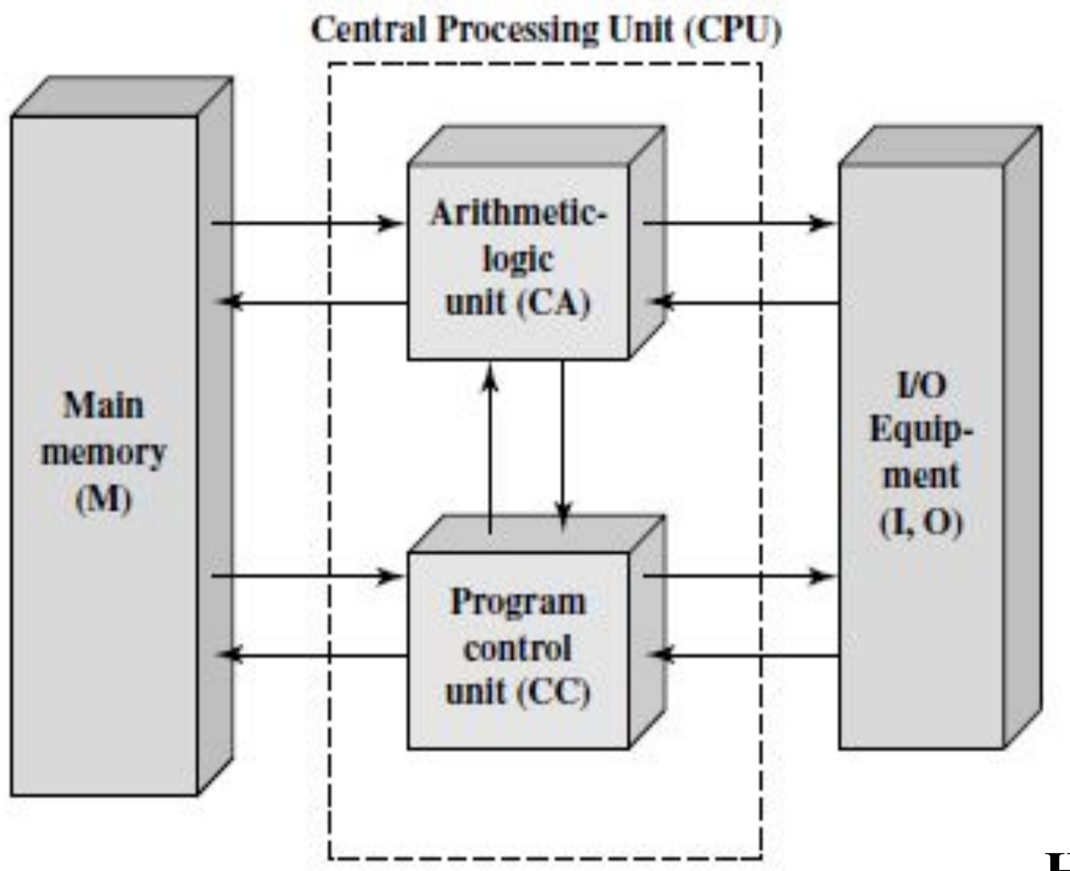
Computer Model (IAS-Institute for Advanced Study**)**


Central Processing Unit (CPU)
Main memory (M)
Arithmetic-logic unit (CA)
Program control unit (CC)
I/O Equip-ment (I, O)

Central Arithmetical (CA),
Central Control (CC),
Memory (M),
Input (I),
Output (O)

Von Neumann Stored Program Concept

A main memory, which stores both data and instructions

An arithmetic and logic unit (ALU) capable of operating on binary data

A control unit, which interprets the instructions in memory and causes them to be executed

Input and output (I/O) equipment operated by the control unit

The model for a class of computing machines designed by John von Neumann.

How Computer Works?

It basically works on stored program principle

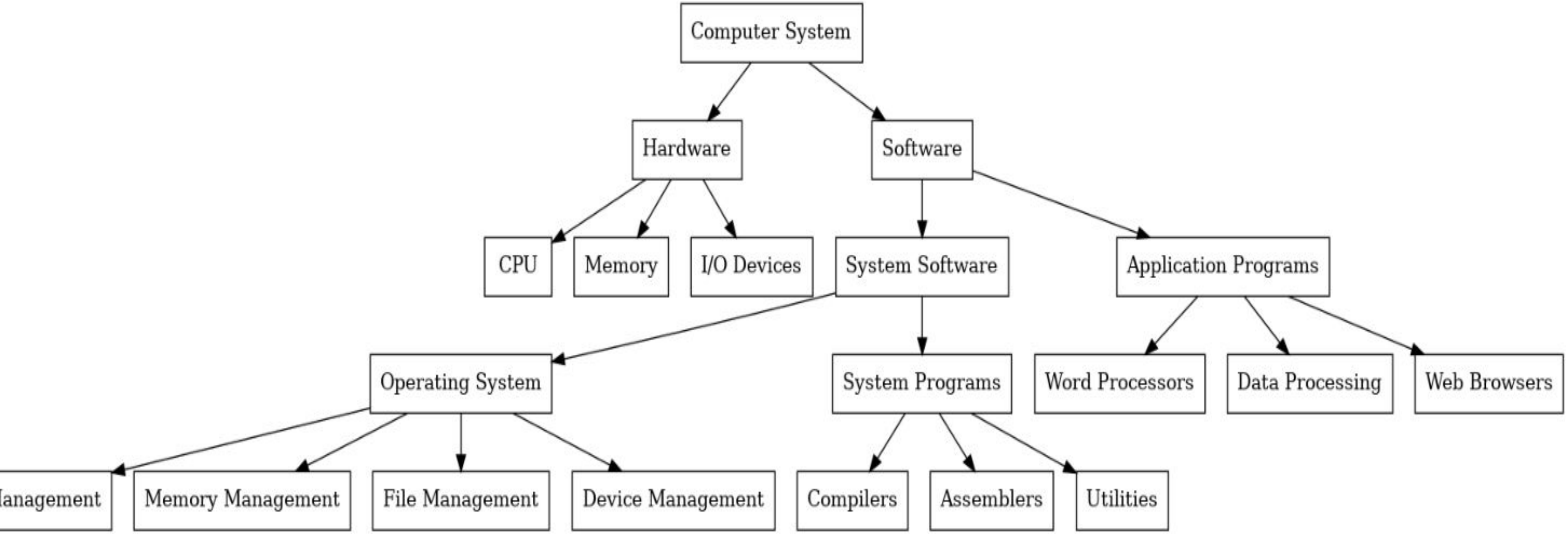
"Von Neumann Stored Program Concept"

Digital Computer

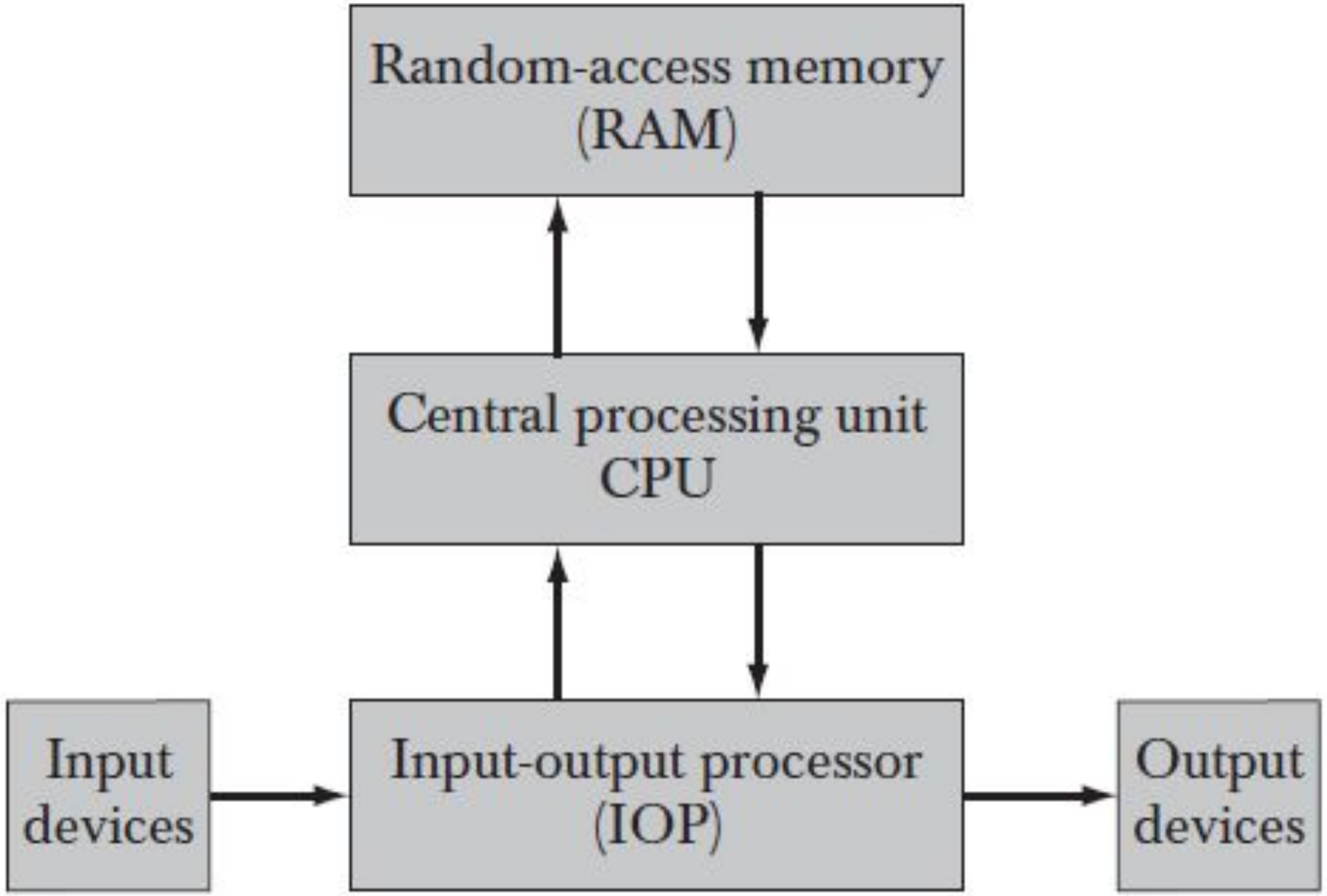
- The digital computer is a digital system that performs various computational tasks.
- Digital computers use the binary number system, which has two digits: 0 and 1.
- A binary digit is called a bit.
- Information is represented in digital computers in groups of bits.

A Computer System

- A computer system is sometimes subdivided into two functional entities:
    - **Hardware and Software**.
- The **hardware** of the computer consists of all the electronic components and electromechanical devices that comprise the physical entity of the device.
- Computer **software** consists of the **instructions** and **data** that the computer manipulates to perform various data-processing tasks.
- A sequence of instructions for the computer is called a **program**.
- The data that are manipulated by the program constitute the **data base.**


Computer System
Hardware
Software
CPU
Memory
I/O Devices
System Software
Application Programs
Operating System
System Programs
Word Processors
Data Processing
Web Browsers
lanagement
Memory Management
File Management
Device Management
Compilers
Assemblers
Utilities

Block diagram of a digital computer


Random-access memory (RAM)
Central processing unit CPU
Input devices
Input-output processor (IOP)
Output devices

Block diagram of a digital computer

- The central processing unit (CPU) contains an arithmetic and logic unit for manipulating data, a number of registers for storing data, and control circuits for fetching and executing instructions.
- The memory of a computer contains storage for instructions and data.
- It is called a random-access memory (RAM) because the CPU can access any location in memory at random and retrieve the binary information within a fixed interval of time.
- The input and output processor (IOP) contains electronic circuits for communicating and controlling the transfer of information between the computer and the outside world.
- The input and output devices connected to the computer include keyboards, printers, terminals, magnetic disk drives, and other communication devices.

Computer organization

- Computer organization is concerned with the way the hardware components operate and the way they are connected together to form the computer system.
- The various components are assumed to be in place and the task is to investigate the organizational structure to verify that the computer parts operate as intended.

Computer organization

- **Focus:** How components operate and interconnect.
- Concerned with the operational aspects of hardware components, such as their performance and interaction.
- **Objective:** Verify and ensure components function as intended.
- **Example Topics:**
    - Data paths and control signals
    - Timing and sequencing of operations
    - Implementation of control units.

Computer design

- Computer design is concerned with the hardware design of the computer.
- Once the computer specifications are formulated, it is the task of the designer to develop hardware for the system.
- Computer design is concerned with the determination of what hardware should be used and how the parts should be connected.
- This aspect of computer hardware is sometimes referred to as computer implementation.

Computer design

- **Focus:** Hardware design and implementation.
- Involves translating specifications into actual hardware.
- **Objective:** Determine the hardware requirements and their interconnection.
- **Example Topics:**
  - Circuit design (e.g., gates and flip-flops)
  - Logic design for processors, memory, and I/O devices
  - Hardware-software co-design.

Computer architecture

- Computer architecture is concerned with the structure and behaviour of the computer as seen by the user.
- It includes the information, formats, the instruction set, and techniques for addressing memory.
- The architectural design of a computer system is concerned with the specifications of the various functional modules, such as processors and memories, and structuring them together into a computer system.

Computer architecture

- **Focus:** Structure and behaviour from the user's perspective.
- Deals with the functional specifications and interface design of components.
- **Objective:** Define the instruction set, addressing techniques, and data formats.
- **Example Topics:**
  - Instruction set architecture (ISA).
  - Memory addressing schemes.
  - High-level system design (e.g., multi-core processor architecture).

Relationship Between These Aspects

- **Architecture** defines the overall structure and functionality.
- **Organization** ensures that the architecture is implementable and performs as required.
- **Design** focuses on building the system based on the organization and architectural specifications.

Classification of Computer Architecture

Classification of Computer Architecture

Broadly can be classified into two types

1. Von-Neumann Architecture
2. Harvard Architecture

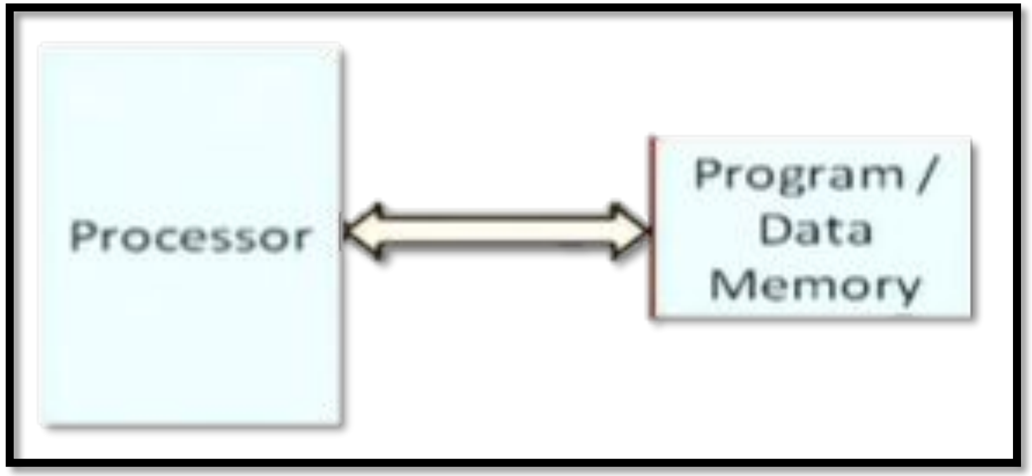
Von-Neumann Architecture

- Instructions and Data are stored in same memory module.
- More flexible and easier to implement.
- Suitable for most of the general-purpose processors.

General Disadvantage:

✔ The processor memory bus acts as bottleneck.

✔ All instructions and data are moved back and forth through the pipe.

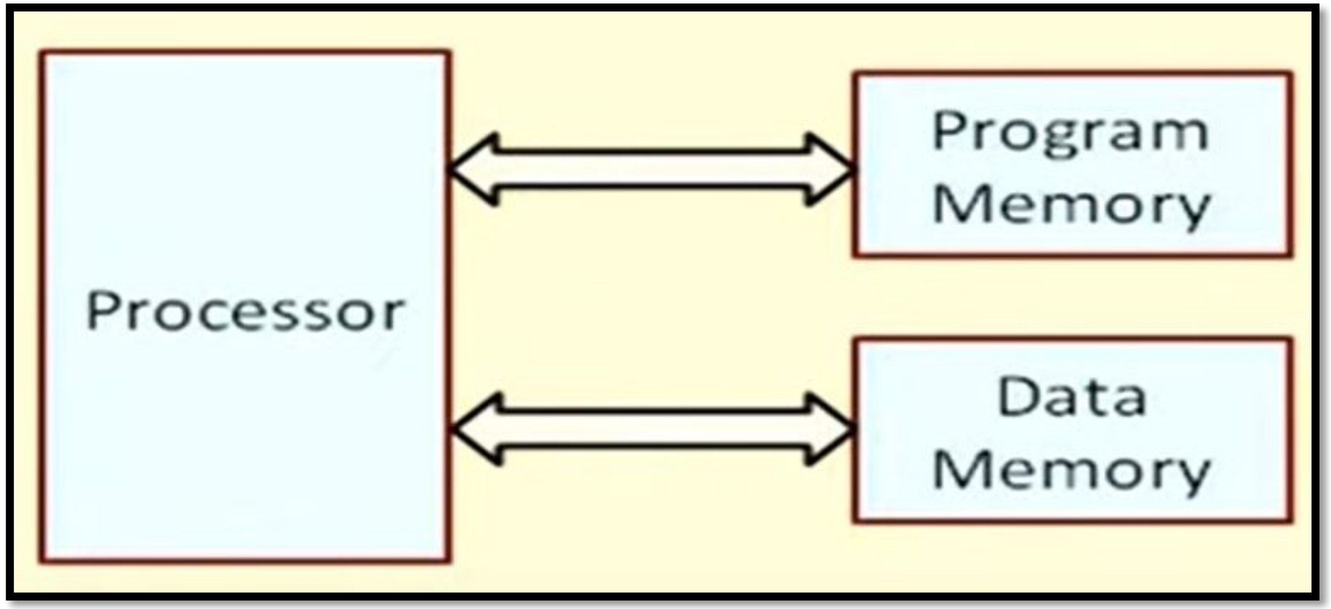


Harvard Architecture

- Separate memory for Program and Data

-Instructions are stored in Program Memory and Data are stored in Data Memory

- Instruction and Data Access can be done in Parallel
- The processor-memory bottleneck remains. (Multiple instructions and data cannot be accessed in same time).



Instruction Set Architecture (ISA)

Digital Computer System

- A digital system is an interconnection of **digital hardware modules** that **accomplish a specific information-processing task**.
- Digital systems **vary in size and complexity** from a few integrated circuits to a complex of interconnected and interacting digital computers.
- Digital system design invariably uses a **modular approach**.
- The modules are constructed from such digital components as registers, decoders, arithmetic elements, and control logic.
- The various modules are interconnected with common data and control paths to form a digital computer system.

Microoperation

- Digital modules are best defined by the registers they contain and the operations that are performed on the data stored in them.
- **“The operations executed on data stored in registers are called microoperations.”**
- A microoperation is an elementary operation performed on the information stored in one or more registers.
- The result of the operation may replace the previous binary information of a register or may be transferred to another register.
- **Examples of microoperations** are shift, count, clear, and load.

Basic Classification of Microoperations

The microoperations most often encountered in digital computers are classified into four categories:

1. **Register transfer microoperations** transfer binary information from one register to another.

2. **Arithmetic microoperations** perform arithmetic operation on numeric data stored in registers.

3. **Logic microoperations** perform bit manipulation operations on nonnumeric data stored in registers.

4. **Shift microoperations** perform shift operations on data stored in registers.

Master Clock Generator

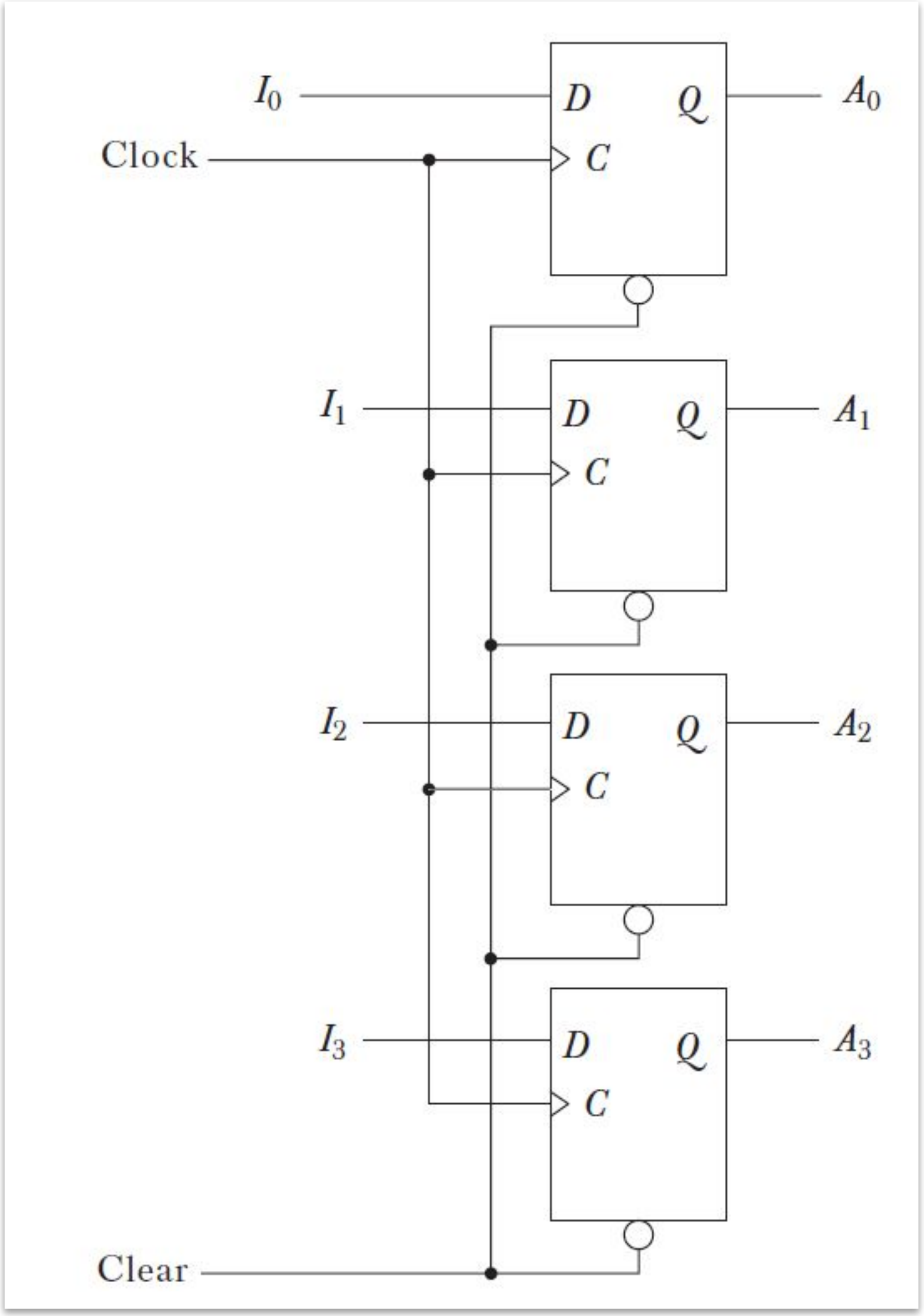
- Most digital systems have a master clock generator that supplies a continuous train of clock pulses.
- The clock pulses are applied to all flip-flops and registers in the system.
- The master clock acts like a pump that supplies a constant beat to all parts of the system.
- A separate control signal must be used to decide which specific clock pulse will have an effect on a particular register.

Registers

- A register consists of a group of flip-flops and gates that effect their transition.
- The flip-flops hold the binary information and the gates control when and how new information is transferred into the register.

4-bit register

When clear input goes to 0, all flip-flops are reset asynchronously



In this configuration, the clock must be inhibited from the circuit if the content of the register must be left unchanged.

Internal Hardware organization of a Digital Computer

The internal hardware organization of a digital computer is best defined by specifying:

1. The set of registers it contains and their function.
2. The sequence of microoperations performed on the binary information stored in the registers.
3. The control that initiates the sequence of microoperations.

Register Transfer Language

- The **symbolic notation** used to describe the microoperation transfers among registers is called a register transfer language.
- The term "register transfer" implies the **availability of hardware logic circuits that can perform a stated microoperation and transfer the result of the operation to the same or another register**.
- A register transfer language is a system for expressing in symbolic form the microoperation sequences among the registers of a digital module.
- It is a convenient tool for **describing the internal organization of digital computers in concise and precise manner**.
- It can specify the register transfers, the microoperations, and the control functions that describe the internal hardware organization of digital computers
- It can also be **used to facilitate the design process of digital systems**.

Register Transfer

Registers

- Computer registers are designated by capital letters (sometimes followed by numerals) to denote the function of the register.

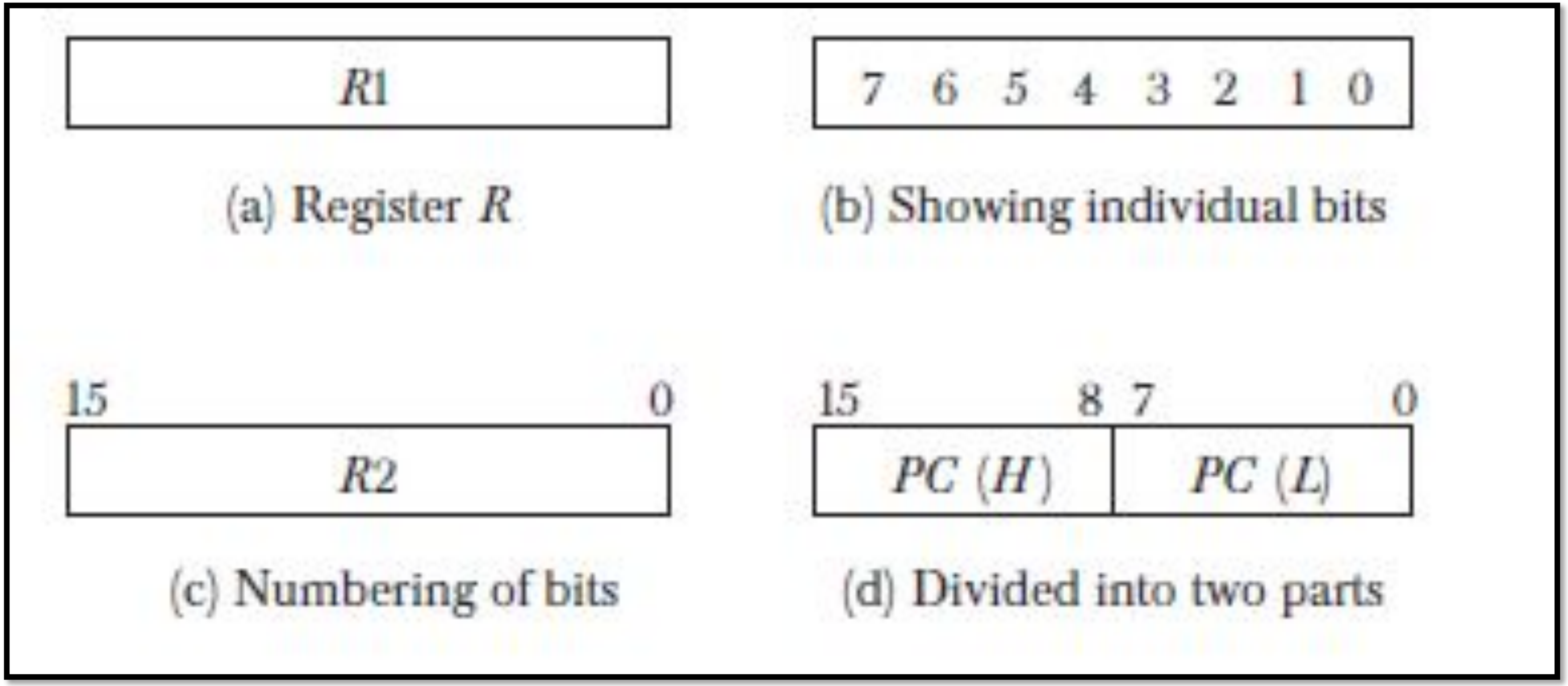
Example

- MAR- Memory Address Register

- The register that holds the address of memory unit is called MAR

- PC-Program Counter
- IR-Instruction Register
- R1-Process Register

Block Diagram of Register



Register Transfer

- Information transfer from one register to another is designated in symbolic form by means of a **replacement operator**

$$\boxed{R2 \leftarrow R1}$$

$R1$: Source register

$R2$: Destination register

- Denotes a transfer of the content of register $R1$ into register $R2$.
- It designates a **replacement of the content of $R2$ by the content of $R1$.**
- By definition, the content of the source register $R1$ does not change after the transfer.

NOTE:

A statement that specifies a register transfer implies that circuits are available from the outputs of the source register to the inputs of the destination register and that the **destination register has a parallel load capability**

Control Function

- Normally, we want the transfer to occur only under a predetermined control condition.
- This can be shown by means of an if-then statement.

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

where P is a control signal generated in the control section.

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

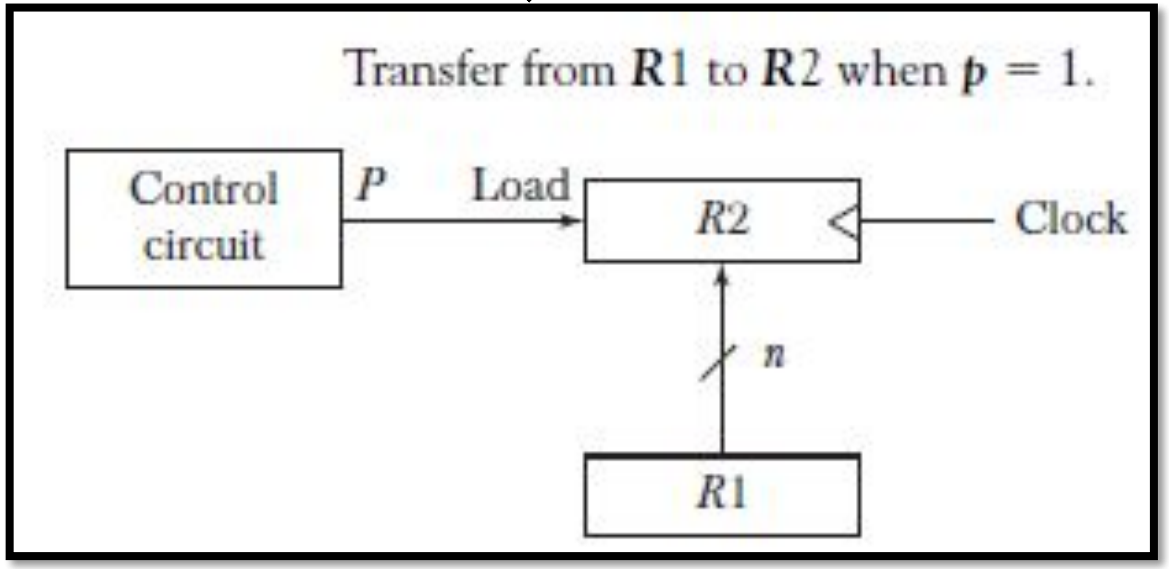
P : R2 $\leftarrow$ R1

the transfer operation be executed by the hardware only if P = 1.

Hardware for Transfer and Timing Diagram

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

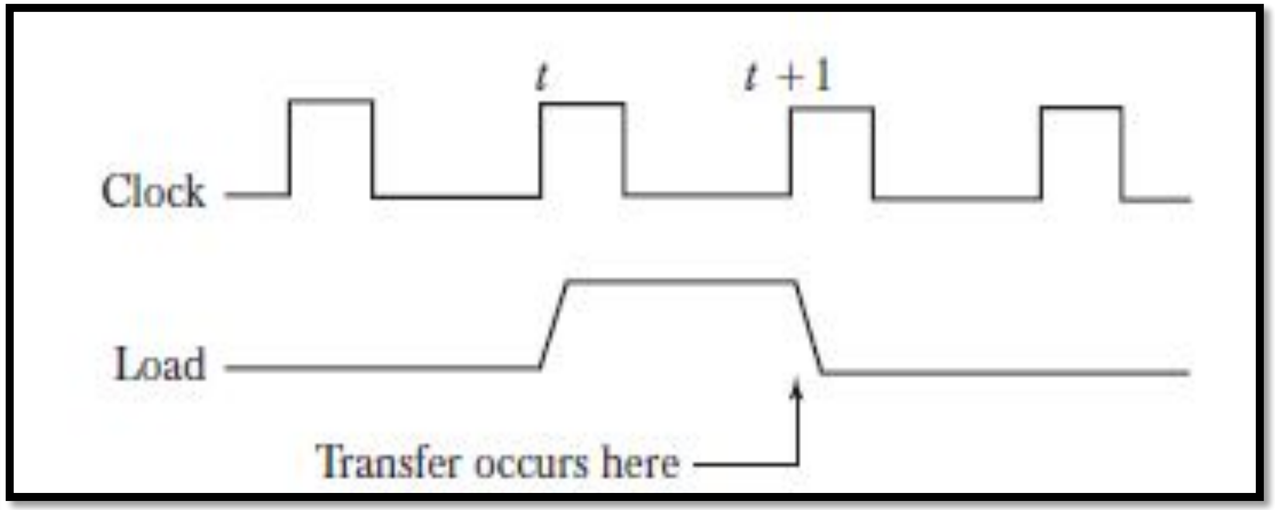
Every statement written in a register transfer notation implies a hardware construction for implementing the transfer.



Note:

It is assumed that the control variable is synchronized with the same clock as the one applied to the register.

Timing Diagram

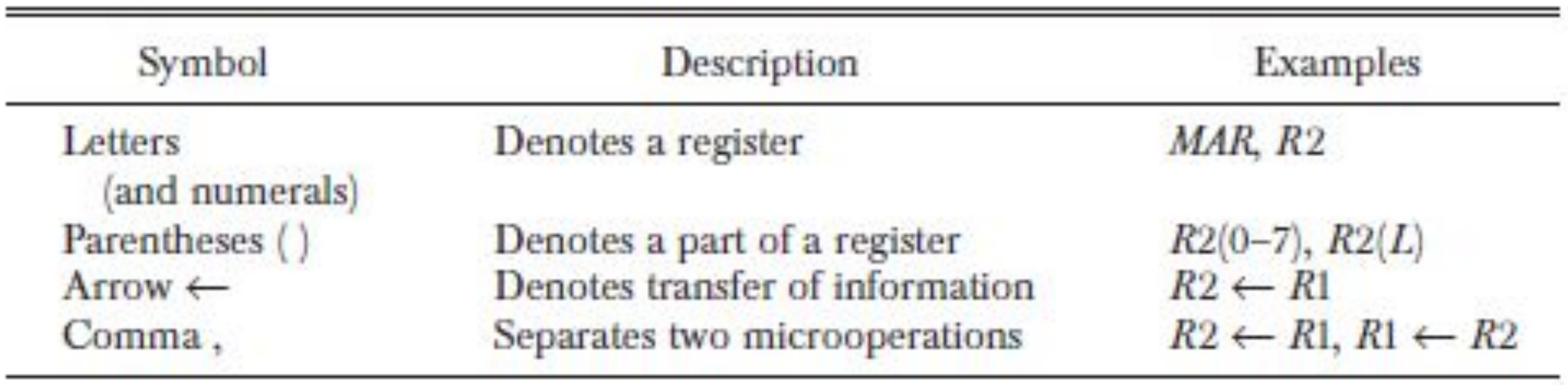


- As shown in the timing diagram, P is activated in the control section by the rising edge of the clock pulse at time t.
- The next positive transaction of the clock at time t+1 finds a load input active and the data inputs of *R2* are then loaded into the register in parallel.
- P may go back to 0 at time t+1;otherwise, the transfer will occur with every clock pulse

Note

- Clock is not included as a variable in the register transfer statements.
- It is assumed that all transfers occur during a clock edge transition
- Even though the control condition such as P becomes active just after time t, the actual transfer does not occur until the register is triggered by the next positive transition of the clock at time t +1.

Basic Symbols for Register Transfers



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

$$\boxed{\mathbf{T: R2 \leftarrow R1, R1 \leftarrow R2}}$$

Denotes an operation that exchanges the contents of two registers during one common clock pulse provided that $T = 1$.

This simultaneous operation is possible with registers that have edge-triggered flip-flops.

Bus and Memory Transfers

Bus Transfer

Issue

- A typical digital computer has many registers, and paths must be provided to transfer information from one register to another.
- The **number of wires will be excessive** if separate lines are used between each register and all other registers in the system.

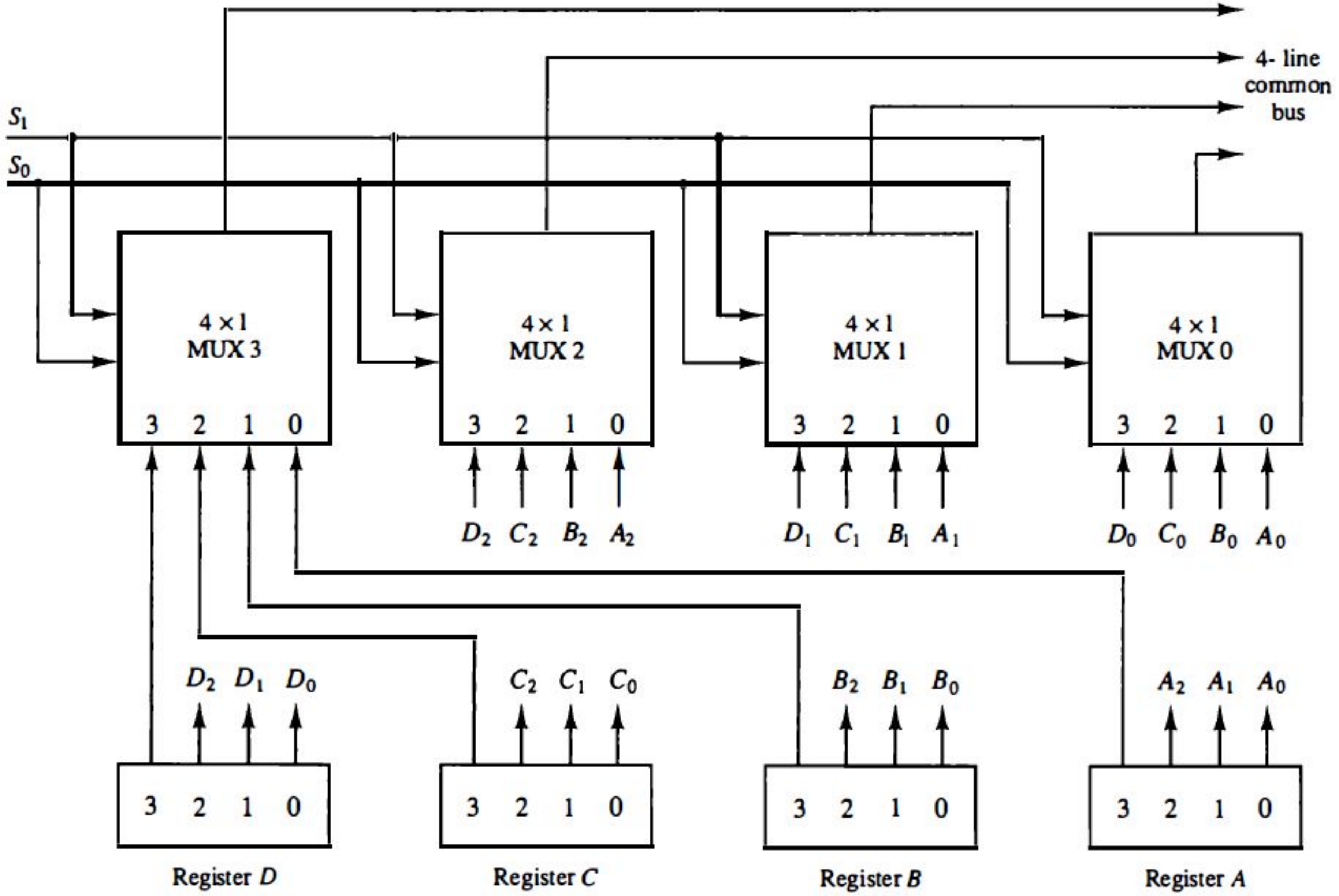
Common Bus

- A more efficient scheme for transferring information between registers in a multiple-register configuration is a **common bus system.**
- A bus structure consists of a **set of common lines, one for each bit** of a register, through which binary information is transferred one at a time.
- **Control signals** determine **which register is selected** by the bus during each particular register transfer.

Constructing Common Bus-System using Multiplexers

Bus System for Four Registers

The multiplexers select the source register whose binary information is then placed on the bus.



Bus Selection

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

Solve

- Number of Registers = 8
- Each Register= 16-bits
- How many Mux you need?
- What is the size of each Mux?

Solution

- Number of Registers = 8
- Each Register= 16-bits
- How many Mux you need?
- 16
- What is the size of each Mux?
- 8 x 1

Bus System

- In general, a bus system will multiplex k registers of n bits each to produce an n-line common bus.
- The number of multiplexers needed to construct the bus is equal to n , the number of bits in each register.
- The size of each multiplexer must be k x 1 since it multiplexes k data lines.
- For example, a common bus for eight registers of 16 bits each requires 16 multiplexers, one for each line in the bus. Each multiplexer must have eight data input lines and three selection lines to multiplex one significant bit in the eight registers.

Transfer from Bus to Destination Registers

- The transfer of information from a bus into one of many destination registers can be accomplished by **connecting the bus lines to the inputs of all destination registers and activating the load control of the particular destination register selected**.

The symbolic statement for a bus transfer

$$BUS \leftarrow C, \qquad R1 \leftarrow BUS$$

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.

If the bus is known to exist in the system, it may be convenient just to show the direct transfer.

$$R1 \leftarrow C$$

From this statement the designer knows **which control signals must be activated** to produce the transfer through the bus.

Memory Transfer

Basics of Memory

Memory Unit

- 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.**
- The bits that make up a word in memory are **treated as a single entity** that can be read and written as a whole.
- A memory word is a group of l's and 0's and may represent **a number, an instruction code, one or more alphanumeric characters, or any other binary-coded information.**
- A group of eight bits is called a **byte.**

Capacity of Memory

- The capacity of memories in commercial computers is usually stated as the total number of bytes that can be stored
- Most computer memories use words whose number of bits is a multiple of 8.
  - Thus a 16-bit word contains two bytes, and a 32-bit word is made up of four bytes.
- The **internal structure of a memory unit** is specified by **the number of words it contains** **and** **the number of bits in each word.**

Address Lines

- Special input lines called address lines select one particular word.
- Each word in memory is assigned an identification number, called an address, starting from 0 and continuing with 1, 2, 3, up to $\mathbf{2^{K-1}}$ where k is the number of address lines.
- The selection of a specific word inside the memory is done by applying the k-bit binary address to the address lines.
- A decoder inside the memory accepts this address and opens the paths needed to select the bits of the specified word.

Find Number of Address Lines

1. Computer Memory with 1024 Words

Number of Address Lines=?

Ans: **10 (since $2^{10} = 1024$)**

Note

- It is customary to refer to the **number of words (or bytes)** in a memory with one of the letters K (kilo), M (mega), or G (giga).
- K=Kilo=2^{10}
- M=Mega= 2^{20}
- G=Giga= 2^{30}

Example:

$$\mathbf{2^{16} = 2^6 K = 64K}$$

$$\mathbf{2^{21} = 2M}$$

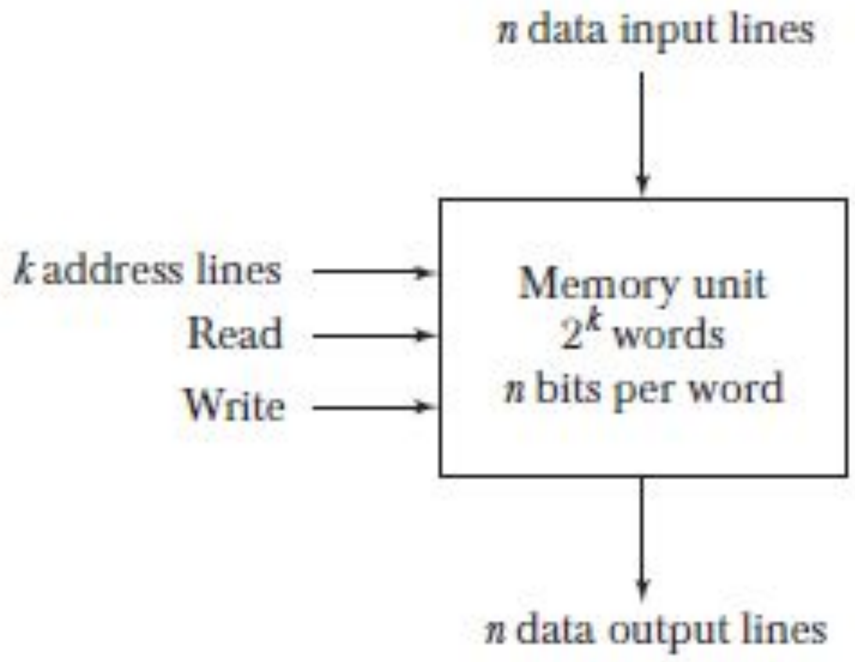
$$\mathbf{2^{32} = 4G}$$

Two major types of memories are used in computer systems

- Random-Access Memory (RAM) and
- Read-Only Memory (ROM)

Random-Access Memory (RAM)

Block diagram of random access memory (RAM).



Communication between a memory and its environment is achieved through data input and output lines, address selection lines, and control lines that specify the direction of transfer.

Operations Performed by RAM

- The two operations that a random-access memory can perform are the **write and read operations**
- The write signal specifies a transfer-in operation and the read signal specifies a transfer-out operation.
- On accepting one of these control signals, the internal circuits inside the memory provide the desired function.

Steps: Write Operation

The steps that must be taken for the purpose of transferring a new word to be stored into memory are as follows:

1. Apply the binary address of the desired word into the address lines.
2. Apply the data bits that must be stored in memory into the data input lines.
3. Activate the write input.

The memory unit will then take the bits presently available in the input data lines and store them in the word specified by the address lines.

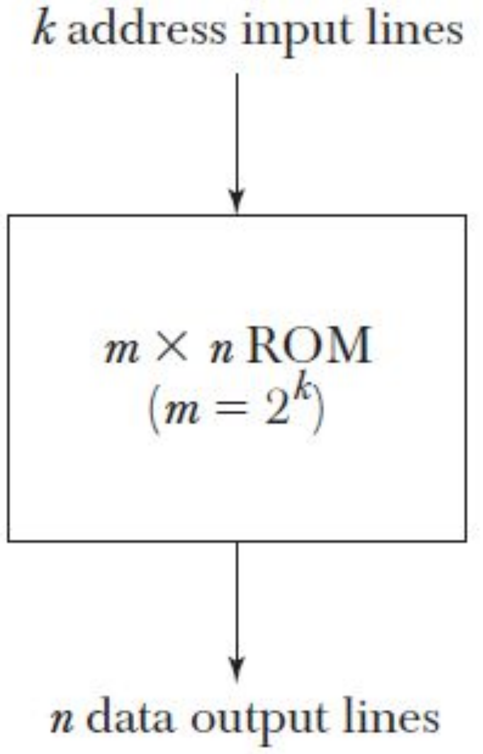
Steps: Read Operation

The steps that must be taken for the purpose of transferring a stored word out of memory are as follows:

1. Apply the binary address of the desired word into the address lines.
2. Activate the read input.

- The memory unit will then take the bits from the word that has been selected by the address and apply them into the output data lines.
- The content of the selected word does not change after reading.

Read-Only Memory



Block diagram of read only memory (ROM).

As the name implies, a read-only memory (ROM) is a memory unit that performs the read operation only; it does not have a write capability.

This implies that the binary information stored in a ROM is made permanent during the hardware production of the unit and cannot be altered by writing different words into it.

Memory Transfer

Memory Read and Memory Write

- The transfer of information from a memory word to the outside environment is called a **read operation.**
- The transfer of new information to be stored into the memory is called a **write operation.**
- **A memory word will be symbolized by the letter M.**
- The particular memory word among the many available is selected by the memory address during the transfer.
- It is necessary to specify the address of M when writing memory transfer operations.
- This will be done by **enclosing the address in square brackets following the letter M.**

Read Operation (Microoperation)

- Consider a **memory unit that receives the address from a register, called the address register, symbolized by AR. The data are transferred to another register, called the data register, symbolized by DR.**
- The read operation can be stated as follows:

$\boxed{\textbf{Read: DR} \leftarrow \textbf{M[AR]}}$

"This causes a transfer of information into DR from the memory word M selected by the address in AR."

Write Operation (Microoperation)

- The write operation transfers the content of a data register to a memory word M selected by the address.
- Assume that the input data are in register R1 and the address is in AR.
- The write operation can be stated symbolically as follows:

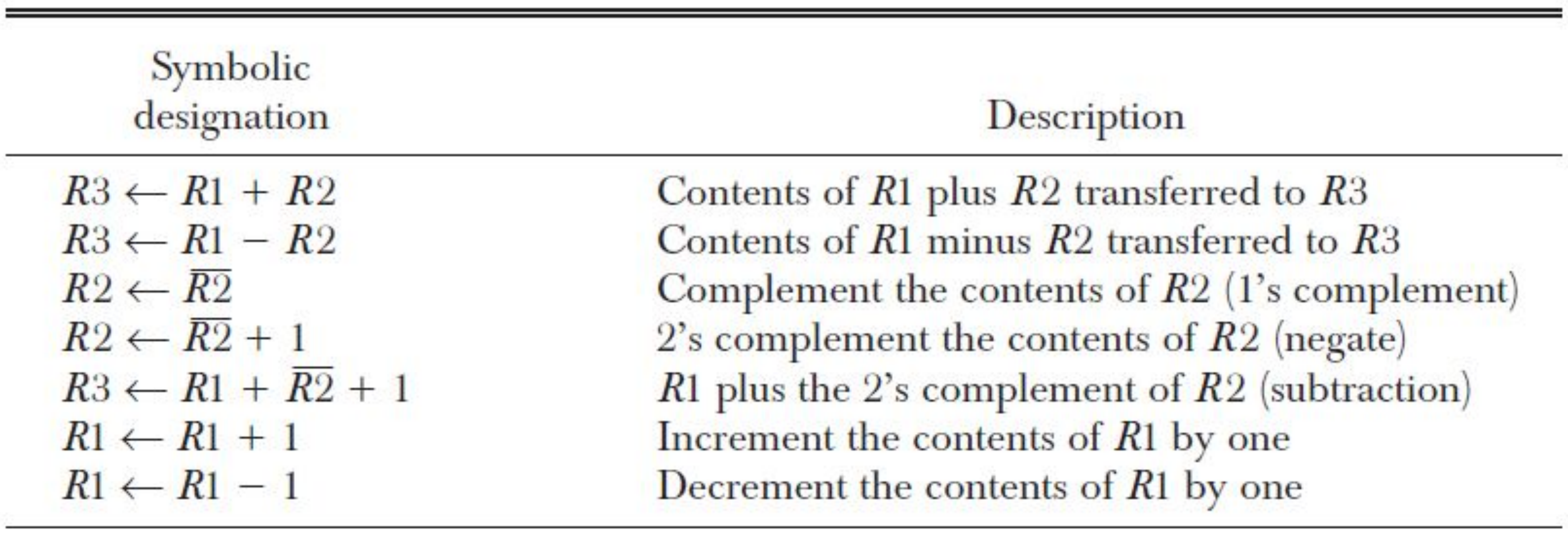
$$\boxed{\textbf{Write: } M[AR] \leftarrow R1}$$

"This causes a transfer of information from R1 into the memory word M selected by the address in AR."

Arithmetic Microoperations

Arithmetic Microoperations

- The basic arithmetic microoperations are **addition, subtraction, increment, decrement, and shift.**



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

Add Microoperation

- The arithmetic microoperation defined by the statement

$$\boxed{R3 \leftarrow R1 + R2}$$

- It states that the contents of register R1 are added to the contents of register R2 and the sum transferred to register R3.
- **To implement this statement with hardware we need three registers and the digital component that performs the addition operation.**

Subtract Microoperation

- Subtraction is most often implemented through complementation and addition.
- Instead of using the minus operator, we can specify the subtraction by the following statement:

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

- $\overline{\boldsymbol{R2}}$ is the symbol for the 1’s complement of R2. Adding 1 to the 1’s complement produces the 2’s complement. Adding the contents of R1 to the 2’s complement of R2 is equivalent to $\boxed{\boldsymbol{R1 - R2}}$

Note

- In most computers, the multiplication operation is implemented with a sequence of add and shift microoperations.
- Division is implemented with a sequence of subtract and shift microoperations.

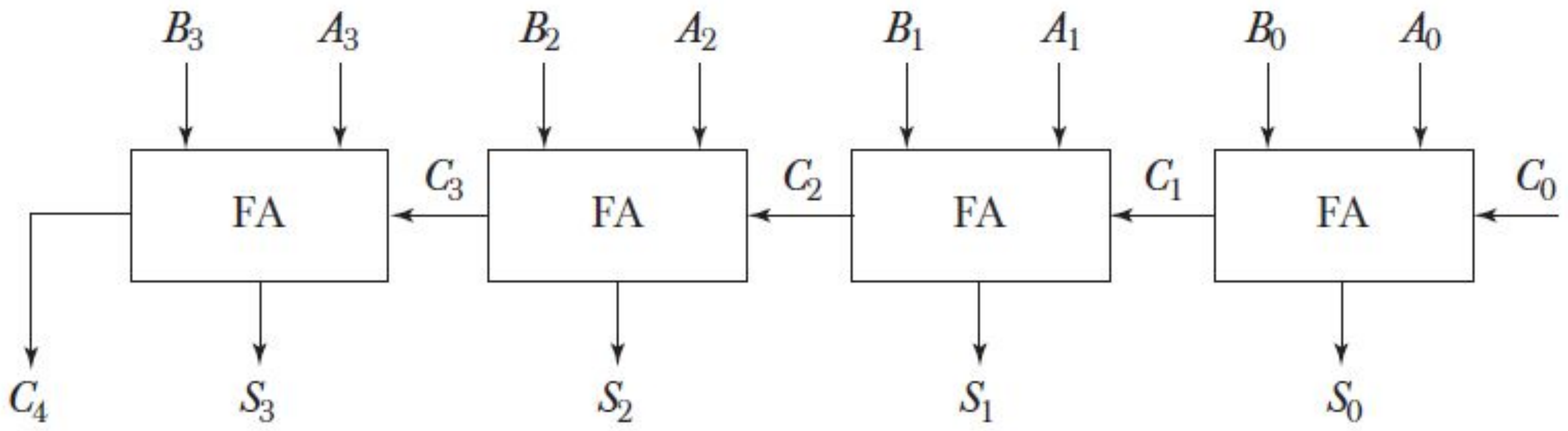
Question

- The increment and decrement microoperations are implemented with a **combinational circuit** or **with a binary up-down counter**.

Binary Adder

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

4-bit Binary Adder

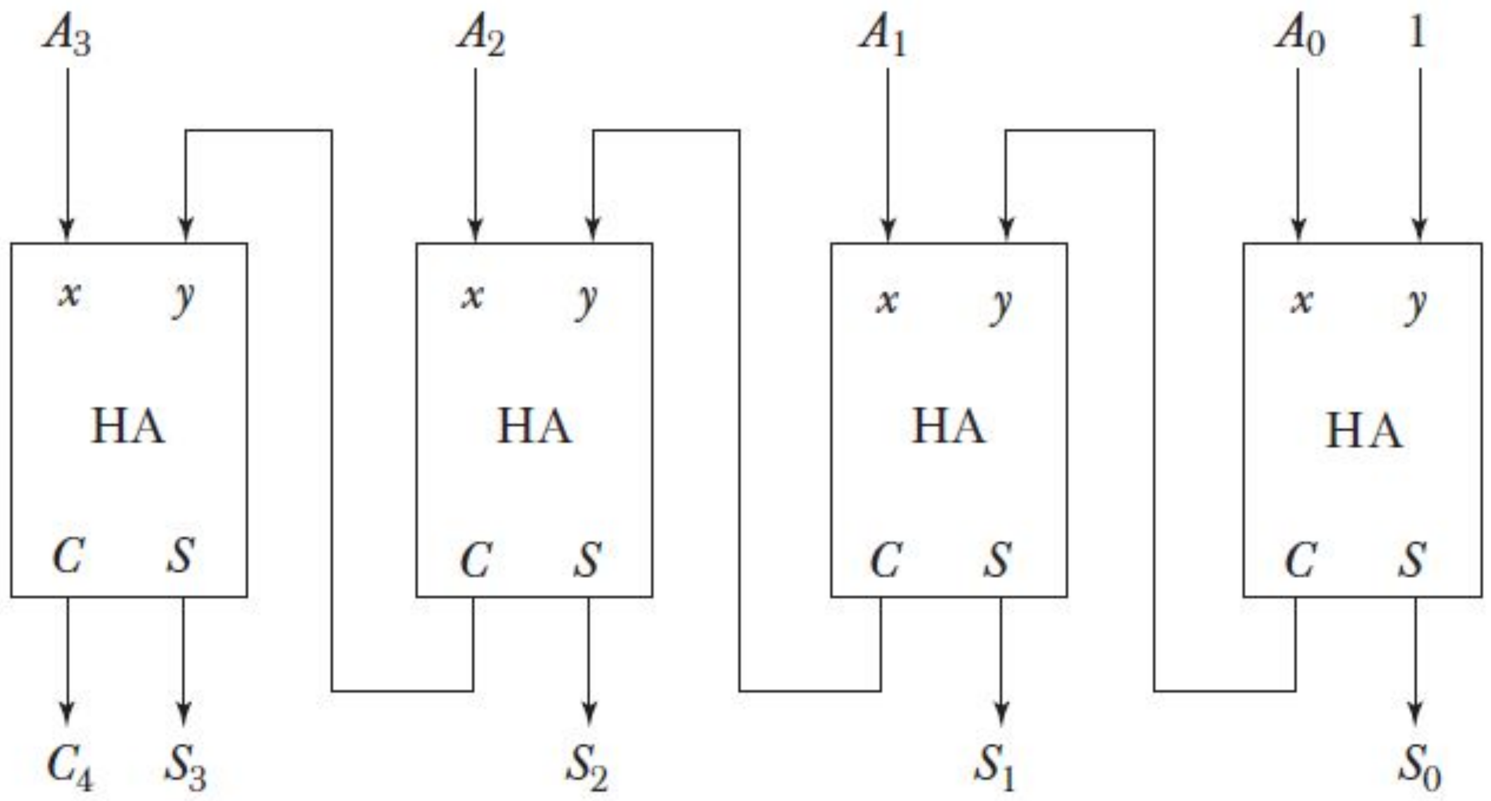


n-bit Binary Adder

- An n-bit binary adder requires n full-adders.
- The output carry from each full-adder is connected to the input carry of the next-high-order full-adder.
- The n data bits for the A inputs come from one register (such as R1), and the n data bits for the B inputs come from another register (such as R2).
- The sum can be transferred to a third register or to one of the source registers (R1 or R2), replacing its previous content.

Binary Incrementer

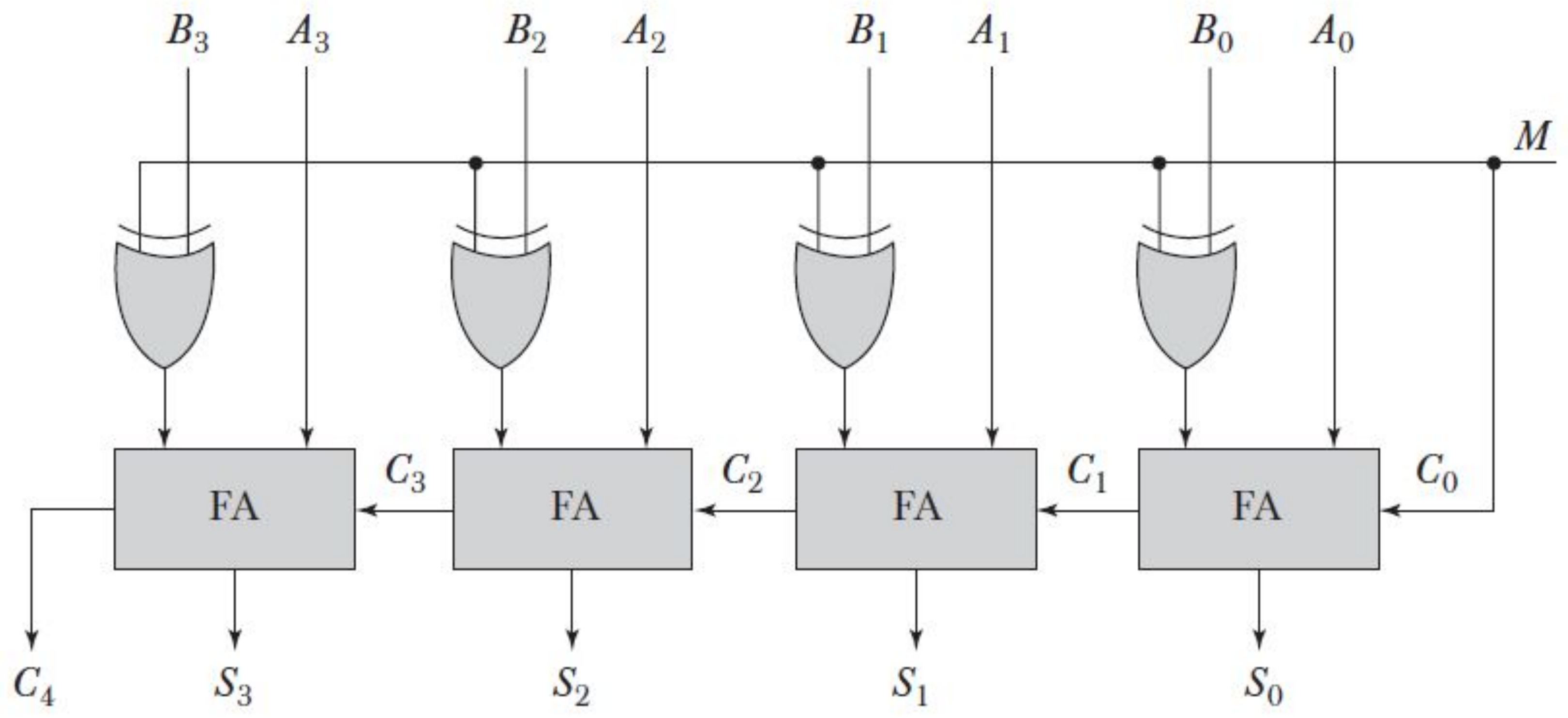
4-bit binary incrementer



One of the inputs to the least significant half-adder (HA) is connected to logic-1 and the other input is connected to the least significant bit of the number to be incremented.
The output carry from one half-adder is connected to one of the inputs of the next-higher-order half-adder.
The circuit receives the four bits from A_0 through A_3, adds one to it, and generates the incremented output in S_0 through S_3. The output carry C_4 will be 1 only after incrementing binary 1111.

Binary Adder-Subtractor

4-bit adder-subtractor



The mode input M controls the operation.
When M =0 the circuit is an adder and
when M =1 the circuit becomes a subtractor.
The 1's complement can be implemented with inverters and a one can be added to the sum through the input carry.
For unsigned numbers, this gives A -B if $A \geq B$ (or)
the 2's complement of (B-A) if $A < B$.

For signed numbers, the result is A -B provided that there is no overflow.

Detailed Explanation (REFER NOTES)

Binary Incrementer

4-bit synchronous binary counter

Every time the count enable is active, the clock pulse transition
increments the content of the register by one.

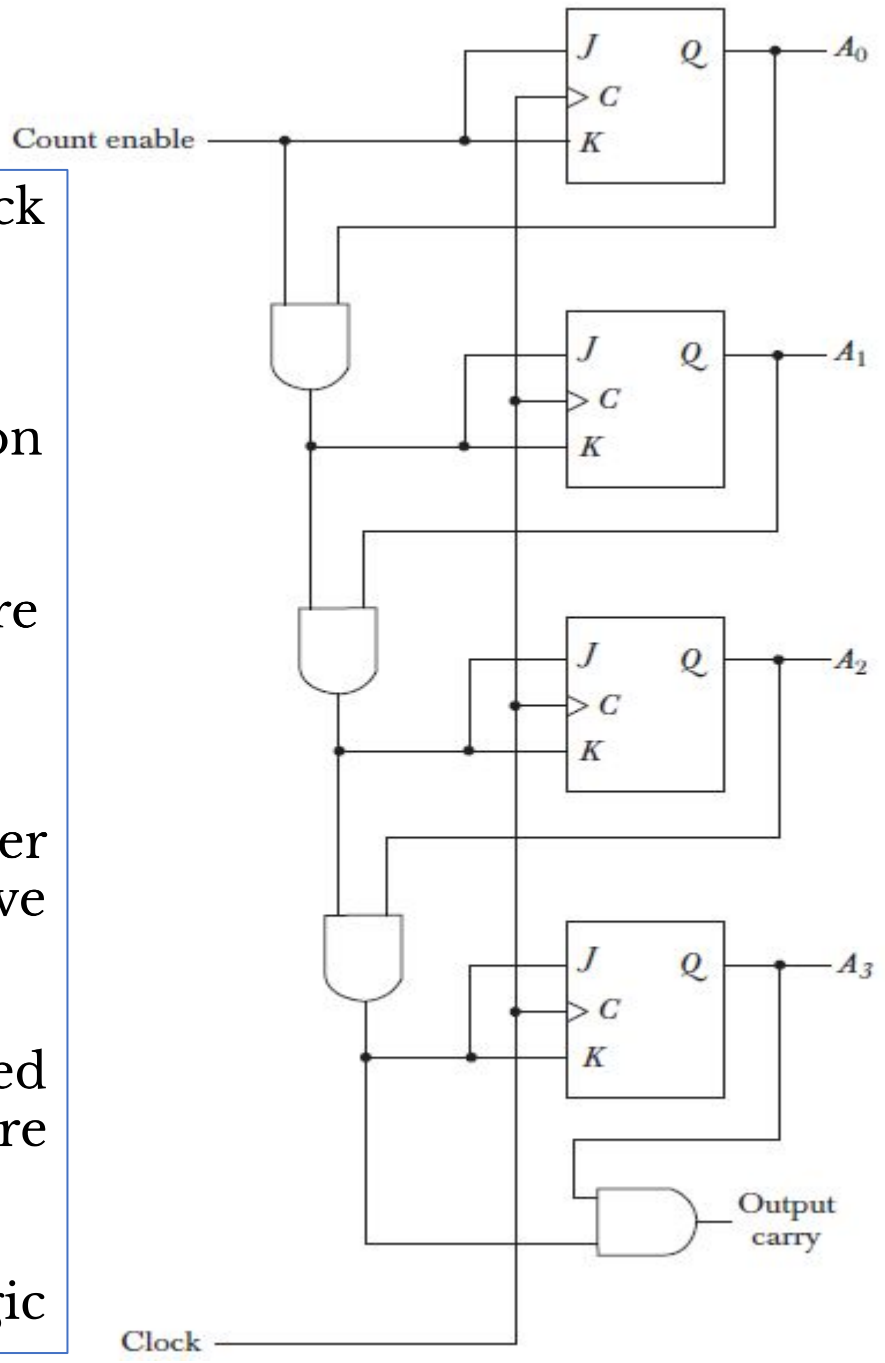
The *C* inputs of all flip-flops receive the common clock.

If the count enable is 0, all *J* and *K* inputs are maintained at 0 and
the output of the counter does not change.

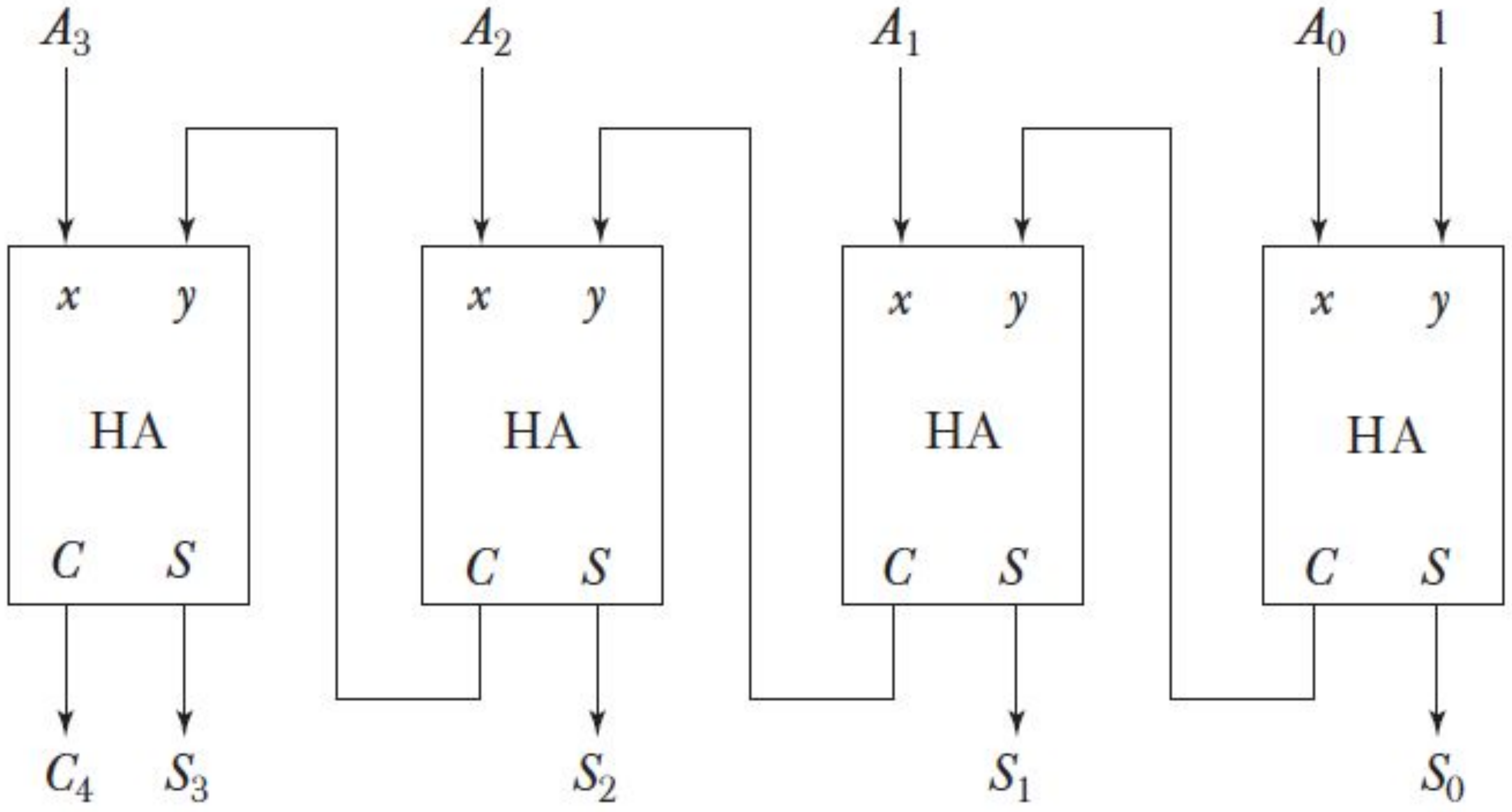
The first stage A_0 is complemented when the counter is enabled and the clock goes through a positive transition.

Each of the other three flip-flops are complemented when all previous least significant flip-flops are equal to 1 and the count is enabled.

The chain of AND gates generate the required logic for the *J* and *K* inputs.



4-bit binary incrementor



Arithmetic Circuit

4-bit arithmetic circuit.

The basic component of an arithmetic circuit is the parallel adder.

By controlling the data inputs to the adder, it is possible to obtain different types of arithmetic operations.

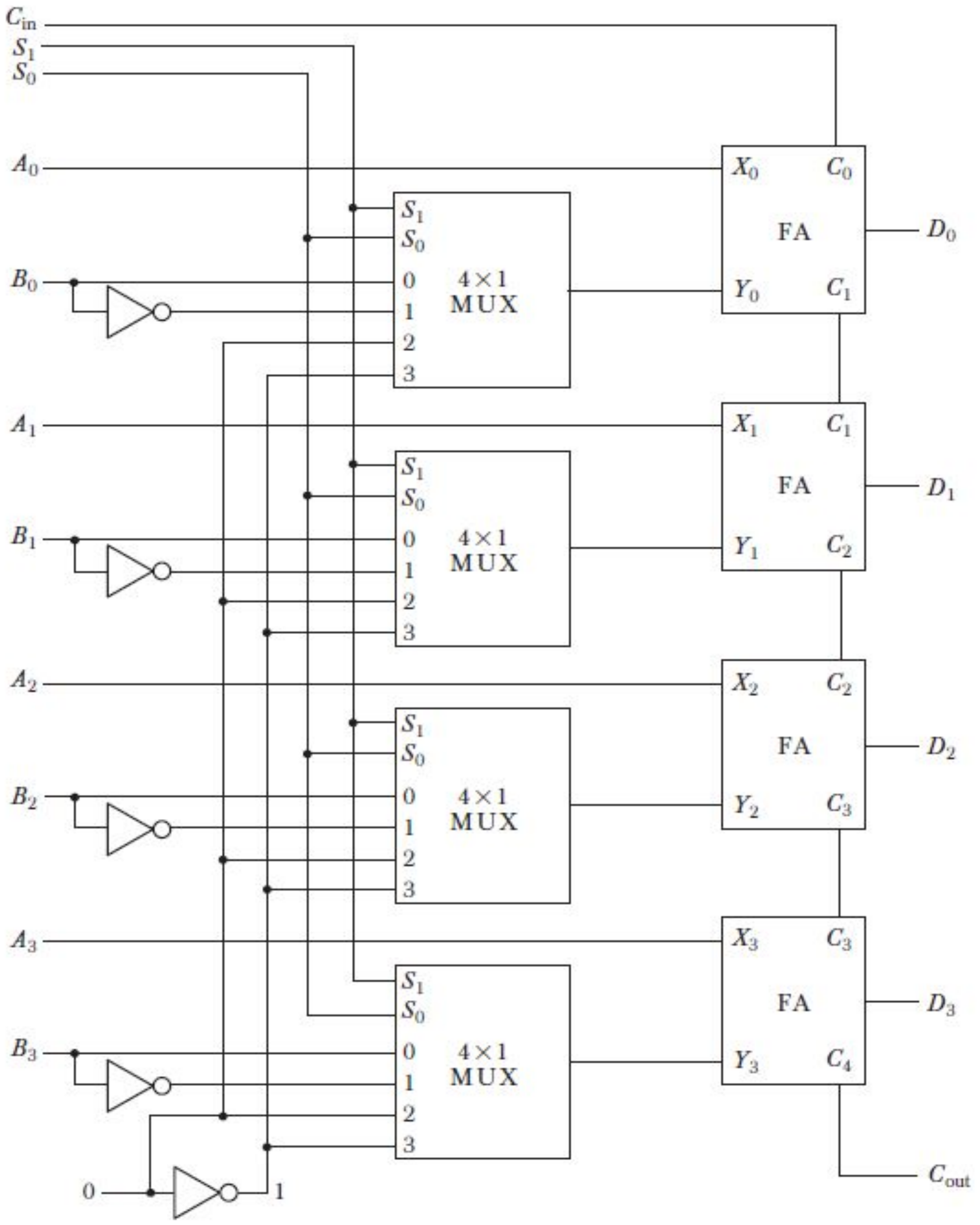
The output of the binary adder is calculated from the following arithmetic sum:

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

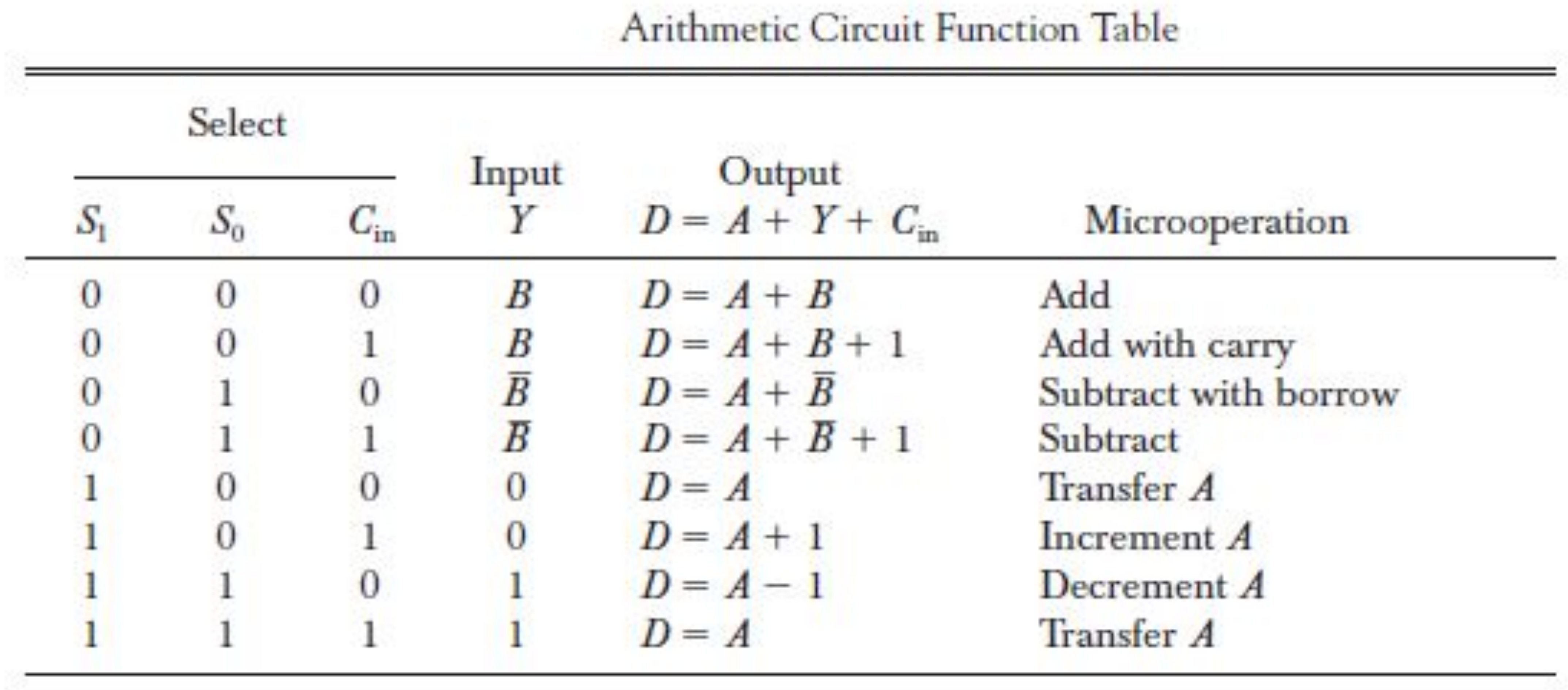
where *A* is the 4-bit binary number at the *X* inputs and *Y* is the 4-bit binary number at the *Y* inputs of the binary adder.

*C*in is the input carry, which can be equal to 0 or 1.

By controlling the value of *Y* with the two selection inputs *S*1 and *S*0 and making *C*in equal to 0 or 1, it is possible to generate the eight arithmetic microoperations



Arithmetic Circuit Function Table



Arithmetic Circuit Function Table

Select			Input	Output	
S_1	S_0	C_{in}	Y	$D = A + Y + C_{in}$	Microoperation
0	0	0	B	$D = A + B$	Add
0	0	1	B	$D = A + B + 1$	Add with carry
0	1	0	$\overline{B}$	$D = A + \overline{B}$	Subtract with borrow
0	1	1	$\overline{B}$	$D = A + \overline{B} + 1$	Subtract
1	0	0	0	$D = A$	Transfer A
1	0	1	0	$D = A + 1$	Increment A
1	1	0	1	$D = A - 1$	Decrement A
1	1	1	1	$D = A$	Transfer A

Logic Microoperations

Logic Microoperations

- Logic microoperations specify binary operations for strings of bits stored in registers.

These operations consider each bit of the register separately and treat them as binary variables.

For example, the exclusive-OR microoperation with the contents of two registers R1 and R2 is symbolized by the statement

$$\boxed{\boldsymbol{P : R1 \leftarrow R1 \oplus R2}}$$

It specifies a logic microoperation (exclusive-OR) to be executed on the individual bits of the registers provided that the control variable $\boldsymbol{P = 1}$.

Logic Microoperations: Applications

- The logic microoperations are seldom used in scientific computations, but they are very useful for bit manipulation of binary data and for making logical decisions.

Symbols

S. No	Microoperation	Symbol
1	AND	
2	OR	
3	Exclusive-OR	

Interpret the following Microoperation

- $$\mathbf{P + Q: R1 \leftarrow R2 + R3, R4 \leftarrow R5 \vee R6}$$

- The + between P and Q is an OR operation between two binary variables of a control function.
- The + between R2 and R3 specifies an add microoperation.
- The OR microoperation is designated by the symbol $\vee$ between registers R5 and R6.

Note
When the symbol + occurs in a microoperation, it will denote an arithmetic plus. When it occurs in a control (or Boolean) function, it will denote an OR operation. We will never use it to symbolize an OR microoperation.

Logic Microoperations

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

The 16 logic microoperations are derived from these functions by replacing variable x by the binary content of register A and variable y by the binary content of register B.

It is important to realize that the Boolean functions listed in the first column of Table represent a relationship between two binary variables x and y.

The logic microoperations listed in the second column represent a relationship between the binary content of two registers A and B.

Each bit of the register is treated as

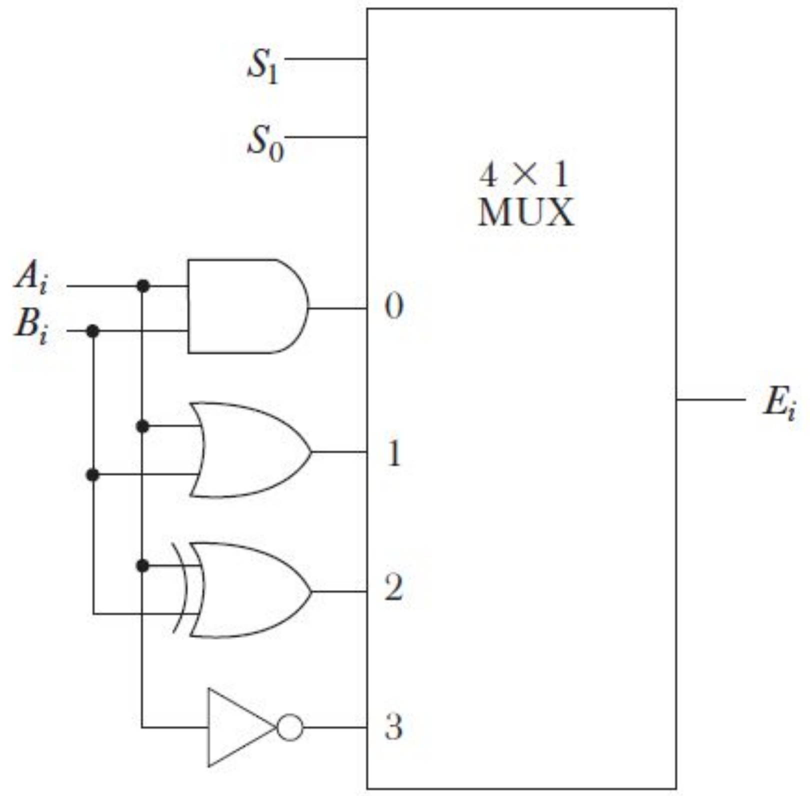
Boolean function	Microoperation	Name
$F_0 = 0$	$F \leftarrow 0$	Clear
$F_1 = xy$	$F \leftarrow A \wedge B$	AND
$F_2 = xy'$	$F \leftarrow A \wedge \overline{B}$	
$F_3 = x$	$F \leftarrow A$	Transfer A
$F_4 = x'y$	$F \leftarrow \overline{A} \wedge B$	
$F_5 = y$	$F \leftarrow B$	Transfer B
$F_6 = x \oplus y$	$F \leftarrow A \oplus B$	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

Truth Table

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

Hardware Implementation

One stage of logic circuit.



(a) Logic diagram

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

(b) Functional table

The diagram shows one typical stage with subscript *i*.

For a logic circuit with *n* bits, the diagram must be repeated *n* times for
$i = 0, 1, 2, \ldots, n - 1$.

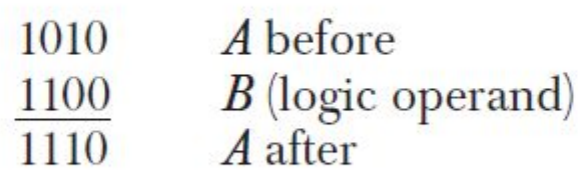
The selection variables are applied to all stages.

Some Applications

- Logic microoperations are very useful for manipulating individual bits or a portion of a word stored in a register.
- They can be used to change bit values, delete a group of bits, or insert new bit values into a register.
- The following examples show how the bits of one register (designated by A) are manipulated by logic microoperations as a function of the bits of another register (designated by B).
- In a typical application, register A is a processor register and the bits of register B constitute a logic operand extracted from memory and placed in register B.

Selective-Select

- 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 pos



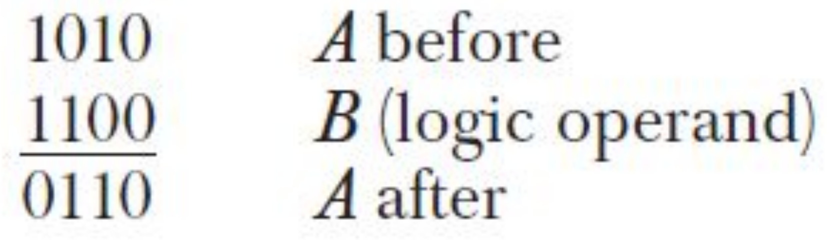
1010	*A* before
1100	*B* (logic operand)
1110	*A* after

The two leftmost bits of *B* are 1’s, so the corresponding bits of *A* are set to 1.
One of these two bits was already set and the other has been changed from 0 to 1.

The **OR microoperation** can be used to selectively set bits of a register.

Selective Complement

- The *selective-complement* operation complements bits in *A* where there are corresponding 1's in *B*. It does not affect bit positions that have 0's in *B*.

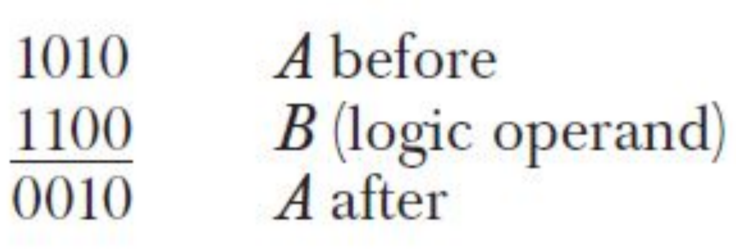


1010	*A* before
1100	*B* (logic operand)
0110	*A* after

Again the two leftmost bits of *B* are 1's, so the corresponding bits of *A* are complemented.
The **exclusive-OR** microoperation can be used to selectively complement bits of a register.

Selective-Clear

- The *selective-clear* operation clears to 0 the bits in *A* only where there are corresponding 1's in *B*.



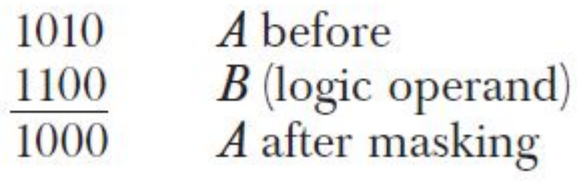
1010	*A* before
1100	*B* (logic operand)
0010	*A* after

Again the two leftmost bits of *B* are 1's, so the corresponding bits of *A* are cleared to 0.

The corresponding logic microoperation is $A \leftarrow A \wedge \bar{B}$

Mask

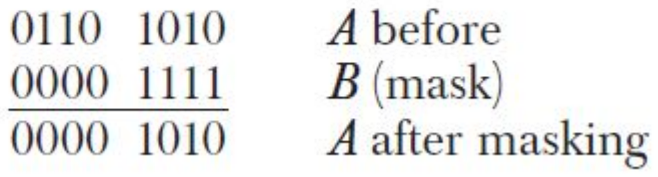
- The *mask* operation is similar to the selective-clear operation except that the bits of *A* are cleared only where there are corresponding 0's in *B*.
- The mask operation is an AND micro operation



1010	*A* before
1100	*B* (logic operand)
1000	*A* after masking

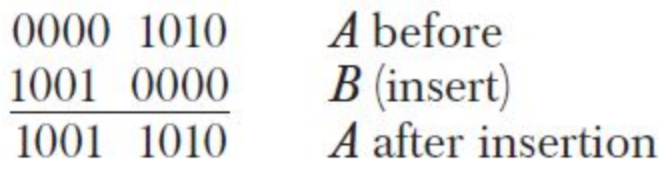
Insert

- The *insert* operation inserts a new value into a group of bits.
- This is done by first masking the bits and then ORing them with the required value.
- For example, suppose that an *A* register contains eight bits, 0110 1010.
- To replace the four leftmost bits by the value 1001 we first mask the four unwanted bits:



0110 1010	*A* before
0000 1111	*B* (mask)
0000 1010	*A* after masking

and then insert the new value:



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

The mask operation is an AND microoperation and the insert operation is an OR microoperation.

Clear

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

$$\begin{array}{ll} 1010 & A \\ \underline{1010} & B \\ 0000 & A \leftarrow A \oplus B \end{array}$$

Shift Microoperations

Shift Microoperations

- Shift microoperations are used for serial transfer of data.
- They are also used in conjunction with arithmetic, logic, and other data-processing operations.
- The contents of a register can be shifted to the left or the right.
- At the same time that the bits are shifted, the first flip-flop receives its binary information from the serial input.
- During a shift-left operation the serial input transfers a bit into the rightmost position.
- During a shift-right operation the serial input transfers a bit into the leftmost position.
- The information transferred through the serial input determines the type of shift.
- **There are three types of shifts:**

  1. logical,

  2. circular, and

  3. arithmetic.

logical shift

- A logical shift is one that **transfers 0** through the serial input.
- We will adopt, the symbols shl and shr for logical shift-left and shift-right microoperations.

$$R1 \leftarrow \text{shl } R1$$
$$R2 \leftarrow \text{shr } R2$$

are two microoperations that specify a 1-bit shift to the left of the content of register *R*1 and a 1-bit shift to the right of the content of register *R*2.

The bit transferred to the end position through the serial input is assumed to be 0 during a logical shift.

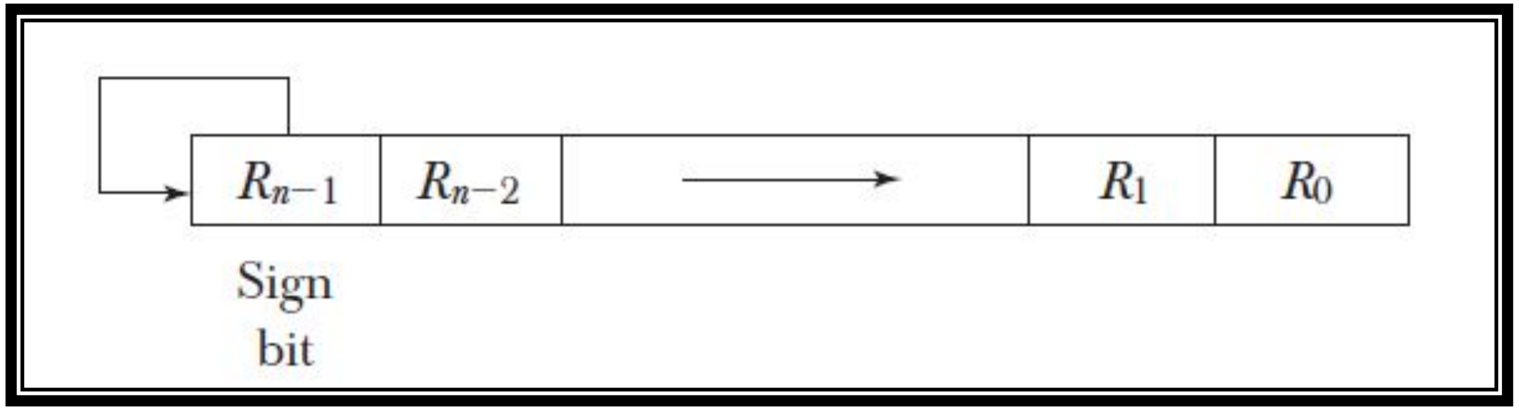
circular shift (rotate)

- The circular shift (also known as a rotate operation) circulates the bits of the register around the two ends without loss of information.
- This is accomplished by connecting the serial output of the shift register to its serial input.
- We will use the symbols cil and cir for the circular shift left and right, respectively.

arithmetic shift

- An arithmetic shift is a microoperation that shifts a signed binary number to the left or right.
- An arithmetic **shift-left multiplies a signed binary number by 2.**
- An arithmetic **shift-right divides the number by 2.**
- Arithmetic shifts must leave the sign bit unchanged because the sign of the number remains the same when it is multiplied or divided by 2.

Arithmetic shift right



- The leftmost bit in a register holds the sign bit, and the remaining bits hold the number.
- The sign bit is 0 for positive and 1 for negative.
- Negative numbers are in 2's complement form.
- Bit R_{n-1} in the leftmost position holds the sign bit.
- R_{n-2} is the most significant bit of the number and R_0 is the least significant bit.
- The arithmetic shift-right leaves the sign bit unchanged and shifts the number (including the sign bit) to the right.
- Thus R_{n-1} remains the same, R_{n-2} receives the bit from R_{n-1} , and so on for the other bits in the register.
- The bit in R_0 is lost.

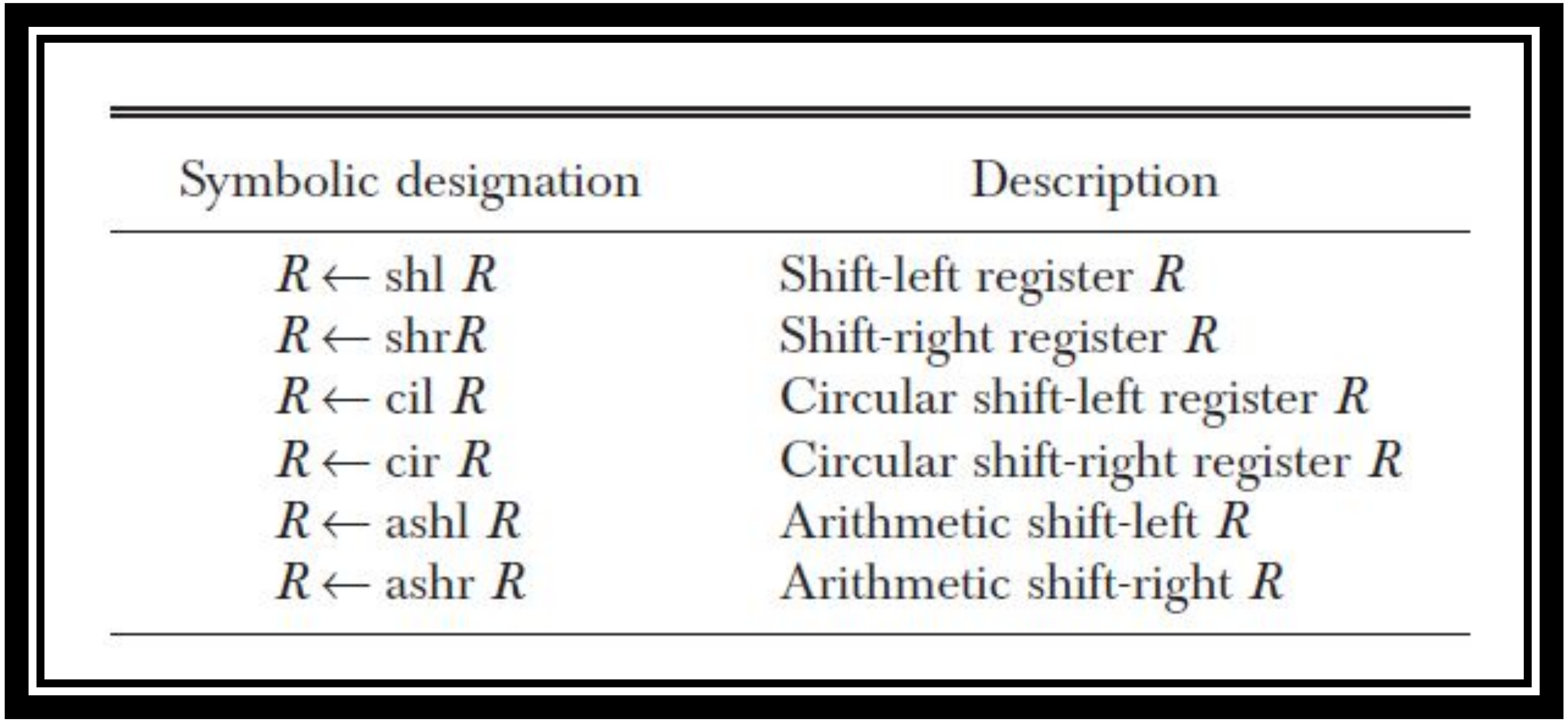
Arithmetic shift left

- The arithmetic shift-left inserts a 0 into R_0 , and shifts all other bits to the left.
- The initial bit of R_{n-1} is lost and replaced by the bit from R_{n-2} .
- A sign reversal occurs if the bit in R_{n-1} changes in value after the shift.
- This happens **if the multiplication by 2 causes an overflow.**
- An overflow occurs after an arithmetic shift left if initially, before the shift, R_{n-1} is not equal to R_{n-2} .
- **An overflow flip-flop $\boldsymbol{V_s}$ can be used to detect an arithmetic shift-left overflow.**

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

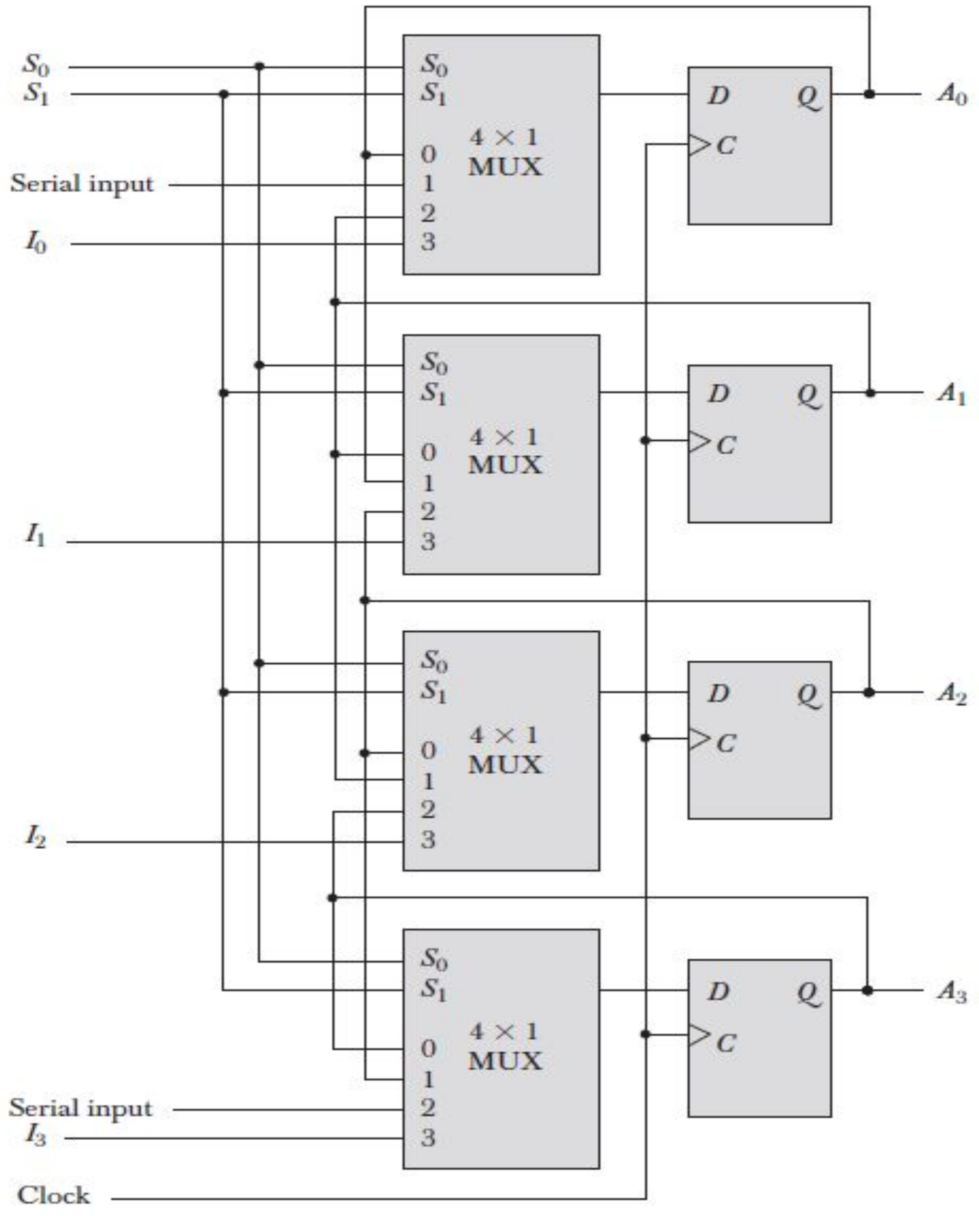
If $V_s = 0$, there is no overflow, but if $V_s = 1$, there is an overflow and a sign reversal after the shift. V_s must be transferred into the overflow flip-flop with the same clock pulse that shifts the register.

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

Hardware: Bidirectional shift register with parallel load

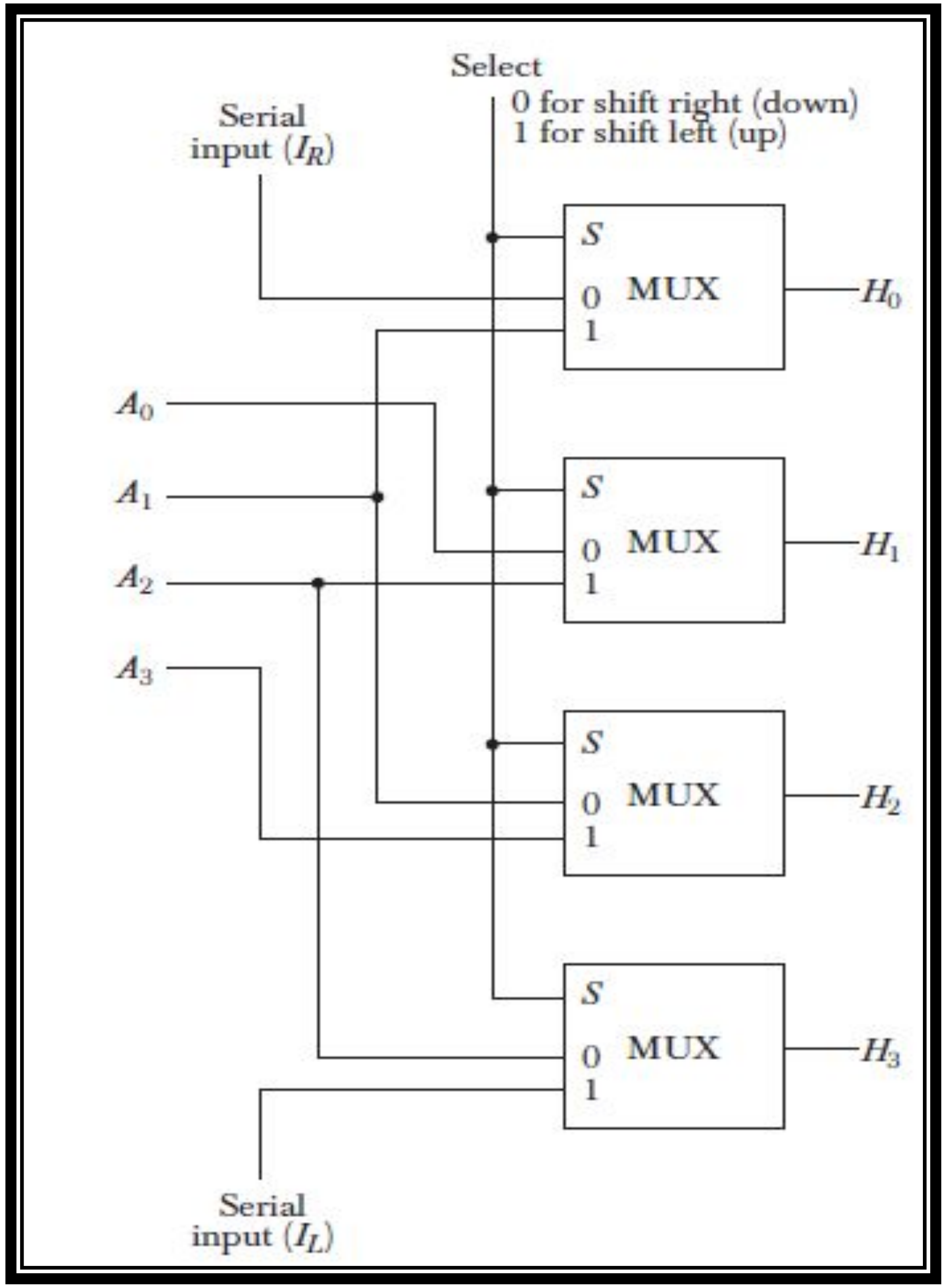


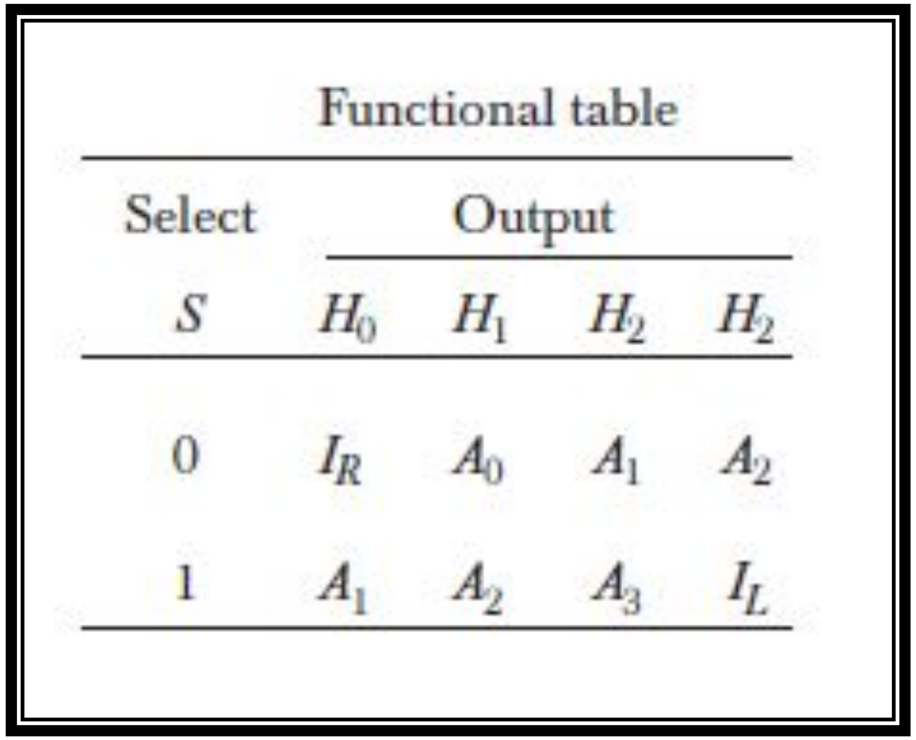
Mode control		
S_1	S_0	Register operation
0	0	No change
0	1	Shift right (down)
1	0	Shift left (up)
1	1	Parallel load

Issue: bidirectional shift register with parallel load

- Information can be transferred to the register in parallel and then shifted to the right or left.
- In this type of configuration, **a clock pulse is needed for loading the data into the register, and another pulse is needed to initiate the shift.**
- In a processor unit with many registers it is more efficient to implement the shift operation with a combinational circuit.
- In this way the content of a register that has to be shifted is first placed onto a common bus whose output is connected to the combinational shifter, and the shifted number is then loaded back into the register.
- This requires **only one clock pulse for loading the shifted value into the register.**

4-bit combinational circuit shifter





Functional table

Select	Output			
S	H_0	H_1	H_2	H_2
0	I_R	A_0	A_1	A_2
1	A_1	A_2	A_3	I_L

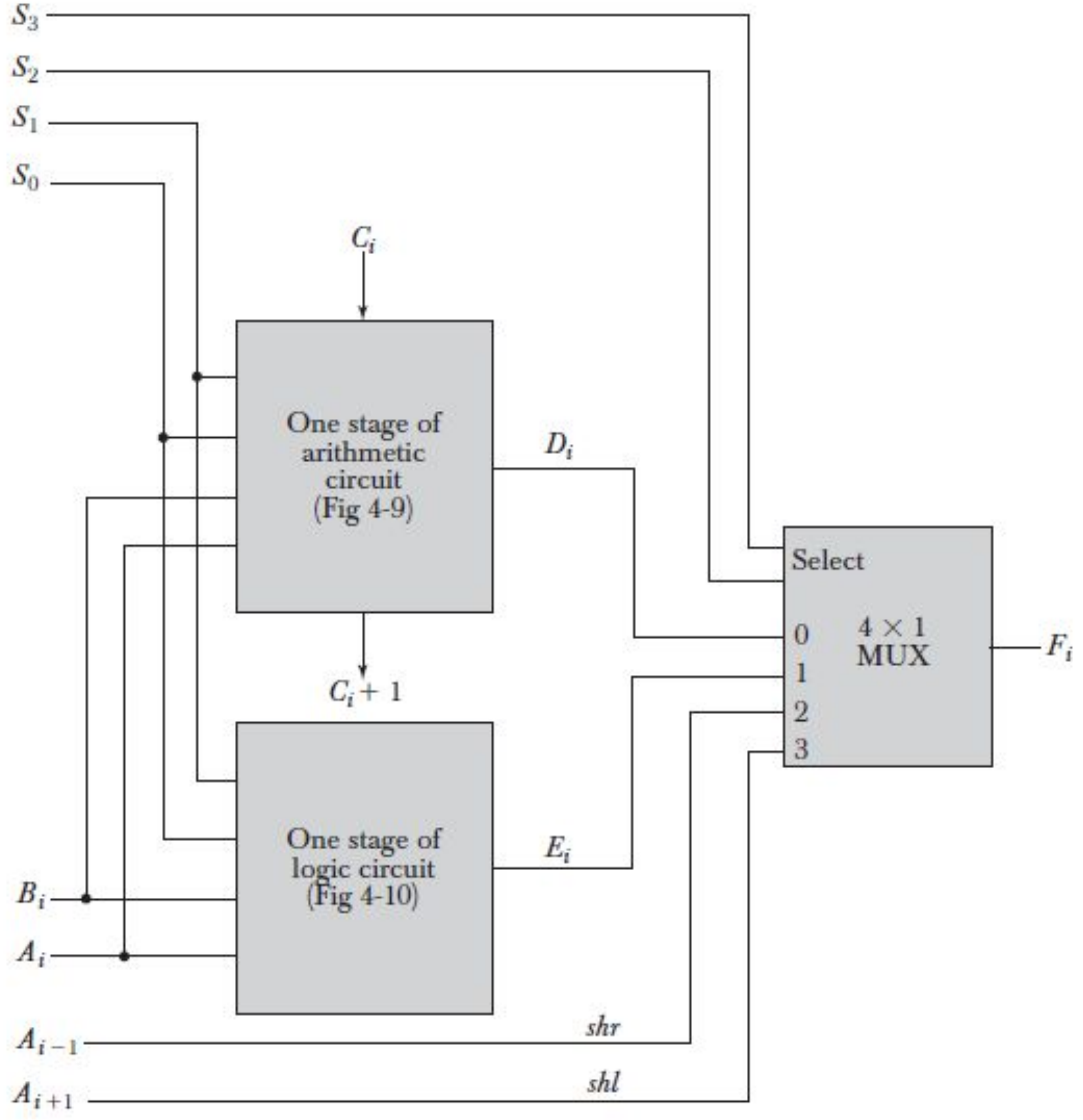
A shifter with *n* data inputs and outputs requires *n* multiplexers. The two serial inputs can be controlled by another multiplexer to provide the three possible types of shifts.

Arithmetic Logic Shift Unit

Arithmetic Logic Unit (ALU)

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

One stage of arithmetic logic shift unit



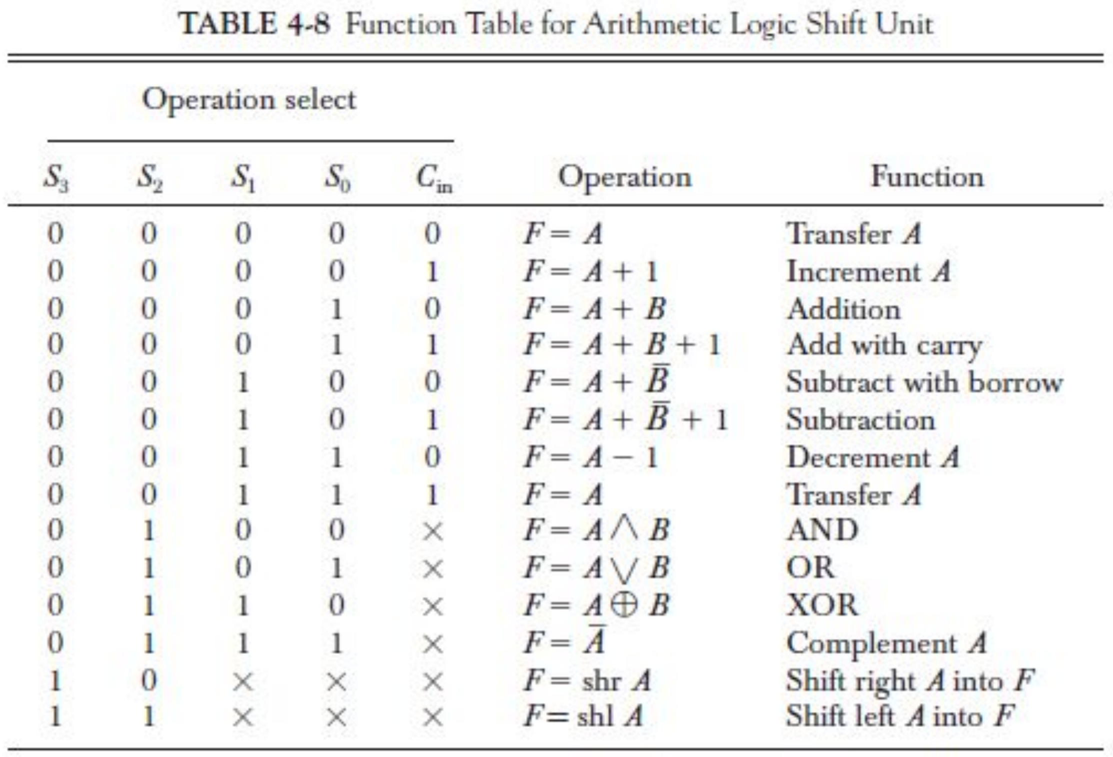


TABLE 4-8 Function Table for Arithmetic Logic Shift Unit

Operation select						
S_3	S_2	S_1	S_0	C_{in}	Operation	Function
0	0	0	0	0	$F = A$	Transfer A
0	0	0	0	1	$F = A + 1$	Increment A
0	0	0	1	0	$F = A + B$	Addition
0	0	0	1	1	$F = A + B + 1$	Add with carry
0	0	1	0	0	$F = A + \overline{B}$	Subtract with borrow
0	0	1	0	1	$F = A + \overline{B} + 1$	Subtraction
0	0	1	1	0	$F = A - 1$	Decrement A
0	0	1	1	1	$F = A$	Transfer A
0	1	0	0	×	$F = A \wedge B$	AND
0	1	0	1	×	$F = A \vee B$	OR
0	1	1	0	×	$F = A \oplus B$	XOR
0	1	1	1	×	$F = \overline{A}$	Complement A
1	0	×	×	×	$F =$ shr A	Shift right A into F
1	1	×	×	×	$F =$ shl A	Shift left A into F

- Note that the diagram shows just one typical stage.
- The circuit must be repeated *n* times for an *n*-bit ALU.

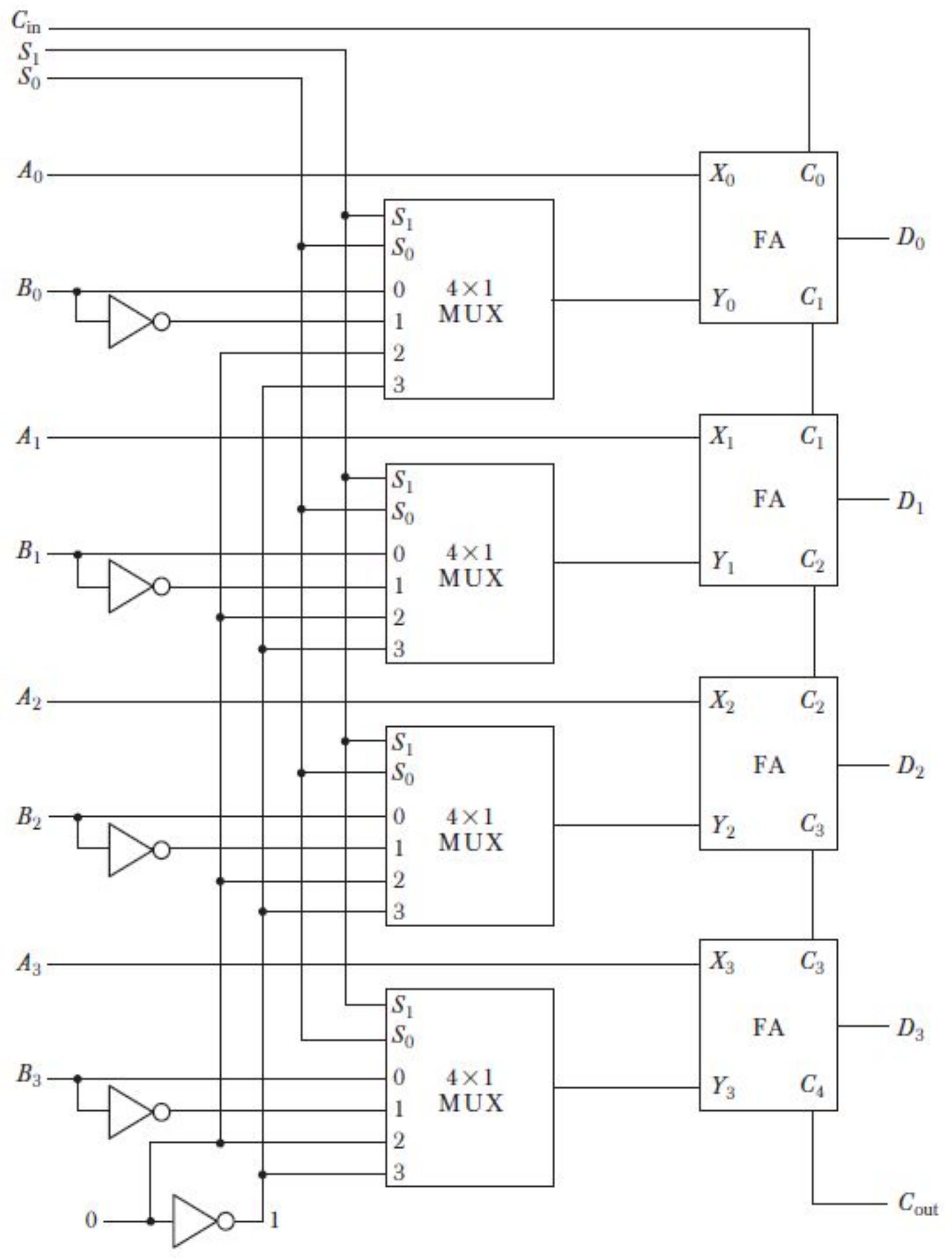
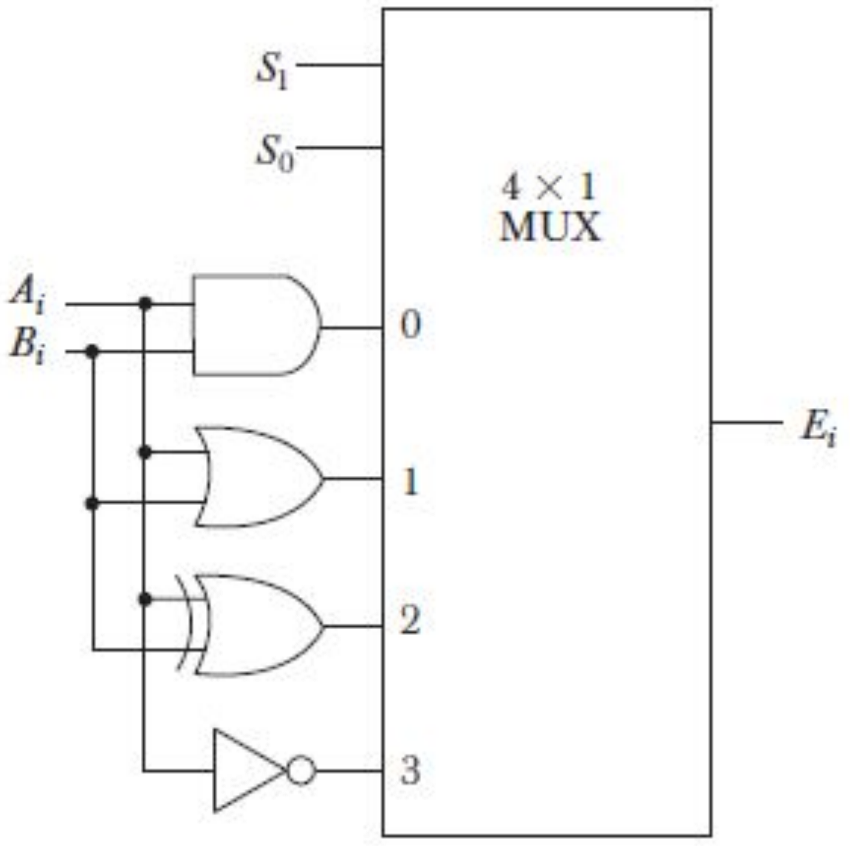


Figure 4-9 4-bit arithmetic circuit.

TABLE 4-4 Arithmetic Circuit Function Table

Select			Input	Output	
S_1	S_0	C_{in}	Y	$D = A + Y + C_{in}$	Microoperation
0	0	0	B	$D = A + B$	Add
0	0	1	B	$D = A + B + 1$	Add with carry
0	1	0	$\overline{B}$	$D = A + \overline{B}$	Subtract with borrow
0	1	1	$\overline{B}$	$D = A + \overline{B} + 1$	Subtract
1	0	0	0	$D = A$	Transfer A
1	0	1	0	$D = A + 1$	Increment A
1	1	0	1	$D = A - 1$	Decrement A
1	1	1	1	$D = A$	Transfer A

Figure 4-10 One stage of logic circuit.



(a) Logic diagram

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

(b) Functional table