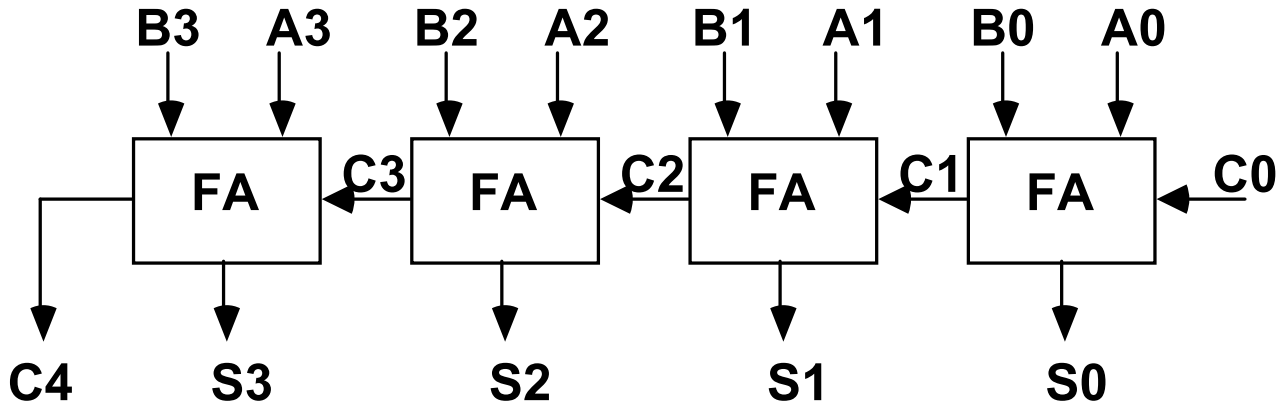
s2	s3	Operation
0	0	Arithmetic
0	1	Logical
1	0	Shr
1	1	shl

Overview

- Register Transfer Language
- Register Transfer
- Bus and Memory Transfers
- **Arithmetic Micro-operations**
- Logic Micro-operations
- Shift Micro-operations
- Arithmetic Logic Shift Unit

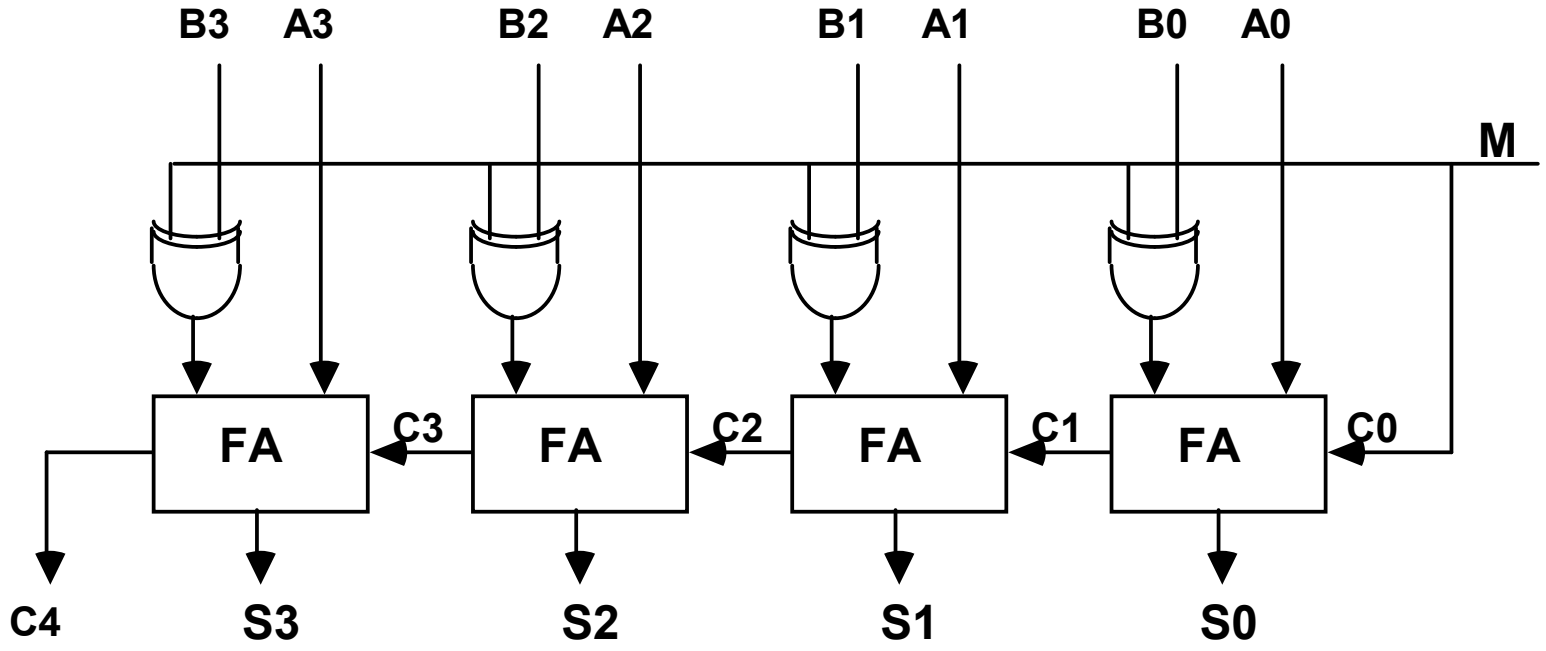
Binary Adder

Binary Adder



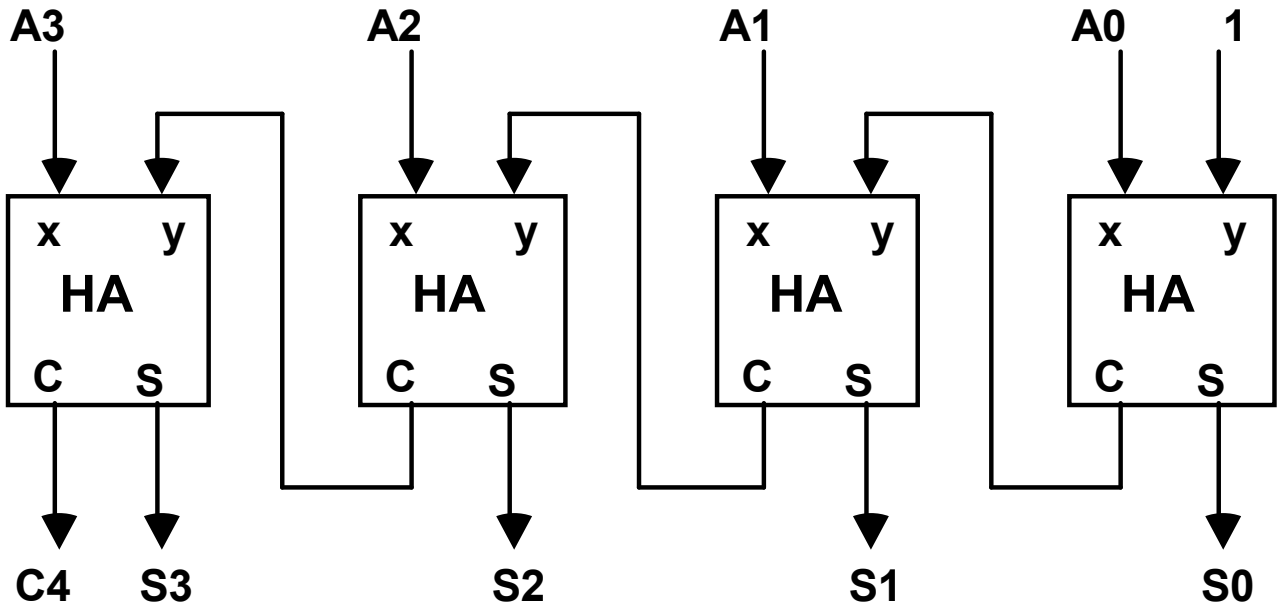
Binary Adder-Subtractor

Binary Adder-Subtractor



Binary Incrementer

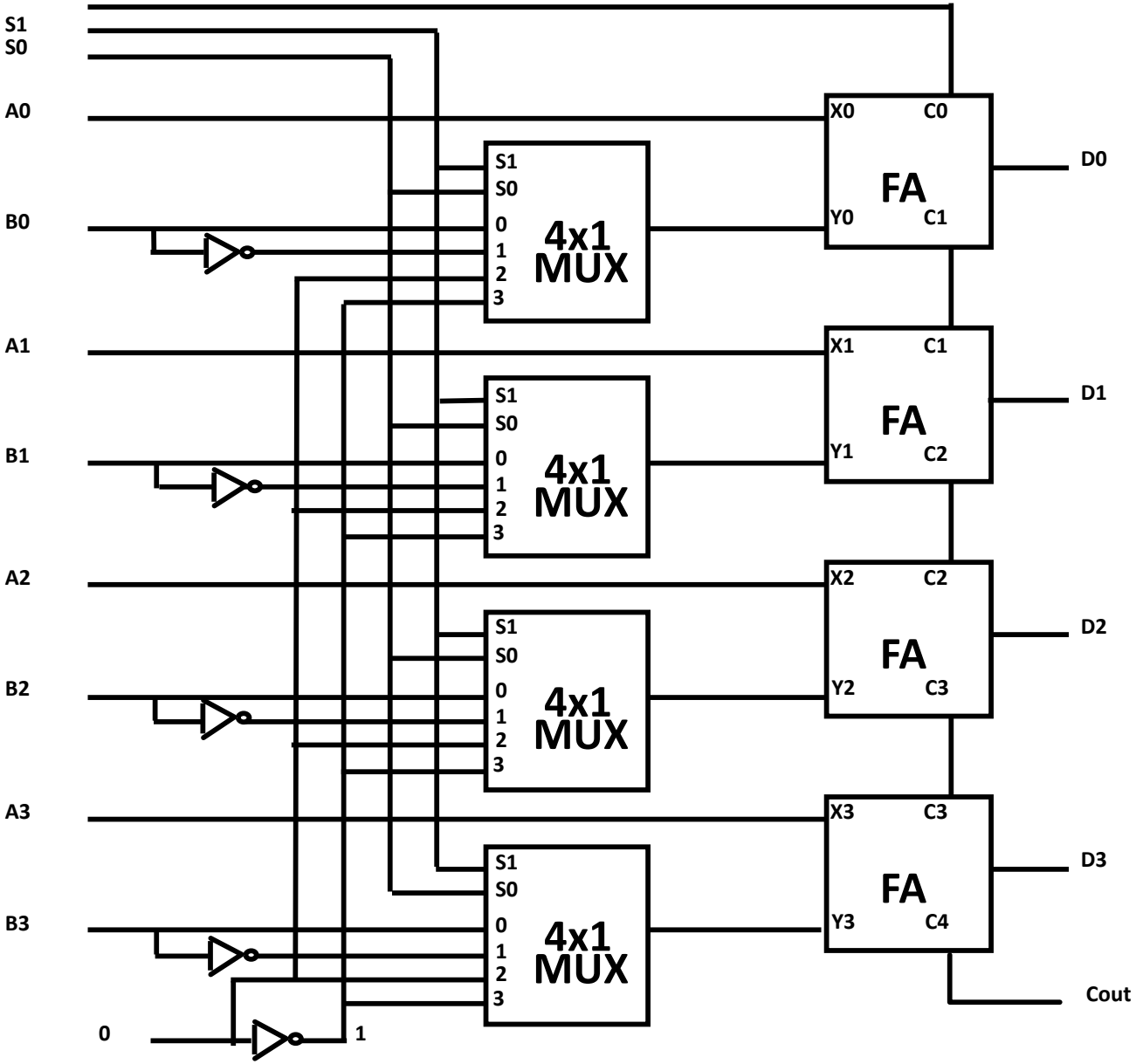
Binary Incrementer



Overview

- Register Transfer Language
- Register Transfer
- Bus and Memory Transfers
- **Arithmetic Micro-operations**
- **Logic Micro-operations**
- Shift Micro-operations
- Arithmetic Logic Shift Unit

Arithmetic Circuits



S1	S0	Cin	Y	O/P
0	0	0	B	D = A + B
0	0	1	B	D = A + B + 1
0	1	0	B’	D = A + B’
0	1	1	B’	D = A + B’+ 1
1	0	0	0	D = A
1	0	1	0	D = A + 1
1	1	0	1	D = A – 1
1	1	1	1	D = A

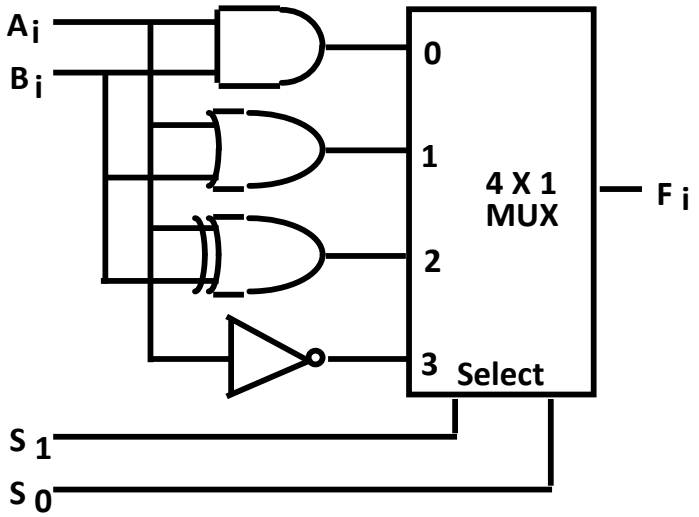
Logic Micro operations

- 16 different logic operations with 2 binary vars.
- n binary vars → 2^{2^n} functions

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

X y	0 0 1 1 0 1 0 1	*Boolean Function*	*Micro-Operations*	*Name*
	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)’	NAND
	1 1 1 1	F15 = 1	F ← all 1's	Set to all 1's

Hardware Implementation



Function table

S_1	S_0	Output	μ-operation
0	0	$F = A \wedge B$	AND
0	1	$F = A \vee B$	OR
1	0	$F = A \oplus B$	XOR
1	1	$F = A'$	Complement

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** $A \leftarrow A \cdot B'$
  - **Mask (Delete)** $A \leftarrow A \cdot B$
  - **Clear** $A \leftarrow A \oplus B$
  - **Insert** $A \leftarrow (A \cdot B) + C$
  - **Compare** $A \leftarrow A \oplus B$

Applications of Logic Microoperations

1. In a selective set operation, the bit pattern in B is used to *set* certain bits in A

1 1 0 0 A_t
1 0 1 0 B
1 1 1 0 A_{t+1} $(A \leftarrow A + B)$

If a bit in B is set to 1, that same position in A gets set to 1, otherwise that bit in A keeps its previous value

2. In a selective complement operation, the bit pattern in B is used to *complement* certain bits in A

1 1 0 0 A_t
1 0 1 0 B

0 1 1 0 A_{t+1} $(A \leftarrow A \oplus B)$

If a bit in B is set to 1, that same position in A gets complemented from its original value, otherwise it is unchanged

Applications of Logic Microoperations

3. In a selective clear operation, the bit pattern in B is used to *clear* certain bits in A

1 1 0 0 A_t

1 0 1 0 B

0 1 0 0 A_{t+1} $(A \leftarrow A \cdot B')$

If a bit in B is set to 1, that same position in A gets set to 0, otherwise it is unchanged

4. In a mask operation, the bit pattern in B is used to *clear* certain bits in A

1 1 0 0 A_t

1 0 1 0 B

1 0 0 0 A_{t+1} $(A \leftarrow A \cdot B)$

If a bit in B is set to 0, that same position in A gets set to 0, otherwise it is unchanged

Applications of Logic Microoperations

5. In a clear operation, if the bits in the same position in A and B are the same, they are cleared in A, otherwise they are set in A

1 1 0 0 A_t

1 0 1 0 B

0 1 1 0 A_{t+1} $(A \leftarrow A \oplus B)$

Applications of Logic Microoperations

6. An insert operation is used to introduce a specific bit pattern into A register, leaving the other bit positions unchanged

This is done as

- **A mask operation to clear the desired bit positions, followed by**
- **An OR operation to introduce the new bits into the desired positions**
- **Example**
  - **Suppose you wanted to introduce 1010 into the low order four bits of A:**
  - **1101 1000 1011 0001 A (Original)**
    1101 1000 1011 1010 A (Desired)

- `1101 1000 1011 0001  A (Original)`
  `1111 1111 1111 0000  Mask`
  `1101 1000 1011 0000  A (Intermediate)`
  `0000 0000 0000 1010  Added bits`
  `1101 1000 1011 1010  A (Desired)`

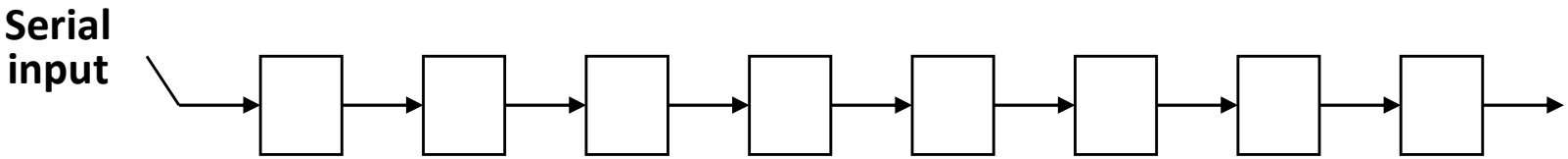
Overview

- Register Transfer Language
- Register Transfer
- Bus and Memory Transfers
- Arithmetic Micro-operations
- Logic Micro-operations
- **Shift Micro-operations**
- **Arithmetic Logic Shift Unit**

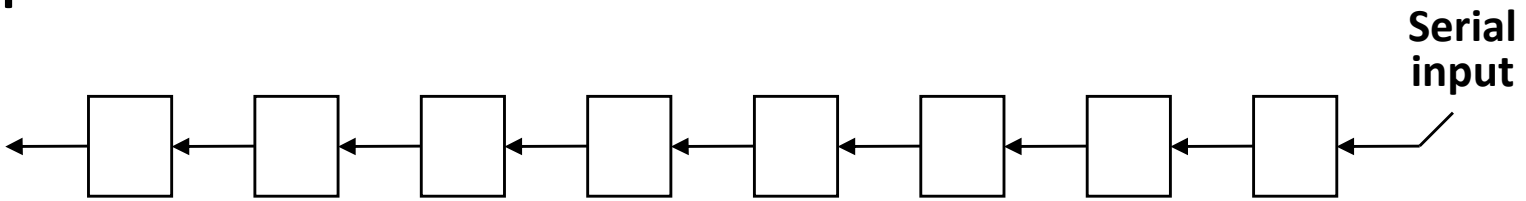
Shift Microoperations

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

- **A right shift operation**

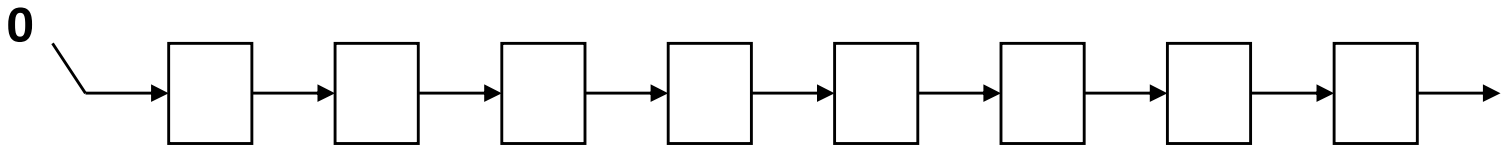


- **A left shift operation**

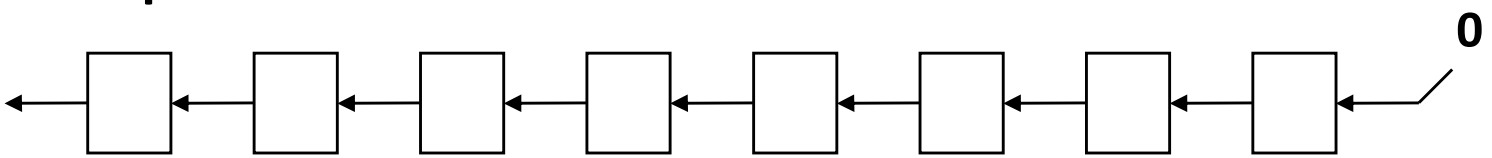


Logical Shift

- **In a logical shift the serial input to the shift is a 0.**
- **A right logical shift operation:**



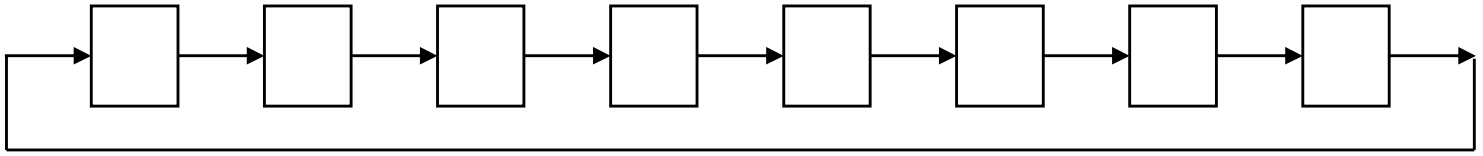
- **A left logical shift operation:**



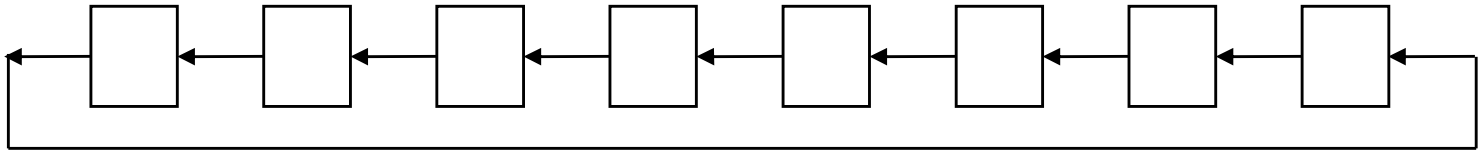
- **In a Register Transfer Language, the following notation is used**
  - ***shl*** **for a logical shift left**
  - ***shr*** **for a logical shift right**
  - **Examples:**
    - **R2 ← *shr* R2**
    - **R3 ← *shl* R3**

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:**

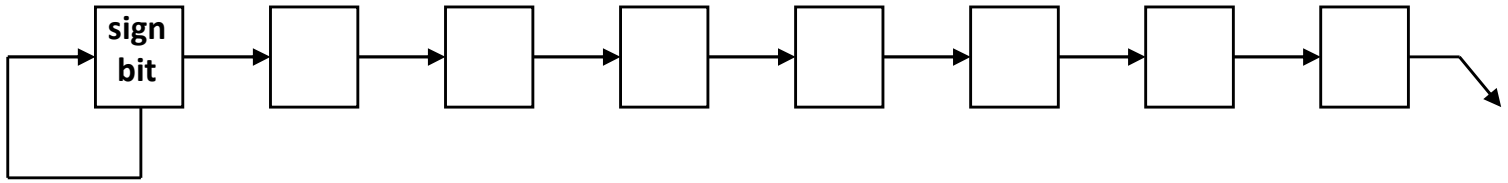


- **In a RTL, the following notation is used**
  - ***cil*** **for a circular shift left**
  - ***cir*** **for a circular shift right**
  - **Examples:**
    - **R2 ← *cir* R2**
    - **R3 ← *cil* R3**

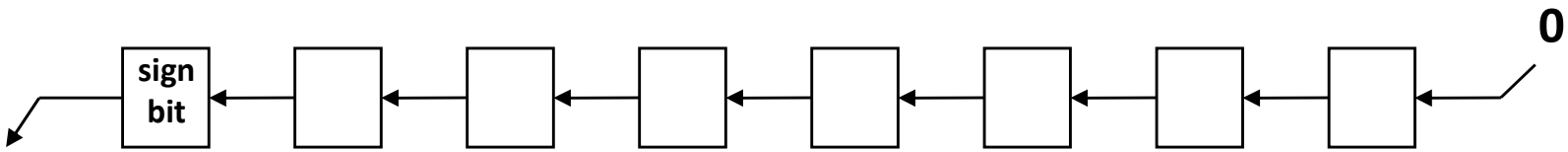
Arithmetic Shift

- **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:**

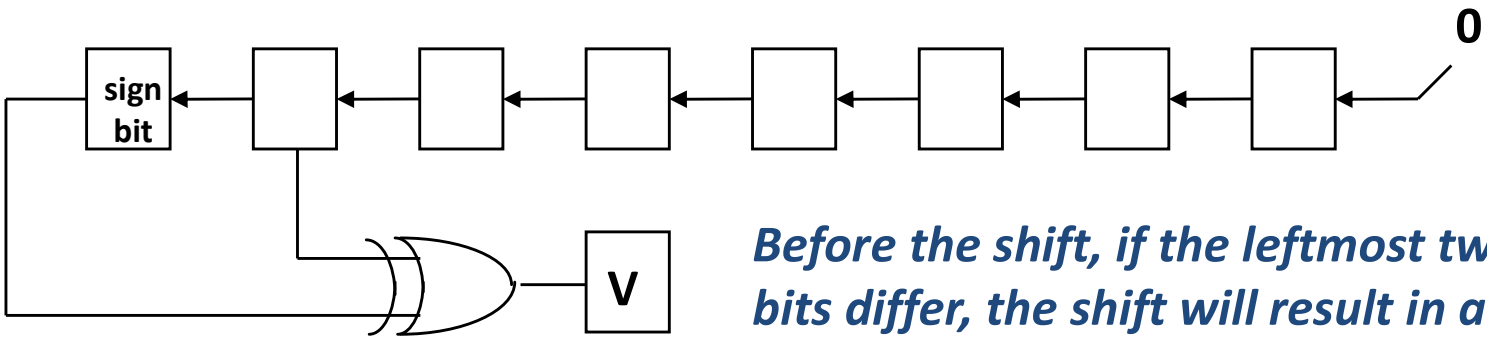


- **A left arithmetic shift operation:**



Arithmetic Shift

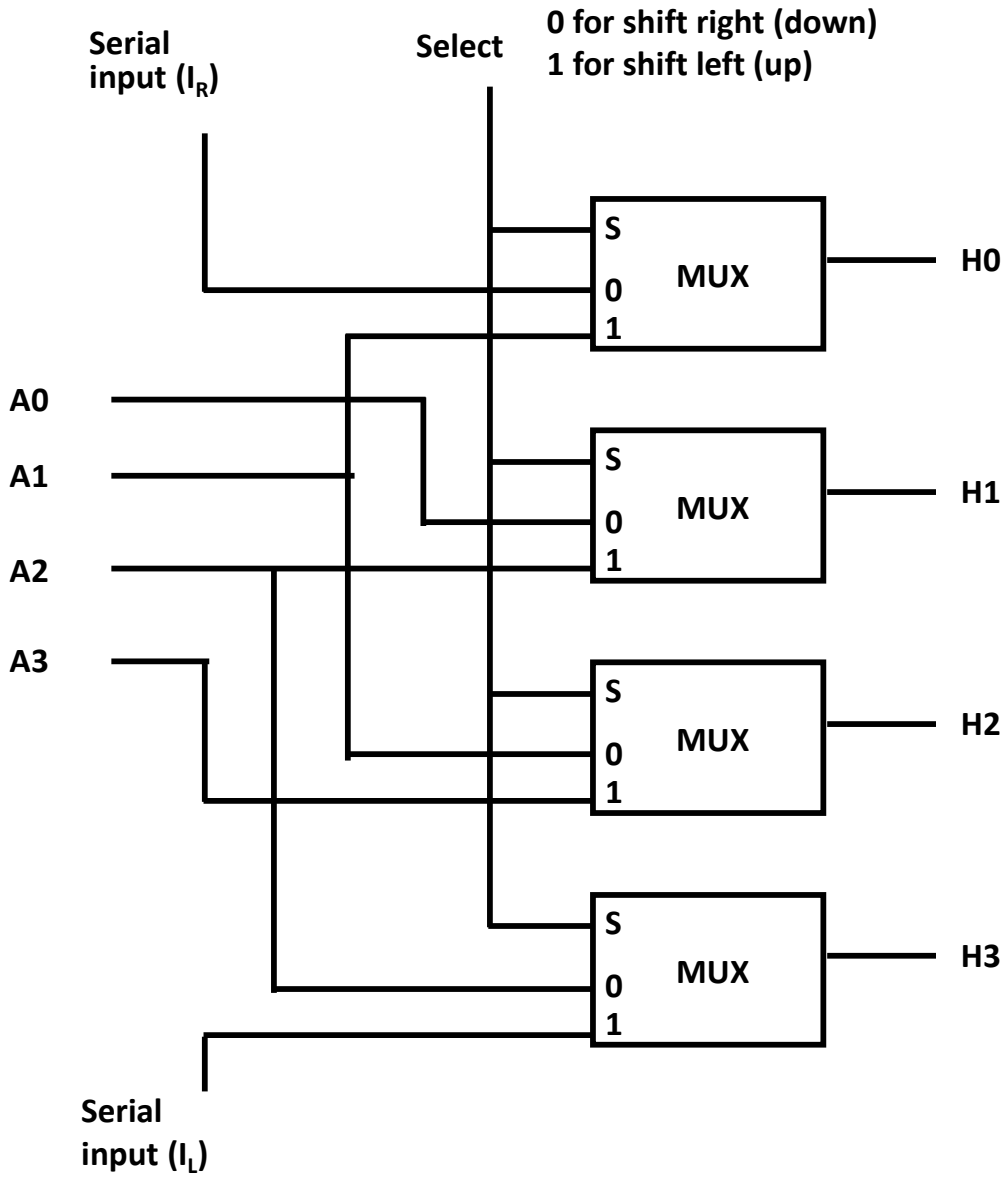
- **An left arithmetic shift operation must be checked for the overflow**



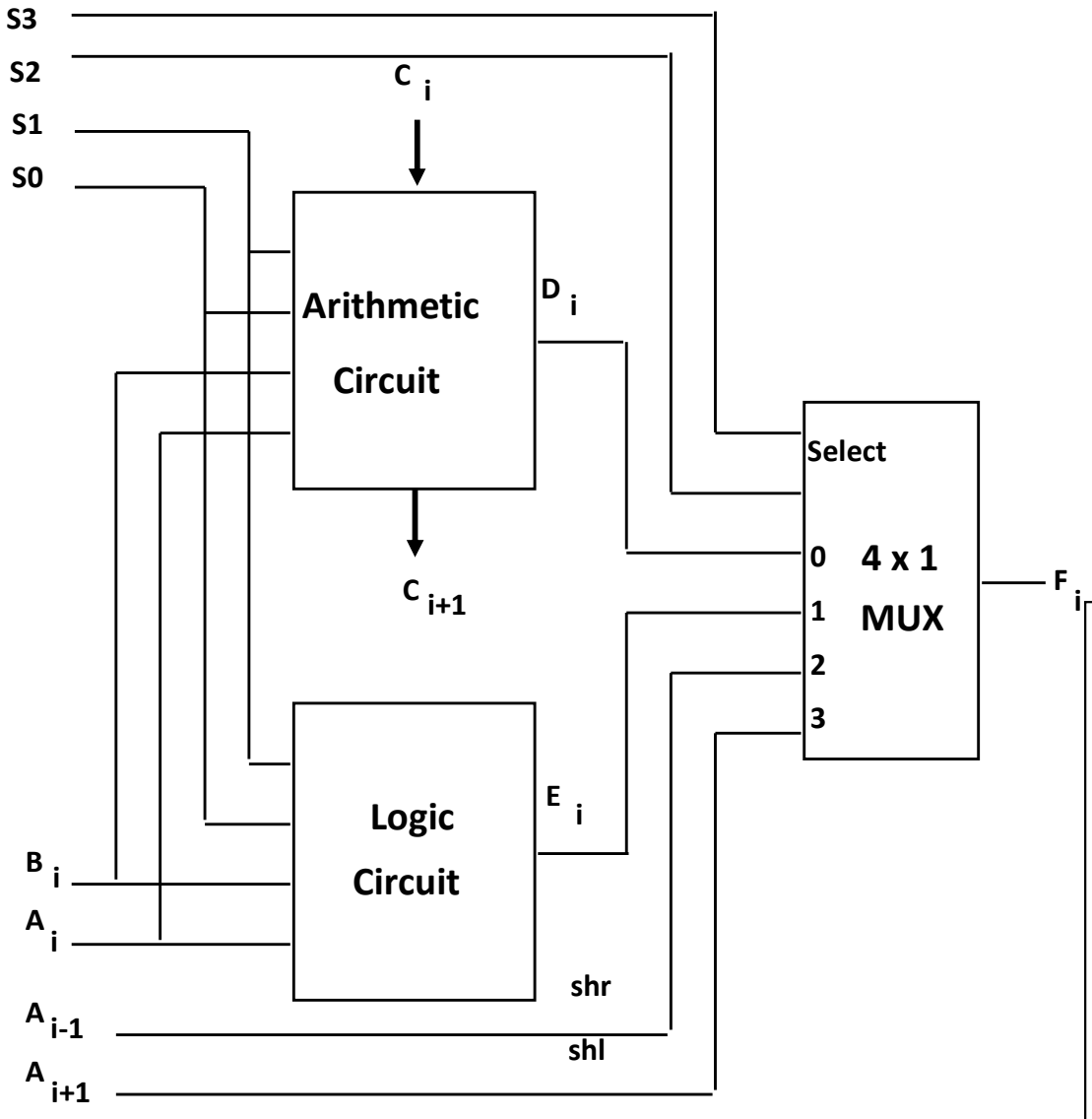
Before the shift, if the leftmost two bits differ, the shift will result in an overflow

- **In a RTL, the following notation is used**
  - ***ashl*** **for an arithmetic shift left**
  - ***ashr*** **for an arithmetic shift right**
  - **Examples:**
    - » **R2 ← *ashr* R2**
    - » **R3 ← *ashl* R3**

Hardware Implementation of Shift Microoperation



Arithmetic Logic and Shift Unit



S3	S2	S1	S0	Cin	Operation
0	0	0	0	0	F = A
0	0	0	0	1	F = A + 1
0	0	0	1	0	F = A + B
0	0	0	1	1	F = A + B + 1
0	0	1	0	0	F = A + B’
0	0	1	0	1	F = A + B’+ 1
0	0	1	1	0	F = A - 1
0	0	1	1	1	F = A
0	1	0	0	X	F = A $\wedge$ B
0	1	0	1	X	F = A $\vee$ B
0	1	1	0	X	F = A $\oplus$ B
0	1	1	1	X	F = A’
1	0	X	X	X	F = shr A
1	1	X	X	X	F = shl A