

Computer Organization & Architecture (3140707)

Prepared By : Karuna Patel
Computer Engineering Department(CKP)

Computer Data Representation

Outlines:

- Basic computer data types
- Complements
- Fixed point representation
- Register Transfer and Micro-operations
- Floating point representation
- Register Transfer language
- Register Transfer
- Bus and Memory Transfers (Tree-State Bus Buffers, Memory Transfer)
- Arithmetic Micro-Operations
- Logic Micro Operations
- Shift Micro-Operations
- Arithmetic logical shift unit

Basic computer data types

- The term data refers to factual information used for analysis or reasoning.
- Binary information in digital computers is stored in memory or processor registers.

Number Systems:

- Radix: A number system of base or radix r is a system that uses distinct symbols for r digits. Numbers are represented by a string of digit symbols.
- To determine the quantity that the number represents it is necessary to multiply each digit by an integer power of r and then form the sum of all weighted digits.

- For example the **decimal number system** in everyday use employs the radix 10 system. The 10 symbols are 0, 1, 2, 3, 4, 5, 6, 7, 8, and 9
- The string of digits 724.5 is interpreted to represent the quantity

$$7*10^2+2*10^1+4*10^0+5*10^{-1}$$

that is 7 hundreds, plus 2 tens, plus 4 units, plus 5 tenths.

- Every decimal number can be similarly interpreted to find the quantity it represents.

The **binary number system** uses the radix 2.

The two digit symbols used are 0 and 1.

The string of digits 101101 is interpreted to represent the quantity

$$1*2^5+0*2^4+1*2^3+1*2^2+0*2^1+1*2^0=45$$

- To distinguish between different radix numbers the digits will be enclosed in parentheses and the radix of the number inserted as a subscript.
- For example to show the equality between decimal and binary forty-five we will write $(101101)_2 = (45)_{10}$.

octal (radix 8) and hexadecimal (radix 16):

- The eight symbols of the octal system are 0, 1, 2, 3, 4, 5, 6, and 7.
- The 16 symbols of the hexadecimal system are 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, and F.

Octal number representation:

$$\begin{aligned}(736.4) &= 7 \times 8^2 + 3 \times 8^1 + 6 \times 8^0 + 4 \times 8^{-1} \\ &= 7 \times 64 + 3 \times 8 + 6 \times 1 + 4/8 \\ &= (478.5)_{10}\end{aligned}$$

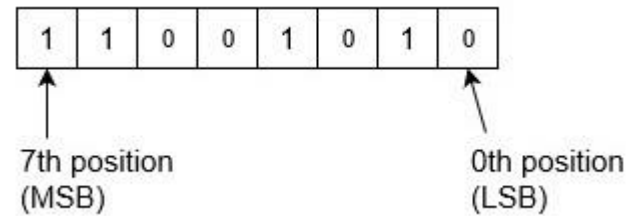
Number System Relationship

$$(F3)_{16} = F \times 16 + 3 = 15 \times 16 + 3 = (243)_{10}$$

HEXADECIMAL	DECIMAL	OCTAL	BINARY
0	0	0	0000
1	1	1	0001
2	2	2	0010
3	3	3	0011
4	4	4	0100
5	5	5	0101
6	6	6	0110
7	7	7	0111
8	8	10	1000
9	9	11	1001
A	10	12	1010
B	11	13	1011
C	12	14	1100
D	13	15	1101
E	14	16	1110
F	15	17	1111

Conversion : Binary to Decimal

1 11001010



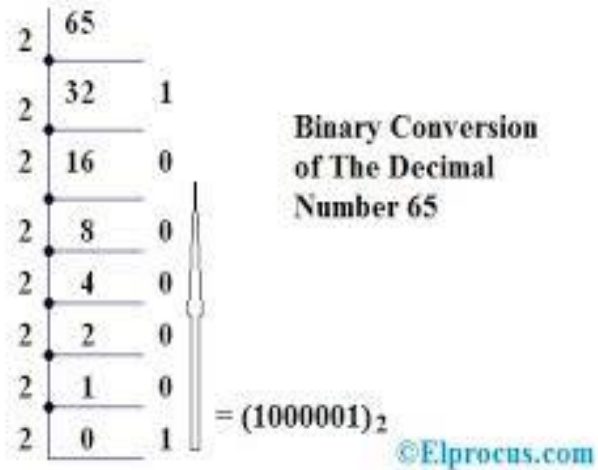
$$\begin{aligned} 11001010 &= 1*2^7 + 1*2^6 + 0*2^5 + 0*2^4 + 1*2^3 + 0*2^2 + 1*2^1 + 0*2^0 \\ &= 128 + 64 + 0 + 0 + 8 + 0 + 2 + 0 \\ &= (202)_{10} \end{aligned}$$

2 (1010.1011)₂ = $1*2^3 + 0*2^2 + 1*2^1 + 0*2^0 + 1*2^{-1} + 0*2^{-2} + 1*2^{-3} + 1*2^{-4}$

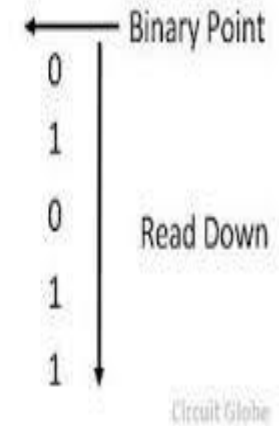
$$\begin{aligned} &= 8 + 0 + 2 + 0 + 0.5 + 0 + 0.125 + 0.0625 \\ &= (10.6875)_{10} \end{aligned}$$

Decimal to binary:

1. 65=

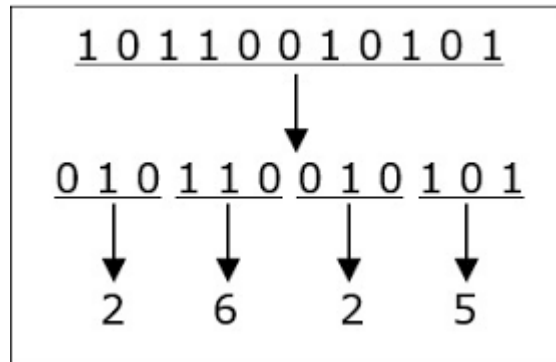


$0.35 \times 2 = 0.70$	with a carry of
$0.70 \times 2 = 0.40$	with a carry of
$0.40 \times 2 = 0.80$	with a carry of
$0.80 \times 2 = 0.60$	with a carry of
$0.60 \times 2 = 0.20$	with a carry of

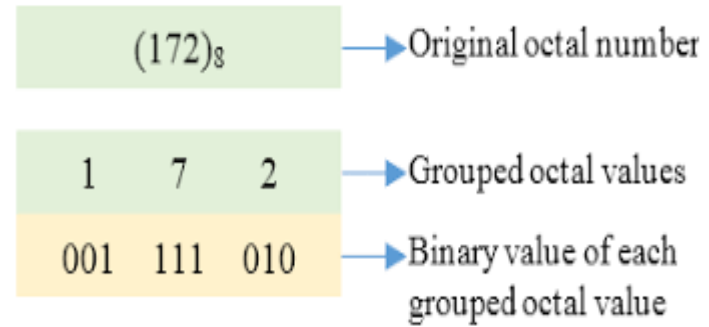


$.95 \times 2 = 1.90$	-----	1 is taken out
$.90 \times 2 = 1.80$	-----	1
$.80 \times 2 = 1.60$	-----	1
$.60 \times 2 = 1.20$	-----	1
$.20 \times 2 = .40$	-----	1
$.40 \times 2 = .80$	-----	0

Binary to octal conversion: 2 → 8



Octal to Binary conversion: 8 → 2



Binary to Hexadecimal: 2 → 16

To convert binary numbers into hexadecimals, you only have to make 4-bit groups and convert directly each group:

<u>1011</u>	<u>0011</u>	<u>0101</u>	(binary)
↓	↓	↓	
B	3	5	(hex)

Hexadecimal to Binary: 16 → 2

Converting Hex to Binary

3AB2			
↙	↓	↘	↘
0011	1010	1011	0010

$3AB2_{16} = 11101010110010_2$

ASCII: ASCII (American Standard Code for Information Interchange)

- which uses seven bits to code 128 characters. The binary code for the uppercase letters, the decimal digits.
- Binary codes play an important part in digital computer operations.
- The codes must be in binary because registers can only hold binary information.

Complements:

- Complements are used in digital computers for simplifying the subtraction operation and for logical manipulation.
- There are two types of complements for each base r system:
 1. r 's complement and
 2. $(r - 1)$'s complement.

- When the value of the base r is substituted in the name, the two types are referred to as the 2's and 1's complement for binary numbers and the 10's and 9's complement for decimal numbers.

1. **($r - 1$)'s Complement:** For decimal numbers $r = 10$ and $r - 1 = 9$, so the 9's

- complement of N is $(10' - 1) - N$.

9's complement

- The 9's complement of a decimal number is obtained by subtracting each digit from 9.
-
- The 9's complement of 546700 is:
 - $999999 - 546700 = 453299$.
- The 9's complement of 012398 is:
 - $999999 - 012398 = 987601$.

10's complement:

1. 623

Handwritten calculation showing the 10's complement of 623. It starts with the 9's complement calculation: 999 minus 623 equals 376. Then it states that 376 is the 9's complement. Finally, it calculates the 10's complement by adding 1 to the 9's complement: 376 + 1 equals 377.

$$\begin{array}{r} 999 \\ - 623 \\ \hline 376 \end{array}$$

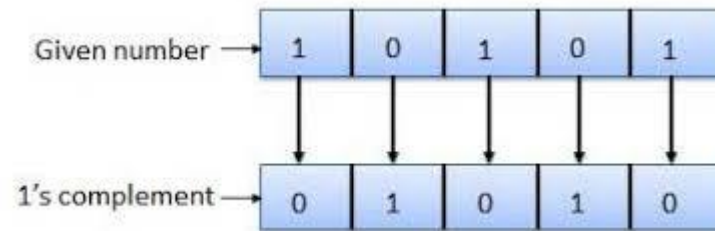
376 is the 9's complement

$$\begin{array}{l} 10's \text{ complement} = 9's \text{ complement} + 1 \\ = 376 + 1 \\ = \underline{377} \end{array}$$

SHOT ON MI 11
MI DUAL CAMERA

2's complement:

- 1's complement
- 2's complement



0 0 0 1 0 1 0 0 → Binary number

1 1 1 0 1 0 1 1 → One's complement

1 1 1 0 1 0 1 1
+ 1
1 1 1 0 1 1 0 0

→ 2s complement

Binary representation of 5 is: 0 1 0 1

1's Complement of 5 is: 1 0 1 0

2's Complement of 5 is: (1's Complement + 1) i.e.

1 0 1 0	(1's Complement)
+ 1	
<u>1 0 1 1</u>	(2's Complement i.e. -5)

Subtraction of Unsigned Numbers:

- The subtraction of two n-digit unsigned numbers $M - N$ ($N \neq 0$) in base r can be done as follows:

1. Add the minuend M to the r 's complement of the subtrahend N .

This performs $M + (r' - N) = M - N + r'$.

2. If $M \geq N$, the sum will produce an end carry r' which is discarded, and what is left is the result $M - N$.

3. If $M < N$, the sum does not produce an end carry and is equal to $r' - (N - M)$, which is the r 's complement of $(N - M)$. To obtain the answer in a familiar form, take the r 's complement of the sum and place a negative sign in front.

- Consider, for example, the subtraction $72532 - 13250 = 59282$. The 10's complement of 13250 is 86750. Therefore:

$$\begin{array}{r}
 M = 72532 \\
 10\text{'s complement of } N = +86750 \\
 \text{Sum} = \underline{159282} \\
 \text{Discard end carry } 10^5 = -100000 \\
 \text{Answer} = \underline{59282}
 \end{array}$$

- Now consider an example with $M < N$. The subtraction $13250 - 72532$ produces negative 59282. Using the procedure with complements.

$$\begin{array}{r}
 M = 13250 \\
 10\text{'s complement of } N = +27468 \\
 \hline
 \text{Sum} = 40718
 \end{array}$$

Answer is negative $59282 = 10\text{'s complement of } 40718$

- Using the two binary numbers $X = 1010100$ and $Y = 1000011$, we perform the subtraction $X - Y$ and $Y - X$ using 2's complements:

$$\begin{array}{r}
 X = 1010100 \\
 2\text{'s complement of } Y = +0111101 \\
 \hline
 \text{Sum} = 10010001 \\
 \text{Discard end carry } 2^7 = -10000000 \\
 \hline
 \text{Answer: } X - Y = 0010001
 \end{array}$$

$$\begin{array}{r}
 Y = 1000011 \\
 2\text{'s complement of } X = +0101100 \\
 \hline
 \text{Sum} = 1101111
 \end{array}$$

- There is no end carry Sum = 1101111
- Answer is negative 0010001 = 2's complement of 110111
- Examples: using the 1o's complement

1.6428-3409 2. 125-1800 3.2043-6152 4.1631-745

Fixed-Point Representation

Positive integers, including zero, can be represented as unsigned numbers.

- The convention is to make the sign bit equal to 0 for positive and to 1 for negative.
- There are two ways of specifying the position of the binary point in a register: by giving it a fixed position or by employing a floating-point representation.

- 1.fixed point

- 2.floating point representation

- There is fixed number of digits after decimal part.
- When the number is negative the sign is represented by 1 but the rest of the number may be represented in one of three possible ways:
 1. Signed-magnitude representation
 2. Signed-1' s complement representation
 3. Signed 2' s complement representation
- The signed-magnitude representation of a negative number consists of the magnitude and a negative sign.
- In the other two representations, the negative number is represented in either the 1's or 2's complement of its positive value.

1. Unsigned fixed point number:

0110110 using 4 integer bits and 3 fractional.

Sign part → consider MSB digit(i.e. 0 here)

$$\begin{aligned}(0110.110)_2 &= 0 \cdot 2^3 + 1 \cdot 2^2 + 1 \cdot 2^1 + 0 \cdot 2^0 + 1 \cdot 2^{-1} + 1 \cdot 2^{-2} + 0 \cdot 2^{-3} \\ &= 4 + 2 + 0.5 + 0.25 \\ &= (6.75)_{10}\end{aligned}$$

2. Signed fixed point number:

i> 2's complement

li> sign and magnitude notation

- Example: represent number $(-7.5)_{10}$ using 8-bit binary representation with 4 digit as integer and 4 fraction bit.

Solution: $(7.5)_{10} = (111.1)_2$

$$= (0111.1000)_2$$

2's complement = 1's complement + 1

$$\begin{array}{r}
 1000.0111 \\
 + \quad 1 \\
 \hline
 1000.1000
 \end{array}$$

So $(-7.5)_{10} = (1000.1000)_2$

Example: **Arithmetic** fixed point compute $0.75 + (-0.625)$ using fixed point numbers.

Solution: step-1: $0.75 = 0000.75$

$= 0000.1100$ // $0.75 * 2 = 1.5$ 1

// $0.5 * 2 = 1.0$ 1

// $0.0 * 2 = 0$ 0

Step-2 : $(0.625) = 0000.625$

// $0.625 * 2 = 1.1$ 1


$= 0000.1010$

// $0.1 * 2 = 0.2$ 0

$-(0.625) = 2\text{'s complement} = 1111.0110$

Step-3: 0000.1100

1111.0110

 10000.0010 --- \rightarrow carry=1 discard it

- Note that negative numbers must initially be in 2's complement and that if the sum obtained after the addition is negative, it is in 2's complement form.

+6	00000110	-6	11111010
+13	00001101	+13	00001101
+19	00010011	+7	00000111
<hr/>			
+6	00000110	-6	11111010
-13	11110011	-13	11110011
-7	11111001	-19	11101101

- In each of the four cases, the operation performed is always addition including the sign bits. Any carry out of the sign bit position is discarded, and negative results are automatically in 2's complement form.

- Arithmetic Subtraction:

1. Take 2's complement of subtrahend

2. Add 2's complement of subtrahend to minuend.

3. If a carry is produced then discard carry and the result is positive . If no carry is produced the result is negative and it is 2's complement form.

Example: 7-5 using 2's complement

Solution: binary of 7 = (0111)₂

binary of 5 = (0101)₂

Step-1: 2's complement of subtrahend (0101)₂

1010 \rightarrow 1's complement

+ 1

1011 \rightarrow 2's complement

Step-2: perform addition of minuend and 2's complement of subtrahend.

0 1 1 1 \rightarrow (7)

1 0 1 1 \rightarrow (-5) (i.e. 2's complement of 5)

1 0 0 1 0

 Carry discard it.

Step-3: Since carry is produced the result is positive.

Final result=(0010)₂

=(+2)₁₀

2. 5-7

Binary of 5 = 0101

Binary of 7 = 0111

Step-1: 2's complement of subtrahend(0111)₂

= 0 1 1 1 → (7)

= 1 0 0 0 → 1's complement

= + 1

1 0 0 1 → 2's complement

Step-2: Addition of minuend and 2's complement of subtrahend

$$\begin{array}{r} 0101 \\ + 1001 \\ \hline 1110 \end{array}$$

Step-3: Since carry is not produced the result is negative and result is in 2's complement form.

To get exact magnitude of result perform 2's complement again.

$$\begin{array}{r} 0001 \rightarrow 1's \text{ complement} \\ + 1 \\ \hline 0010 \rightarrow 2's \text{ complement} \\ = (-2)_{10} \end{array}$$

2. (-6) - (- 13)

Over flow: When two numbers of n digits each are added and the sum occupies $n + 1$ digits, we say that an overflow occurred.

- An overflow is a problem in digital computers because the width of registers is finite. A result that contains $n + 1$ bits cannot be accommodated in a register with a standard length of n bits. For this reason, many computers detect the occurrence of an overflow.

$$\begin{array}{r} \text{carries: } 0 \quad 1 \\ +70 \quad 0 \quad 1000110 \\ +80 \quad 0 \quad 1010000 \\ \hline +150 \quad 1 \quad 0010110 \end{array}$$

$$\begin{array}{r} \text{carries: } 1 \quad 0 \\ -70 \quad 1 \quad 0111010 \\ -80 \quad 1 \quad 0110000 \\ \hline -150 \quad 0 \quad 1101010 \end{array}$$

- **Decimal Fixed-Point Representation** : The representation of decimal numbers in registers is a function of the binary code used to represent a decimal digit.
- A 4-bit decimal code requires four flip-flops for each decimal digit.
- The representation of 4385 in BCD requires 16 flip-flops, four flip-flops for each digit.
- The number will be represented in a register with 16 flip-flops as follows:
0100 0011 1000 0101
- The representation of signed decimal numbers in BCD is similar to the representation of signed numbers in binary.

- Consider the addition $(+375) + (-240) = +135$ done in the signed-10's complement system.

$$\begin{array}{rcl}
 0\ 375 & (0000\ 0011\ 0111\ 0101)_{\text{BCD}} & \\
 +9\ 760 & (1001\ 0111\ 0110\ 0000)_{\text{BCD}} & \\
 \hline
 0\ 135 & (0000\ 0001\ 0011\ 0101)_{\text{BCD}} &
 \end{array}$$

Floating-Point Representation : The floating-point representation of a number has two parts.

- The first part represents a signed, fixed-point number called the mantissa.

- The second part designates the position of the decimal (or binary) point and is called the exponent.
- The fixed-point mantissa may be a fraction or an integer.
- Example: + 6132.789

<i>Fraction</i>	<i>Exponent</i>
+0.6132789	+04

- This representation is equivalent to the scientific notation +0. 6132789 X 10^4.
- Floating-point is always interpreted to represent a number in the following form:

$$m \times r^e$$

- Only the mantissa m and the exponent e are physically represented in the register (including their signs).
- The radix r and the radix-point position of the mantissa are always assumed.
- A floating-point binary number is represented in a similar manner except that it uses base 2 for the exponent.
- Example: + 1001 . 1 1 with a n 8-bit fraction and 6-bit exponent as follows:

<i>Fraction</i>	<i>Exponent</i>
01001110	000100

- The exponent has the equivalent binary number +4. The floating-point number is equivalent to

$$m \times 2^e = +(.1001110)_2 \times 2^{+4}$$

- A floating-point number is said to be normalized if the most significant digit of the mantissa is nonzero.
- For example the decimal number 350 is normalized but 00035 is not.
- Regardless of where the position of the radix point is assumed to be in the mantissa, the number is normalized only if its leftmost digit is nonzero.
- For example the 8-bit binary number 00011010 is not normalized because of the three leading 0' s.
- The number can be normalized by shifting it three positions to the left and discarding the leading 0's to obtain 11010000.
- The three shifts multiply the number by $2^3 = 8$.

Register Transfer Language (RTL)

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

- Registers + Operations performed on the data stored in them = Digital Module
- Modules are interconnected with common data and control paths to form a digital computer system.

Micro operations: operations executed on data stored in one or more registers.

- For any function of the computer, a sequence of micro operations is used to describe it.
- The result of the operation may be:
 - replace the previous binary information of a register or
 - transferred to another register



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

- The set of registers it contains and their function
- The sequence of micro operations performed on the binary information stored in the registers
- The control that initiates the sequence of micro operations

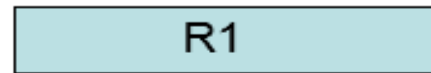
Registers + Micro operations Hardware + Control Functions = Digital Computer

- Register Transfer Language (RTL) : a symbolic notation to describe the micro operation transfers among registers

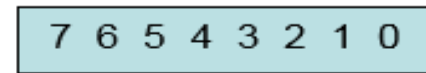
Next steps:

- Define symbols for various types of micro operations
- Describe the hardware that implements these micro operations
- Computer registers are designated by capital letters (sometimes followed by numerals) to denote the function of the register
 - R1: processor register
 - MAR: Memory Address Register (holds an address for a memory unit)
 - PC: Program Counter
 - IR: Instruction Register
 - SR: Status Register

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



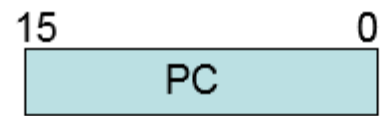
Register R1



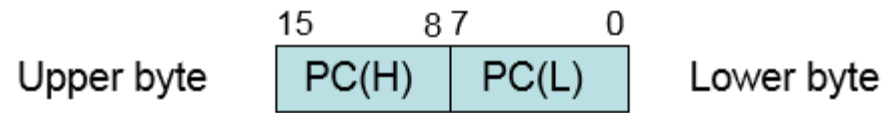
Showing individual bits

A block diagram of a register

Other ways of drawing the block diagram of a register:



Numbering of bits



Upper byte

Lower byte

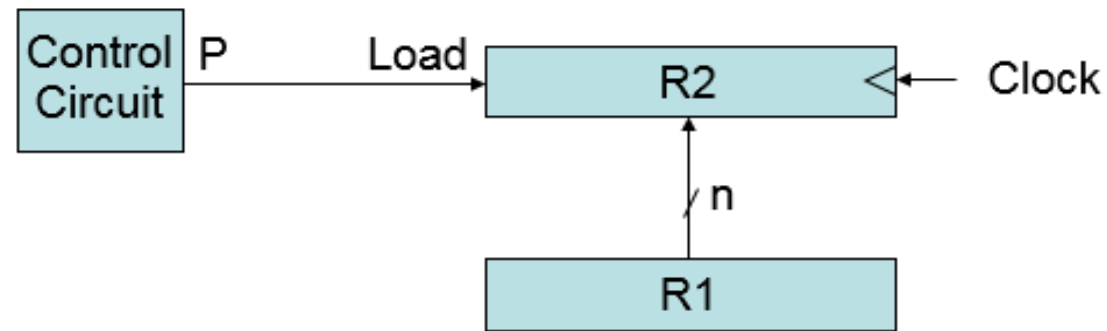
Partitioned into two parts

- Information transfer from one register to another is described by a *replacement operator*: **$R2 \leftarrow R1$**
- This statement denotes a transfer of the content of register R1 into register R2
- The transfer happens in one clock cycle
- The content of the R1 (source) does not change
- The content of the R2 (destination) will be lost and replaced by the new data transferred from R1.
- We are assuming that the circuits are available from the outputs of the source register to the inputs of the destination register, and that the destination register has a parallel load capability

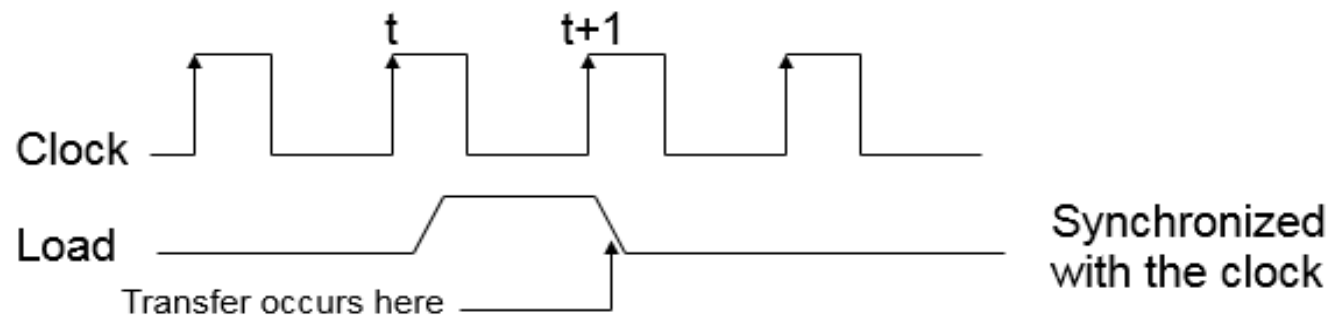
- Conditional transfer occurs only under a control condition
- Representation of a (conditional) transfer
P: $R2 \leftarrow R1$
- A binary condition (P equals to 0 or 1) determines when the transfer occurs
- The content of R1 is transferred into R2 only if P is 1

Hardware implementation of a controlled transfer: $P: R2 \leftarrow R1$

Block diagram:



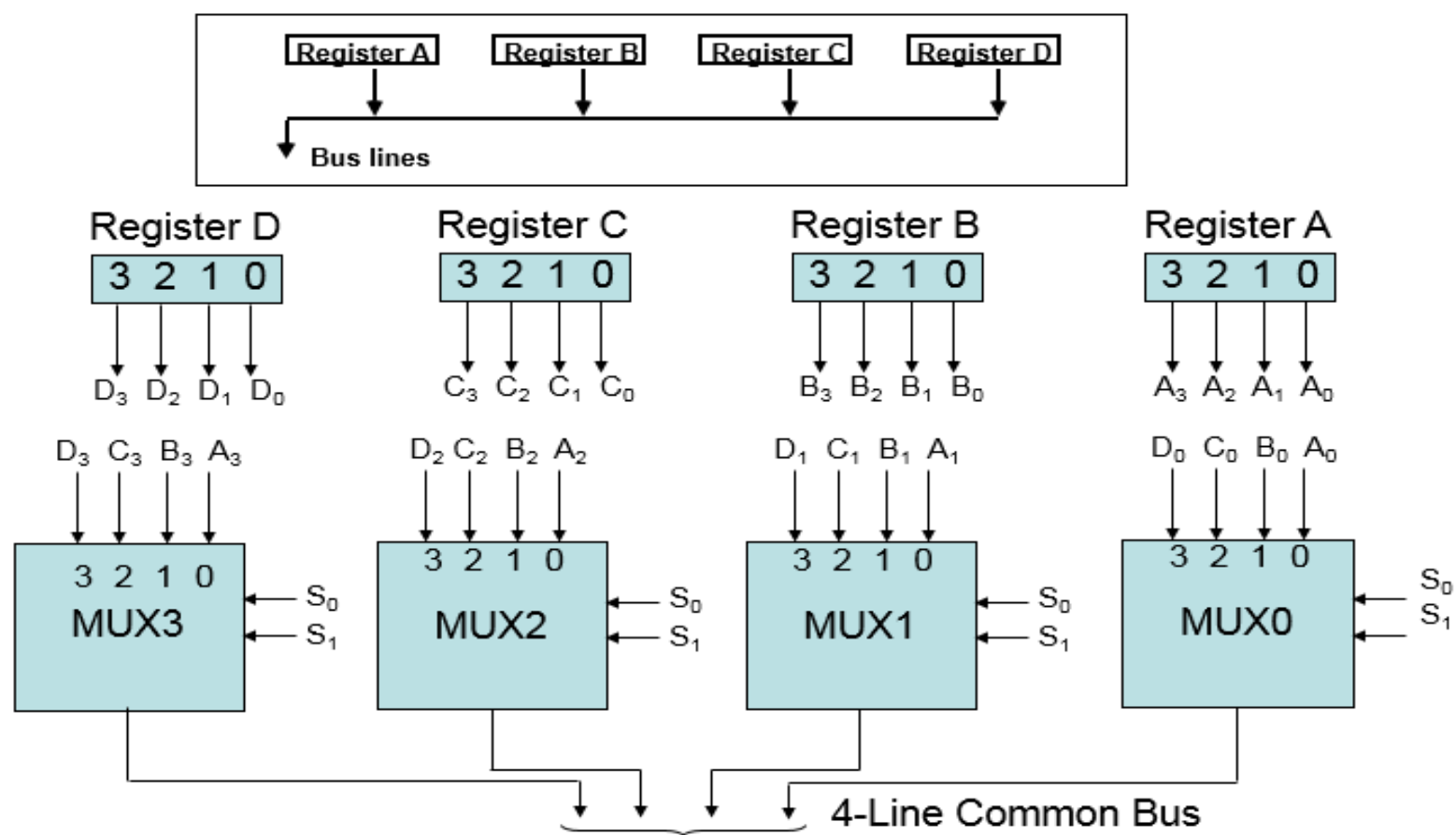
Timing diagram



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

Bus and Memory Transfers

- Paths must be provided to transfer information from one register to another
- A Common Bus System is a scheme for transferring information between registers in a multiple-register configuration.
- A bus: set of common lines, one for each bit of a register through which binary information is transferred one at a time.
- Control signals determine which register is selected by the bus during each particular register transfer.

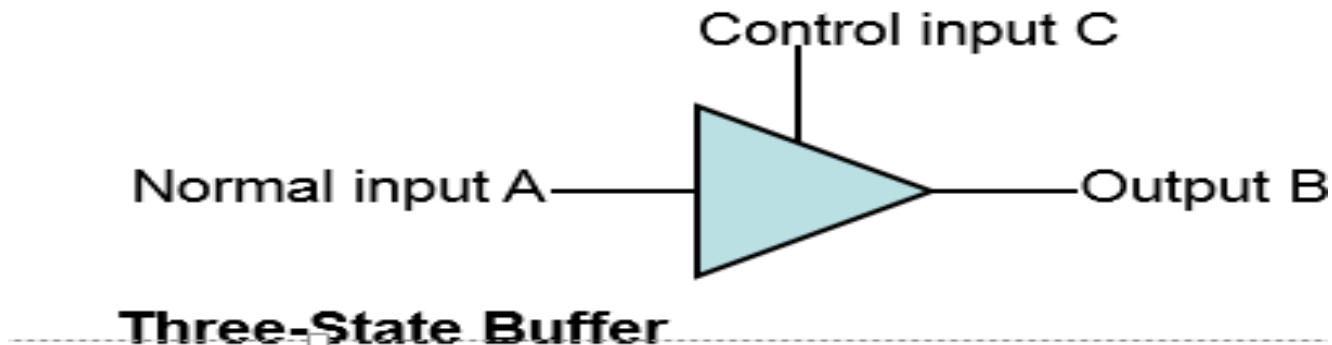


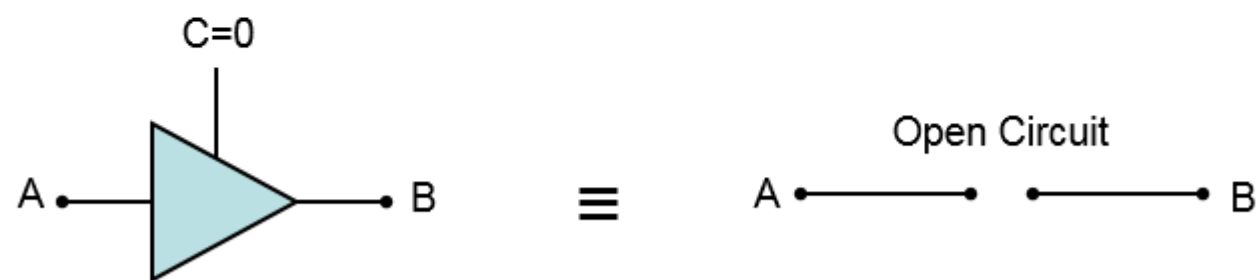
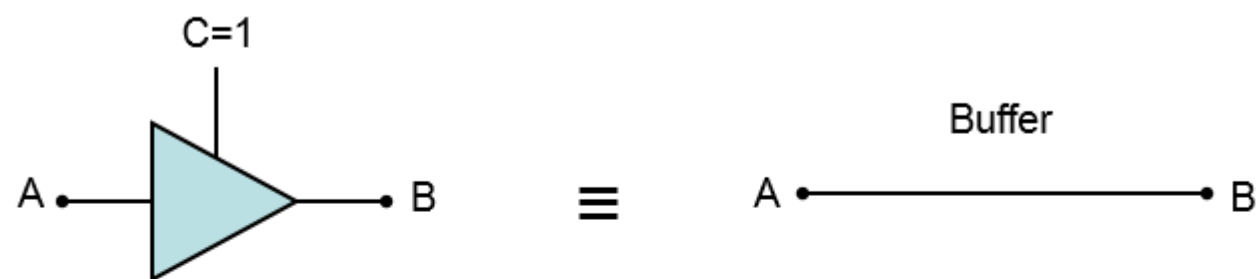
- The transfer of information from a bus into one of many destination registers is done:
 - By connecting the bus lines to the inputs of all destination registers and then:
 - activating the load control of the particular destination register selected
- We write: $R2 \leftarrow C$ to symbolize that the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

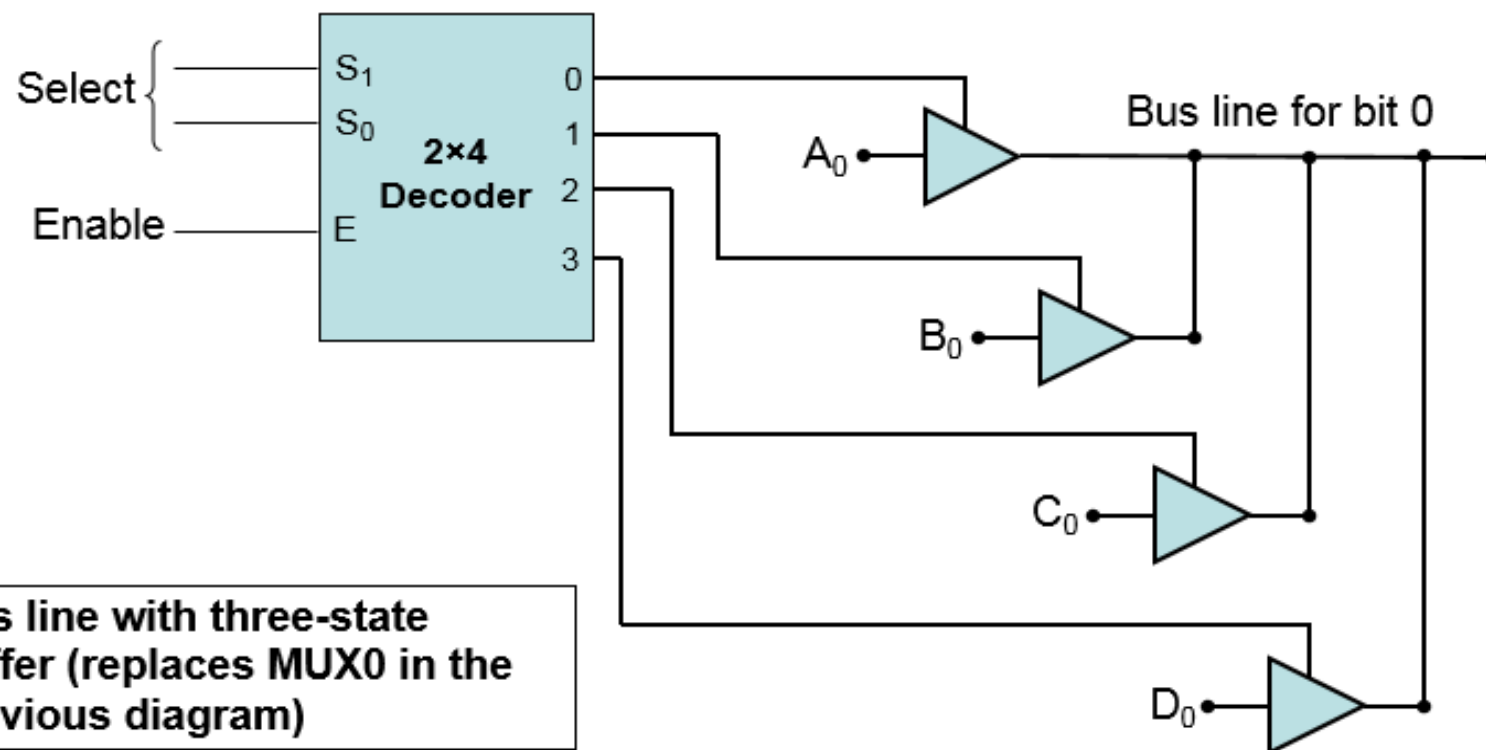
$R2 \leftarrow BUS$ (Load $R2$)

Bus and Memory Transfers: Three-State Bus Buffers

- A bus system can be constructed with three-state buffer gates instead of multiplexers
- A three-state buffer is a digital circuit that exhibits three states: logic-0, logic-1, and high-impedance (Hi-Z)







**Bus line with three-state
buffer (replaces MUX0 in the
previous diagram)**

Memory Transfer

- Memory read : Transfer from memory
- Memory write : Transfer to memory
- Data being read or wrote is called a memory word (called M).
- It is necessary to specify the address of M when writing /reading memory
- This is done by enclosing the address in square brackets following the letter M
- Example: M[0016] : the memory contents at address 0x0016
- Assume that the address of a memory unit is stored in a register called the Address Register AR

Arithmetic Micro operations

- Lets represent a Data Register with DR, then:
- Read: $DR \leftarrow M[AR]$
- Write: $M[AR] \leftarrow DR$

Arithmetic Micro operations:

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

- The basic arithmetic micro operations are: addition, subtraction, increment, decrement, and shift
- Addition Micro operation:

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

- Subtraction Micro operation:

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

$$\mathbf{R3 \leftarrow R1 + R2' + 1}$$

1's complement

One's Complement Micro operation: $R2 \leftarrow \overline{R2}$

Two's Complement Micro operation: $R2 \leftarrow R2+1$

Increment Micro operation: $R2 \leftarrow R2+1$

Decrement Micro operation: $R2 \leftarrow R2-1$

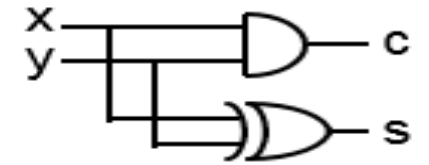
Half Adder/Full Adder

Half Adder

x	y	c	s
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

$$c = xy$$

$$s = xy' + x'y = x \oplus y$$



Full Adder

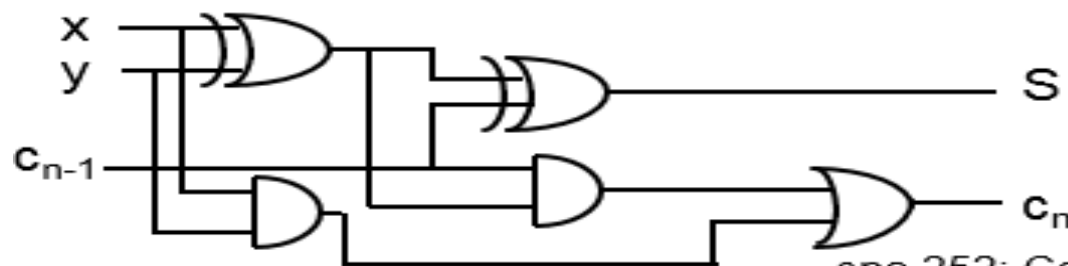
x	y	c_{n-1}	c_n	s
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

			y
	0	0	
	0	1	
x	1	1	c_{n-1}
	0	1	
			c_n

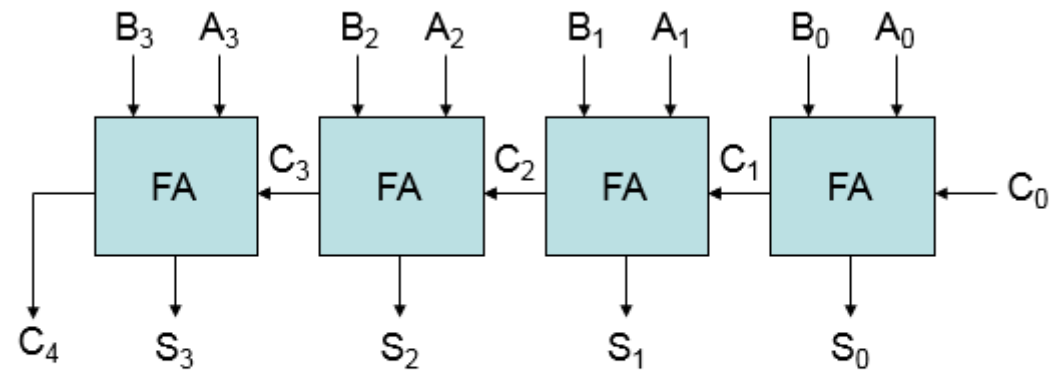
			y
	0	1	
	1	0	
x	0	1	c_{n-1}
	1	0	
			s

$$c_n = xy + xc_{n-1} + yc_{n-1} = xy + (x \oplus y)c_{n-1}$$

$$s = x'y'c_{n-1} + x'yc'_{n-1} + xy'c'_{n-1} + xyc_{n-1} = x \oplus y \oplus c_{n-1} = (x \oplus y) \oplus c_{n-1}$$



4 bit binary Adder



**4-bit binary adder
(connection of FAs)**

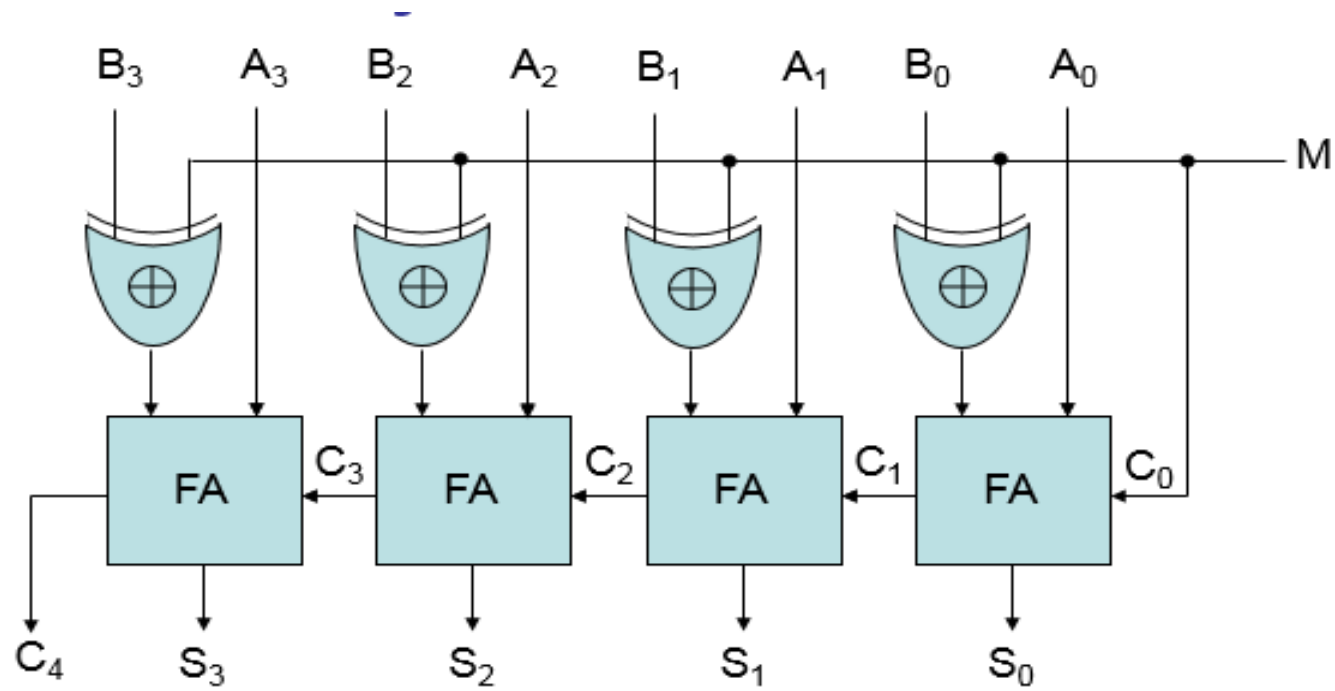
c 1 1 1 1 0

R1 1 1 0 1

R2 + 0 1 1 1

sum 1 0 1 0 0

4 bit binary adder - subtractor



4-bit adder-subtractor

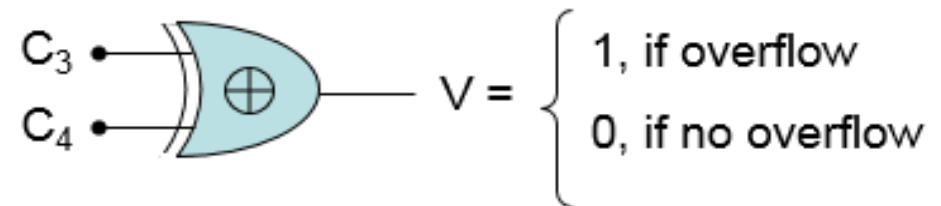
- For unsigned numbers, this gives $A - B$ if $A \geq B$ or the 2's complement of $(B - A)$ if $A < B$

(example: $3 - 5 = -2 = 1110$)

- For signed numbers, the result is $A - B$ provided that there is no overflow.
(example : $-3 - 5 = -8$)

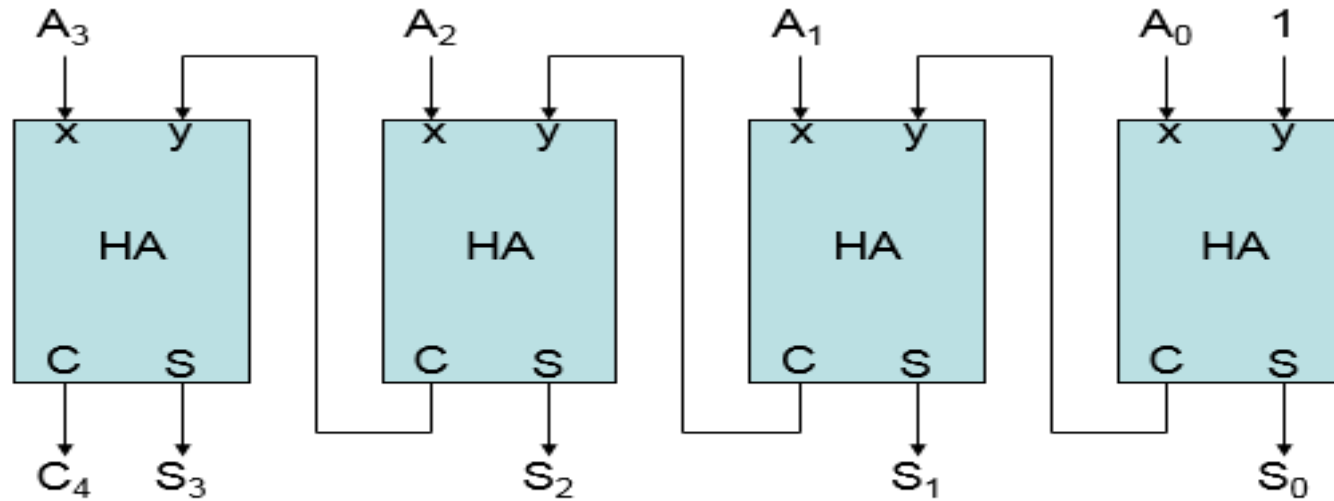
- 1101

$$\begin{array}{r} 1011 + \\ \hline 1000 \end{array}$$



Overflow detector for signed numbers

Arithmetic Microoperations Binary Incrementer



4-bit Binary Incrementer

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

Arithmetic Microoperations Arithmetic Circuit:

- This circuit performs seven distinct arithmetic operations and the basic component of it is the parallel adder
- The output of the binary adder is calculated from the following arithmetic sum:

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

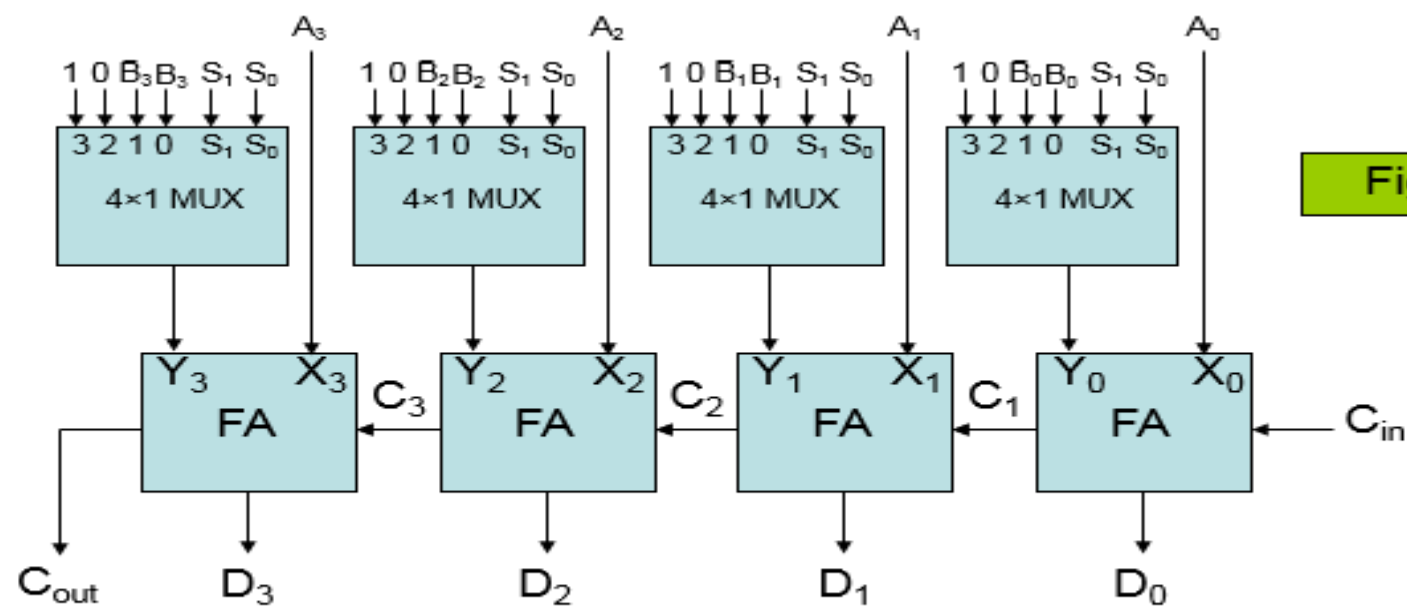


Figure A

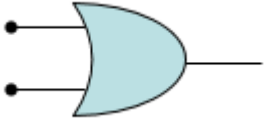
4-bit Arithmetic Circuit

s1	s0	cin	y	D=A+Y+Cin	Micro operation
0	0	0	B	D=A+B	Add
0	0	1	B	D=A+B+1	Add with carry
0	1	0	B'	D=A+B'	Subtract with borrow
0	1	1	B'	D=A+B'+1	subtract
1	0	0	0	D=A	Transfer
1	0	1	0	D=A+1	Increment
1	1	0	1	D=A-1	Decrement
1	1	1	1	D=A	Transfer A

Logic Microoperations

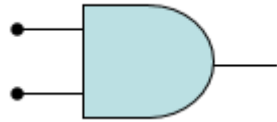
The four basic microoperations

OR Microoperation

- Symbol: \vee , +
- Gate:
A light blue OR gate symbol with two input lines on the left and one output line on the right.
- Example: $100110_2 \vee 1010110_2 = 1110110_2$
- P+Q: $R1 \leftarrow R2 + R3, R4 \leftarrow R5 \vee R6$

AND Microoperation

- Symbol: \wedge
- Gate:

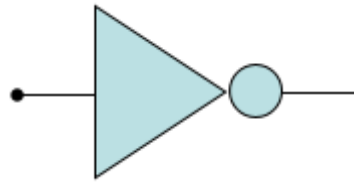


Example: $100110_2 \wedge 1010110_2 = 0000110_2$

Complement (NOT) Microoperation

Symbol: $\overline{}$

Gate:

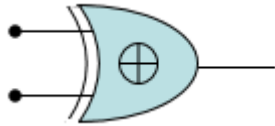


Example: $1010110_2 = 0101001_2$

XOR (Exclusive-OR) Microoperation

- Symbol: \oplus

- Gate:



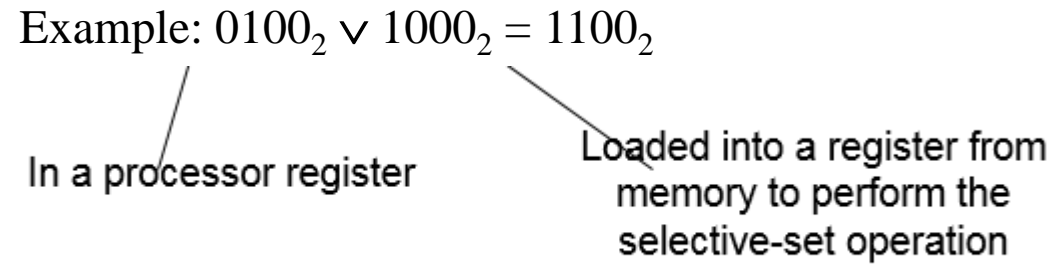
Example: $100110_2 \oplus 1010110_2 = 1110000_2$

Selective-set Operation:

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

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

In a processor register



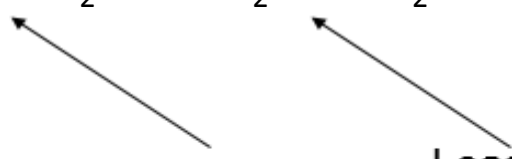
Loaded into a register from
memory to perform the
selective-set operation

Selective-complement (toggling) Operation:

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

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

In a processor register



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

Insert Operation

Step1: mask the desired bits

Step2: OR them with the desired value

Example: suppose $R1 = 0110\ 1010$, and we desire to replace the leftmost 4 bits (0110) with 1001 then:

Step1: $0110\ 1010 \wedge 0000\ 1111$

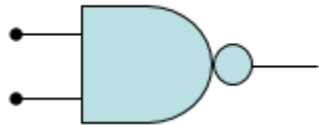
Step2: $0000\ 1010 \vee 1001\ 0000$

→ $R1 = 1001\ 1010$

NAND Microoperation

Symbols: \wedge and \neg

Gate:

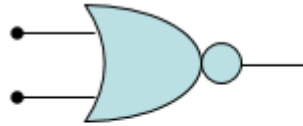


Example: $100110_2 \wedge 1010110_2 = 1111001_2$

NOR Microoperation

- Symbols: \vee and $-$

- Gate:



Example: $100110_2 \vee 1010110_2 = 0001001_2$

Set (Preset) Microoperation:

- Force all bits into 1's by ORing them with a value in which all its bits are being assigned to logic-1
- Example: $100110_2 \vee 111111_2 = 111111_2$

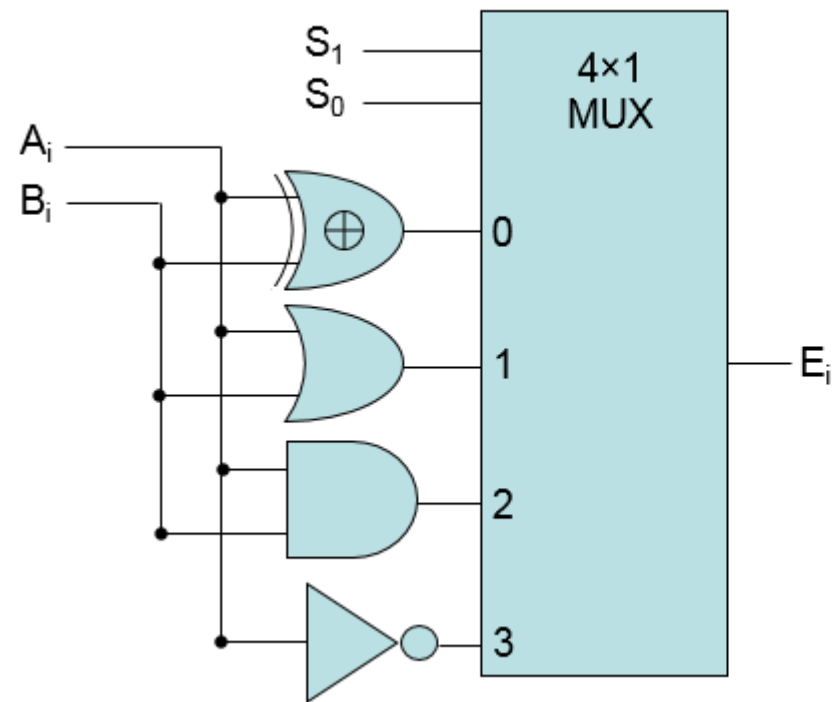
Clear (Reset) Microoperation:

- Force all bits into 0's by ANDing them with a value in which all its bits are being assigned to logic-0
- Example: $100110_2 \wedge 000000_2 = 000000_2$

Logic Microoperations

Hardware Implementation

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

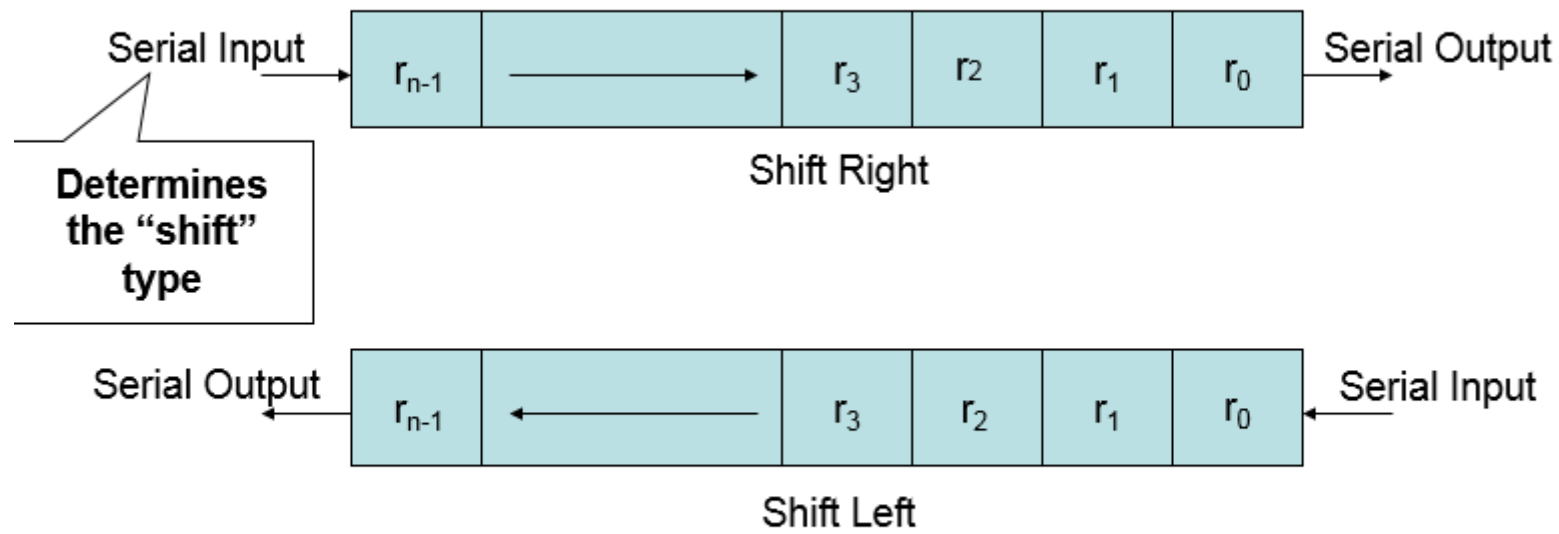


S_1	S_0	Output	Operation
0	0	$E = A \oplus B$	XOR
0	1	$E = A \vee B$	OR
1	0	$E = A \wedge B$	AND
1	1	$E = \neg A$	Complement

This is for one bit i

Shift Microoperations

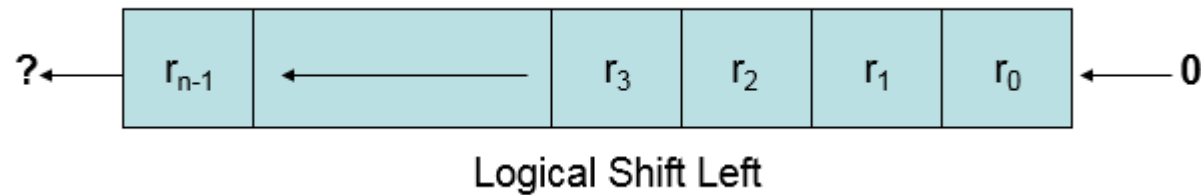
- Used for serial transfer of data
- Also used in conjunction with arithmetic, logic, and other data-processing operations
- The contents of the register can be shifted to the left or to the right
- As being shifted, the first flip-flop receives its binary information from the serial input
- Three types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



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

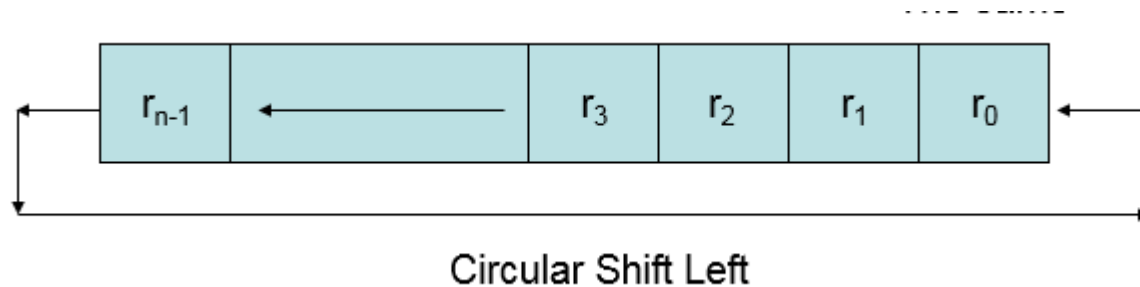
Shift Microoperations: Logical Shifts

- Transfers 0 through the serial input
- Logical Shift Right: $R1 \leftarrow \text{shr } R1$
- Logical Shift Left: $R2 \leftarrow \text{shl } R2$



Shift Microoperations: Circular Shifts (Rotate Operation)

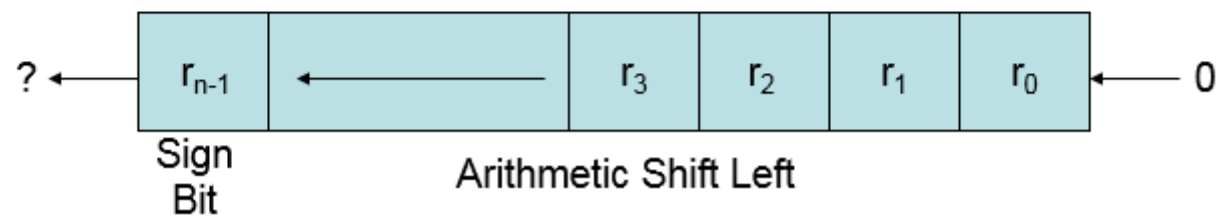
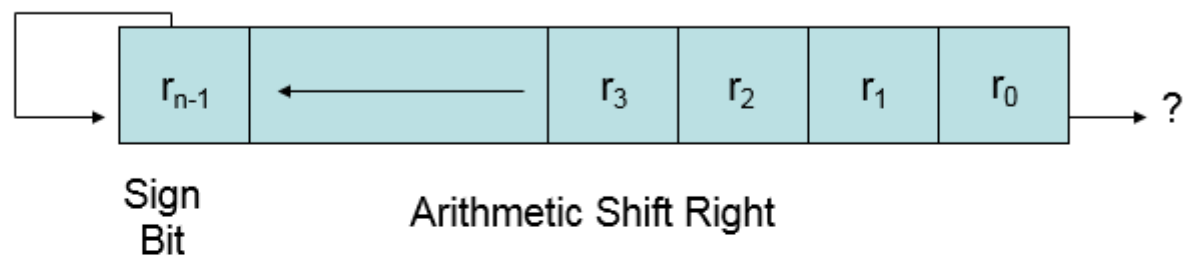
- Circulates the bits of the register around the two ends without loss of information
- Circular Shift Right: $R1 \leftarrow \text{cir } R1$
- Circular Shift Left: $R2 \leftarrow \text{cil } R2$



Shift Microoperations

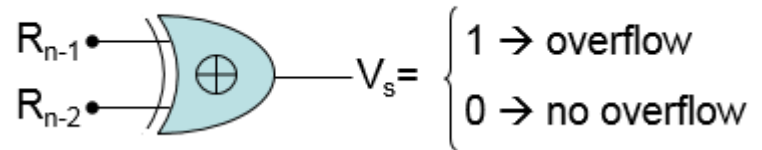
Arithmetic Shifts

- Shifts a signed binary number to the left or right
- An arithmetic shift-left multiplies a signed binary number by 2: ashl (00100):
01000
- An arithmetic shift-right divides the number by 2
 ashr (00100) : 00010
- An overflow may occur in arithmetic shift-left, and occurs when the sign bit is changed (sign reversal)



- An overflow flip-flop V_s can be used to detect an arithmetic shift-left overflow

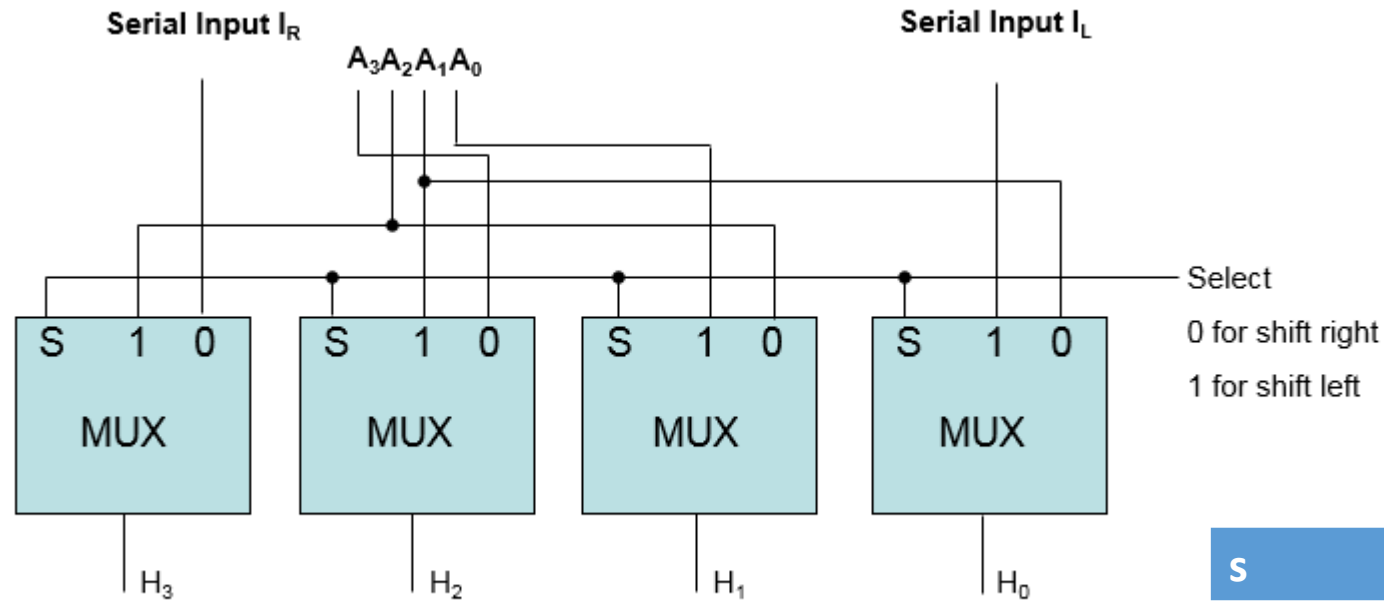
$$V_s = R_{n-1} \oplus R_{n-2}$$



• Example: Assume $R1=11001110$, then:

- Arithmetic shift right once : $R1 = 11100111$
- Arithmetic shift right twice : $R1 = 11110011$
- Arithmetic shift left once : $R1 = 10011100$
- Arithmetic shift left twice : $R1 = 00111000$
- Logical shift right once : $R1 = 01100111$
- Logical shift left once : $R1 = 10011100$
- Circular shift right once : $R1 = 01100111$
- Circular shift left once : $R1 = 10011101$

- A possible choice for a shift unit would be a bidirectional shift register with parallel load. Has drawbacks:
 - Needs two pulses (the clock and the shift signal pulse)
 - Not efficient in a processor unit where multiple number of registers share a common bus
- It is more efficient to implement the shift operation with a combinational circuit.

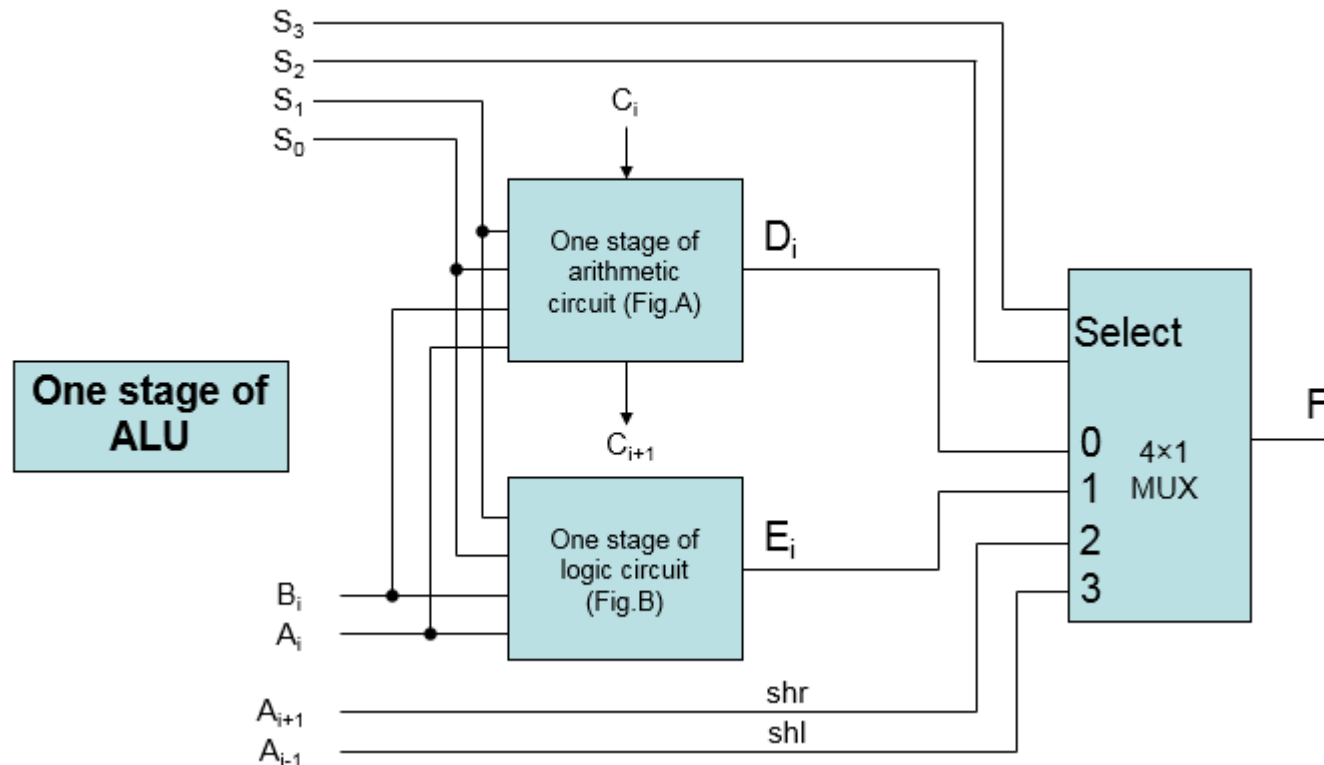


4-bit Combinational Circuit Shifter

s	H0	H1	H2	H3
0	IR	A0	A1	A2
1	A1	A2	A3	IL

Arithmetic Logic Shift Unit

- Instead of having individual registers performing the micro operations directly computer systems employ a number of storage registers connected to a common operational unit called an Arithmetic Logic Unit (**ALU**)



S3	S2	S1	S0	CIN	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	$F=A+B+1$	Addition with carry
0	0	1	0	0	$F=A+B'$	Subtraction with borrow
0	0	1	0	1	$F=A+B'+1$	subtraction
0	0	1	1	0	$F=A-1$	Decrement A