DETAILED CONTENTS

1. Introduction (02 Periods) a) Distinction between analog and digital signal. b) Applications and advantages of digital signals.
2. Number System (03 Periods) a) Binary, octal and hexadecimal number system: conversion from decimal and hexadecimal to binary and vice-versa. b) Binary addition and subtraction including binary points. 1's and 2's complement method of addition/subtraction.
3. Codes and Parity (03 Periods) a) Concept of code, weighted and non-weighted codes, examples of 8421, BCD, excess-3 and Gray code. b) Concept of parity, single and double parity and error detection
4. Logic Gates and Families (05 Periods) a) Concept of negative and positive logic b) Definition, symbols and truth tables of NOT, AND, OR, NAND, NOR, EXOR Gates, NAND and NOR as universal gates. (c) Introduction to TTL and CMOS logic families.
5. Logic Simplification (04 Periods) a) Postulates of Boolean algebra, De Morgan's Theorems. Implementation of Boolean (logic) equation with gates b) Karnaugh map (upto 4 variables) and simple application in developing combinational logic circuits.

6. Arithmetic circuits (02 Periods) a) Half adder and Full adder circuit, design and implementation. b) 4 bit adde circuit
7. Decoders, Multiplexeres, De Multiplexeres and Encoder (04 Periods) a) Four bit decoder circuits for 7 segment display and decoder/driver ICs. b) Basic functions and block diagram of MUX and DEMUX with different ICs c) Basic functions and block diagram of Encoder
8. Latches and flip flops (04 Periods) a) Concept and types of latch with their working and applications b) Operation using waveforms and truth tables of RS, T, D, Master/Slave JK flip flops. c) Difference between a latch and a flip flop
9. Counters (06 Periods) a) Introduction to Asynchronous and Synchronous counters b) Binary counters c) Divide by N ripple counters, Decade counter, Ring counter
10. Shift Register (06 Periods) Introduction and basic concepts including shift left and shift right. a) Serial in parallel out, serial in serial out, parallel in serial out, parallel in parallel out. b) Universal shift register
11. A/D and D/A Converters (06 Periods) - Working principle of A/D and D/A converters - Brief idea about different techniques of A/D conversion and study of : · Stair step Ramp A/D converter · Dual Slope A/D converter · Successive Approximation A/D Converter - Detail study of : · Binary Weighted D/A converter · R/2R ladder D/A converter - Applications of A/D and D/A converter.
12. Semiconductor Memories (03 periods) Memory organization, classification of semiconductor memories (RAM, ROM, PROM, EPROM, EEPROM), static and dynamic RAM, introduction to 74181 ALU IC

LIST OF PRACTICALS

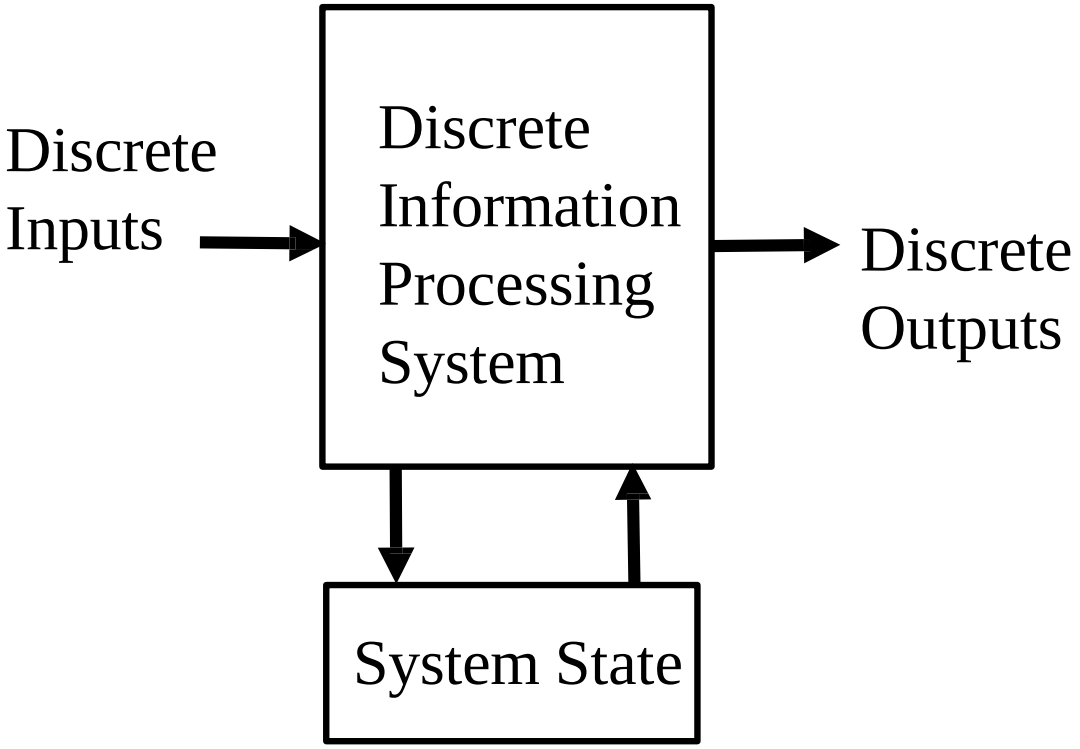
1. Verification and interpretation of truth tables for AND, OR, NOT NAND, NOR and Exclusive OR (EXOR) and Exclusive NOR(EXNOR) gates.
2. Realisation of logic functions with the help of NAND or NOR gates
3. To design a half adder using XOR and NAND gates and verification of its operation - Construction of a full adder circuit using XOR and NAND gates and verify its operation
4. Verification of truth table for positive edge triggered, negative edge triggered, level triggered IC flip-flops (At least one IC each of D latch , D flip-flop, JK flip-flops).
5. Verification of truth table for encoder and decoder ICs, Mux and DeMux
6. To design a 4 bit SISO, SIPO, PISO, PIPO shift registers using JK/D flip flops and verification of their operation.
7. To design a 4 bit ring counter and verify its operation.
8. Use of Asynchronous Counter ICs (7490 or 7493)

Overview

- **Introduction to Digital Systems**
- **Information Representation**
- **Number Systems** [binary, octal and hexadecimal]
- **Arithmetic Operations**
- **Base Conversion**
- **Decimal Codes** [Binary Coded Decimal (BCD)]
- **Gray Codes**
- **Alphanumeric Codes**
- **Parity Bit**

DIGITAL & COMPUTER SYSTEMS

- **Takes a set of discrete information inputs and discrete internal information (system state) and generates a set of discrete information outputs.**



Types of Digital Systems

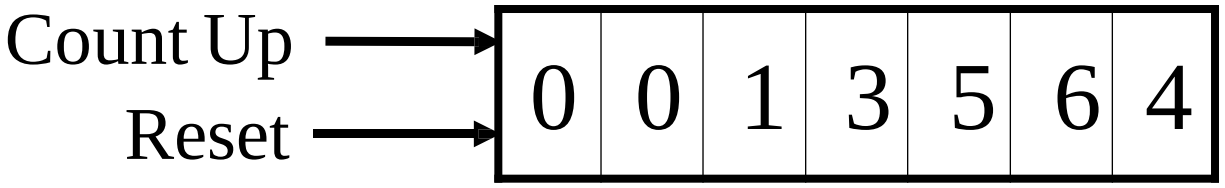
- **No state present**
  - **Combinational Logic System**
  - **Output = Function(Input)**
- **State present**
  - **State updated at discrete times**

    => Synchronous Sequential System
  - **State updated at any time**

    =>Asynchronous Sequential System
  - **State = Function (State, Input)**
  - **Output = Function (State, Input)**

Digital System Example:

A Digital Counter (e. g., odometer):



Inputs: **Count Up, Reset**
Outputs: **Visual Display**
State: **"Value" of stored digits**

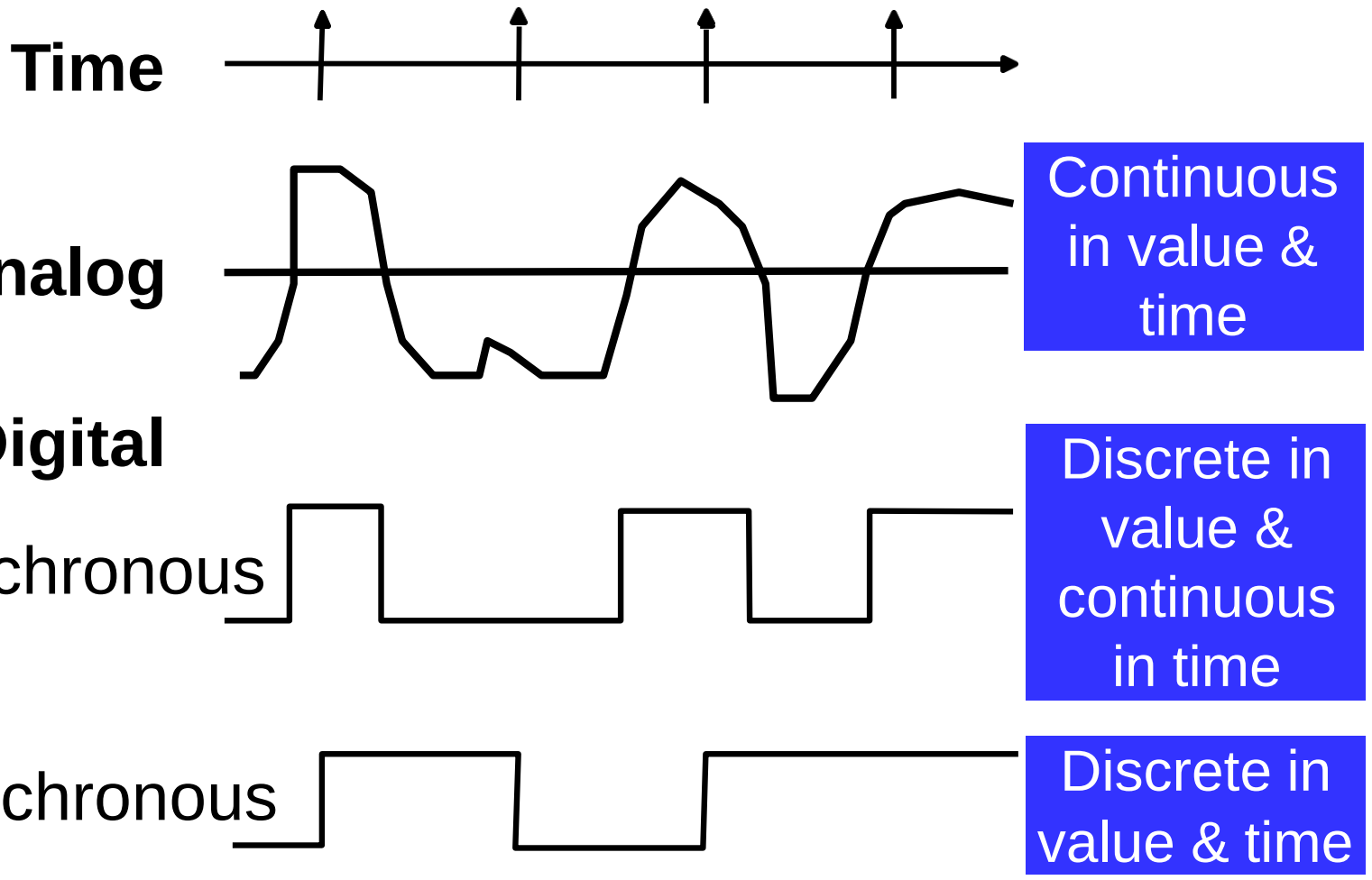
INFORMATION REPRESENTATION - Signals

- **Information variables represented by physical quantities.**
- **For digital systems, the variables take on discrete values.**
- **Two level, or binary values are the most prevalent values in digital systems.**

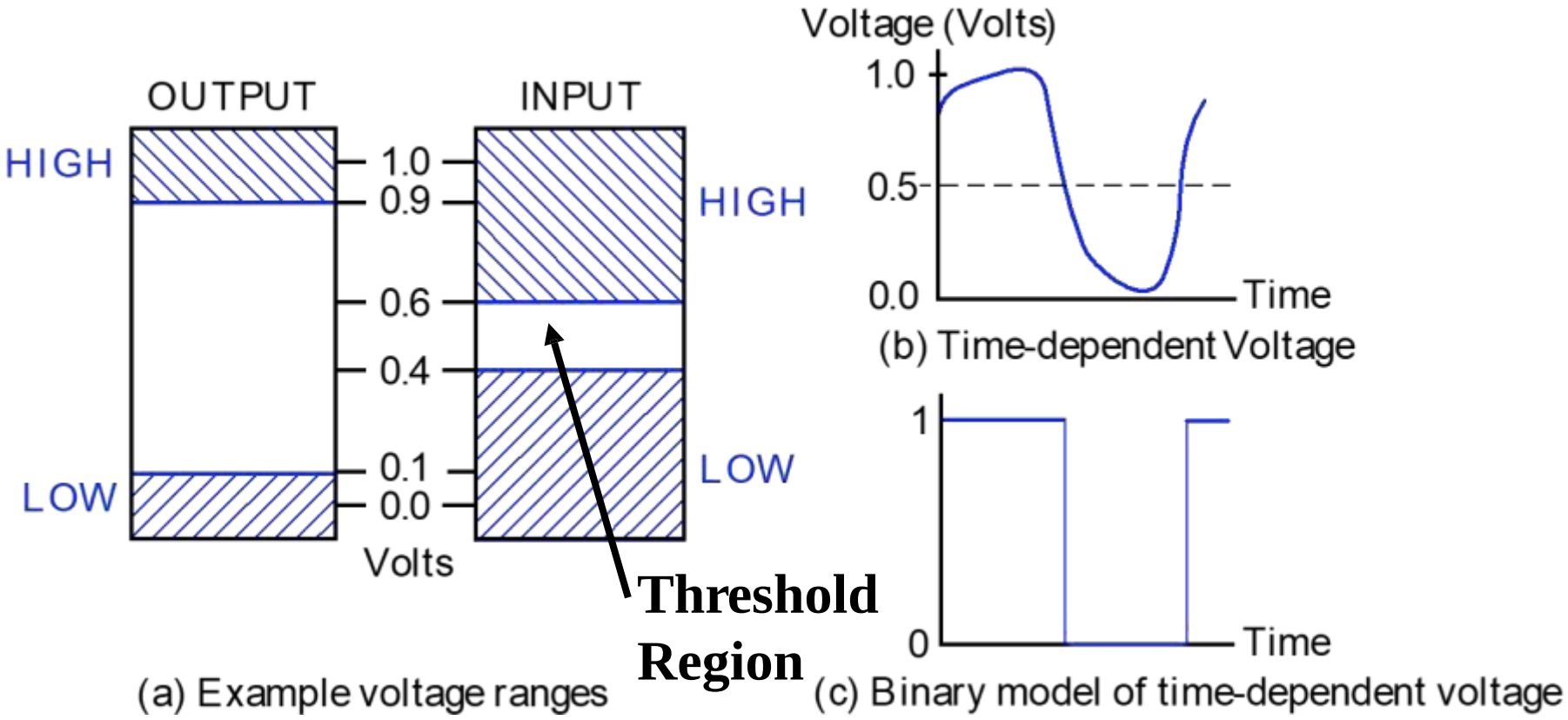
- **Binary values are represented abstractly by:**
  - **digits 0 and 1**
  - **words (symbols) False (F) and True (T)**
  - **words (symbols) Low (L) and High (H)**
  - **and words On and Off.**

- **Binary values are represented by values or ranges of values of physical quantities**

Signal Examples Over Time



Signal Example – Physical Quantity: Voltage



NUMBER SYSTEMS – Representation

- Positive radix, positional number systems
- A number with *radix* r is represented by a string of digits:

$$A_{n-1}A_{n-2}\ldots A_1A_0 . A_{-1}A_{-2}\ldots A_{-m+1}A_{-m}$$

in which $0 \le A_i < r$ and . is the *radix point*.

- The string of digits represents the power series:

$$(\text{Number})_r = \left(\sum_{i=0}^{i=n-1} A_i \cdot r^i\right) + \left(\sum_{j=-m}^{j=-1} A_j \cdot r^j\right)$$

(Integer Portion) + (Fraction Portion)

Number Systems – Examples

		General	Decimal	Binary
Radix (Base)		r	10	2
Digits		0 => r - 1	0 => 9	0 => 1
Powers of Radix	0	r^0	1	1
	1	r^1	10	2
	2	r^2	100	4
	3	r^3	1000	8
	4	r^4	10,000	16
	5	r^5	100,000	32
	-1	r^{-1}	0.1	0.5
	-2	r^{-2}	0.01	0.25
	-3	r^{-3}	0.001	0.125
	-4	r^{-4}	0.0001	0.0625
	-5	r^{-5}	0.00001	0.03125

Special Powers of 2

- 2^{10} (1024) is Kilo, denoted "K"
- 2^{20} (1,048,576) is Mega, denoted "M"
- 2^{30} (1,073, 741,824) is Giga, denoted "G"
- 2^{40} (1,099,511,627,776) is Tera, denoted “T"

ARITHMETIC OPERATIONS - Binary Arithmetic

- **Single Bit Addition with Carry**
- **Multiple Bit Addition**
- **Single Bit Subtraction with Borrow**
- **Multiple Bit Subtraction**
- **Multiplication**
- **BCD Addition**

Natural Numbers

Aha!

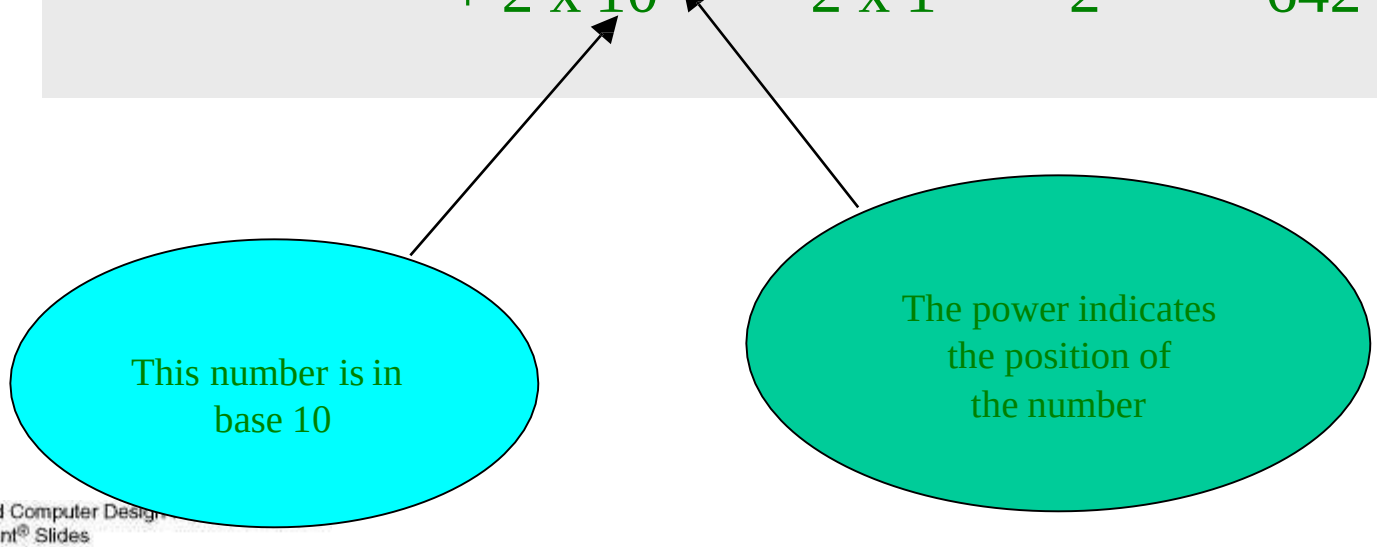
642 is 600 + 40 + 2 in BASE 10

The base of a number determines the number of digits and the value of digit positions

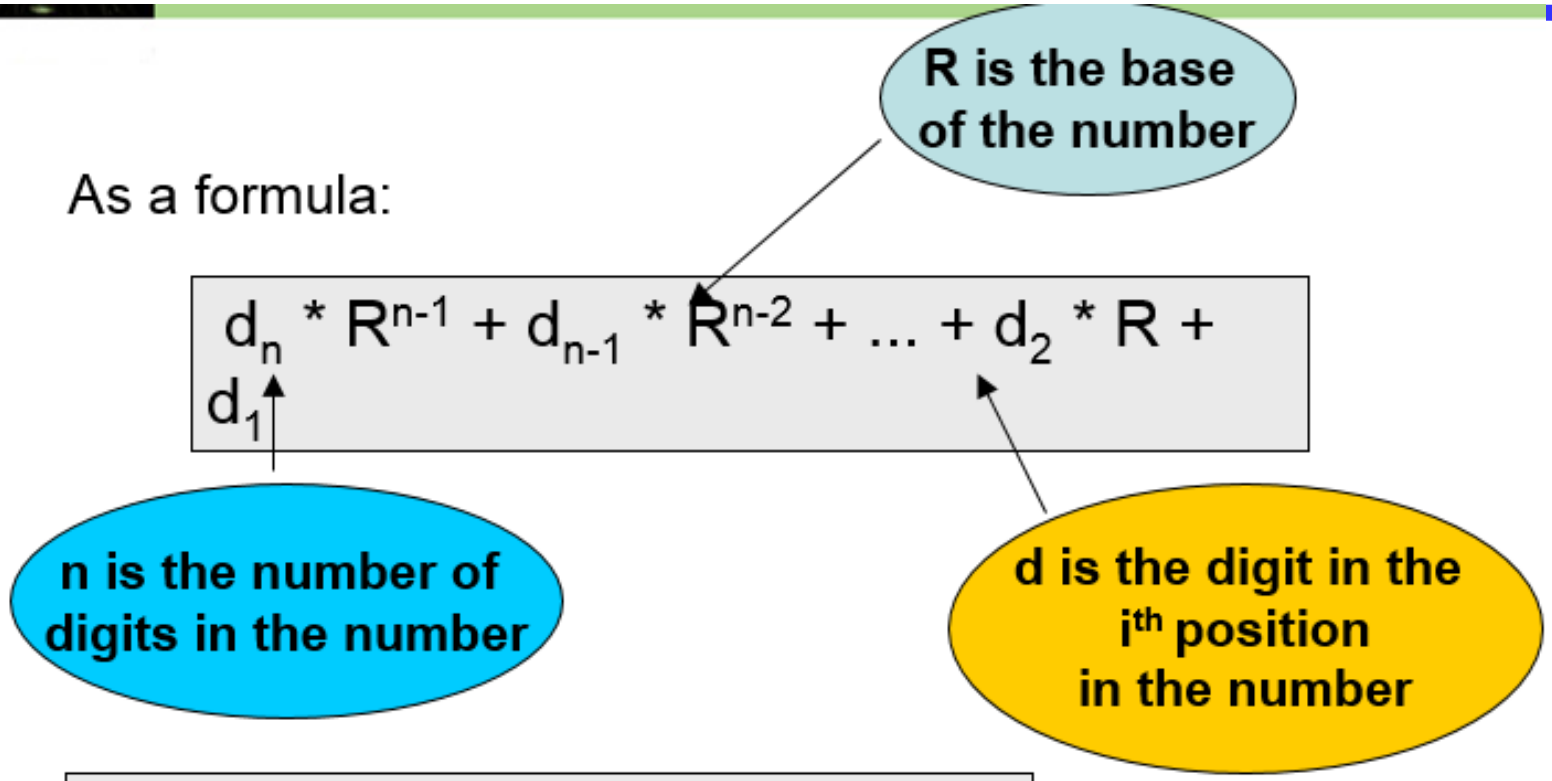
Positional Notation

Continuing with our example…
642 in base 10 *positional notation* is:

$$\begin{aligned} 6 \times 10^2 &= 6 \times 100 &&= 600 \\ +\ 4 \times 10^1 &= 4 \times 10 &&= 40 \\ +\ 2 \times 10^0 &= 2 \times 1 &&= 2 \qquad = 642 \text{ in base } 10 \end{aligned}$$



Positional Notation



Positional Notation

What if 642 has the base of 13?

$$
\begin{aligned}
+ 6 \times 13^2 &= 6 \times 169 &&= 1014 \\
+ 4 \times 13^1 &= 4 \times 13 &&= 52 \\
+ 2 \times 13^0 &= 2 \times 1 &&= 2 \\
&&&= 1068 \text{ in base } 10
\end{aligned}
$$

642 in base 13 is equivalent to 1068 in base 10

Binary

Decimal is base 10 and has 10 digits:
0,1,2,3,4,5,6,7,8,9

Binary is base 2 and has 2 digits:
0,1

For a number to exist in a given number system, the number system must include those digits. For example: The number 284 only exists in base 9 and higher.

Bases Higher than 10

How are digits in bases higher than 10 represented?

Base 16:

0,1,2,3,4,5,6,7,8,9,A,B,C,D,E, and F

Converting Octal to Decimal

What is the decimal equivalent of the octal number 642?

$$
\begin{aligned}
6 \times 8^2 &= 6 \times 64 = 384 \\
+\ 4 \times 8^1 &= 4 \times 8 = 32 \\
+\ 2 \times 8^0 &= 2 \times 1 = 2 \\
&= 418 \text{ in base } 10
\end{aligned}
$$

Converting Hexadecimal to Decimal

What is the decimal equivalent of the hexadecimal number DEF?

$$
\begin{aligned}
D \times 16^2 &= 13 \times 256 = 3328 \\
+\ E \times 16^1 &= 14 \times 16 = 224 \\
+\ F \times 16^0 &= 15 \times 1 = 15 \\
&= 3567 \text{ in base } 10
\end{aligned}
$$

Remember, base 16 is 0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F

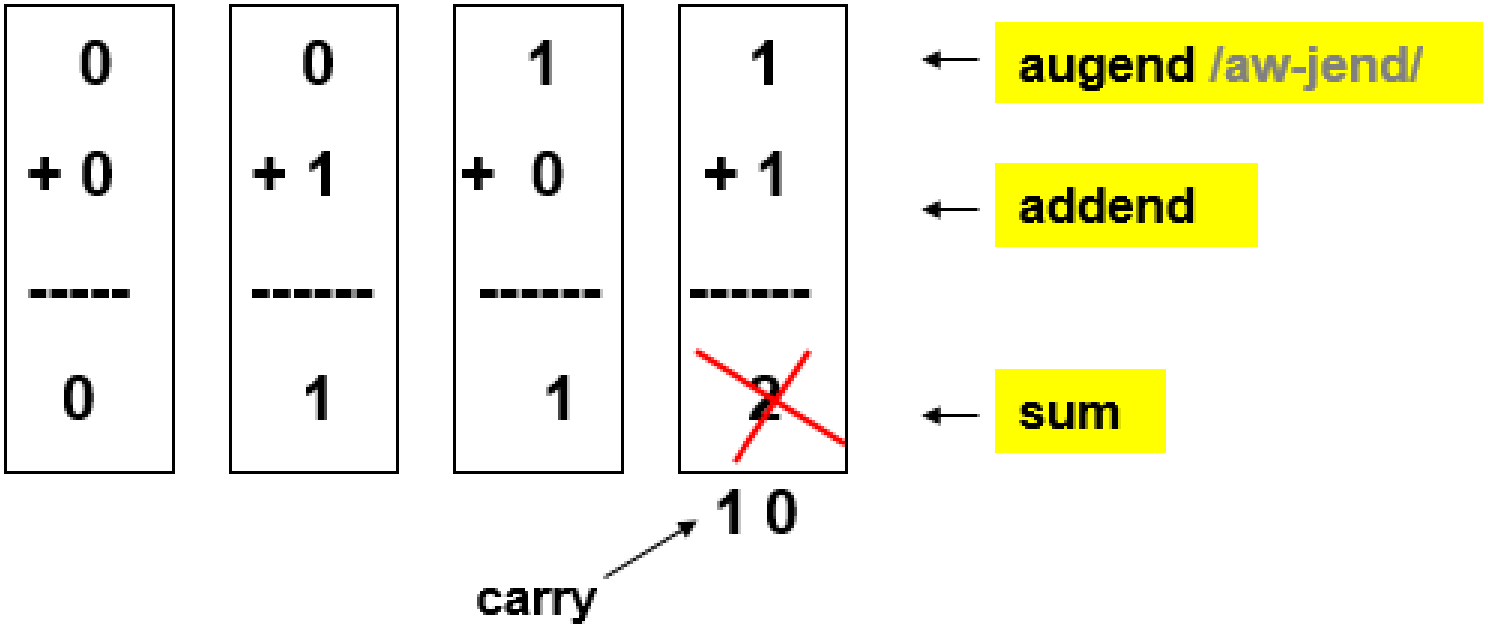
Converting Binary to Decimal

What is the decimal equivalent of the binary number 010110?

$$
\begin{aligned}
& & & 6 & & \\
& 1 \times 2_5 & = & 1 \times 64 & = & 64 \\
+ & 1 \times 2_4 & = & 1 \times 32 & = & 32 \\
+ & 0 \times 2_3 & = & 0 \times 16 & = & 0 \\
+ & 1 \times 2_2 & = & 1 \times 8 & = & 8 \\
+ & 1 \times 2_1 & = & 1 \times 4 & = & 4 \\
+ & 1 \times 2 & = & 1 \times 2 & = & 2 \\
+ & 0 \times 2^0 & = & 0 \times 1 & = & 0 \\
& & & & = & 112 \text{ in base } 10
\end{aligned}
$$

Binary Addition

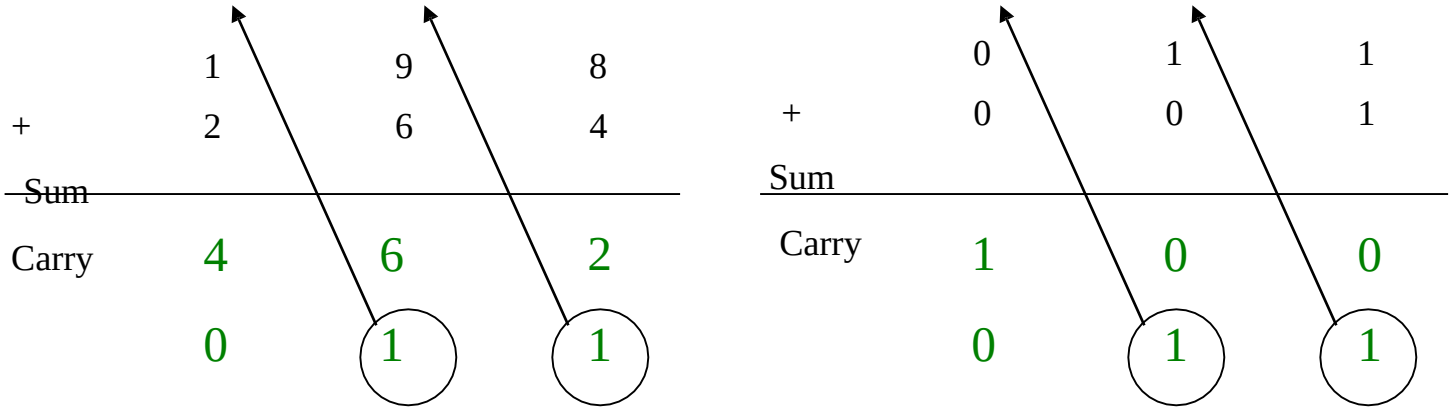
One bit addition:



2 doesn't exist in binary!

Addition

- From right to left, we add each pair of digits
- We write the sum, and add the carry to the next column on the left



Binary Sums and Carries

a	b	Sum
0	0	0
0	1	1
1	0	1
1	1	0

a	b	Carry
0	0	0
0	1	0
1	0	0
1	1	1

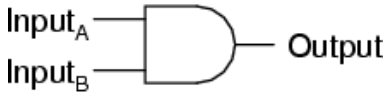
XOR

Exclusive-OR gate

A	B	Output
0	0	0
0	1	1
1	0	1
1	1	0

AND

2-input AND gate



A	B	Output
0	0	0
0	1	0
1	0	0
1	1	1

Binary Addition (cont.)

Example:

```
  1        1 1 1      <- carries
  1 1 0 0 0 0 1 1 1 1
+ 0 1 1 1 1 0 1 0 1 0
-----------------------
1 0 0 1 1 1 1 1 0 0 1   <- sum
```

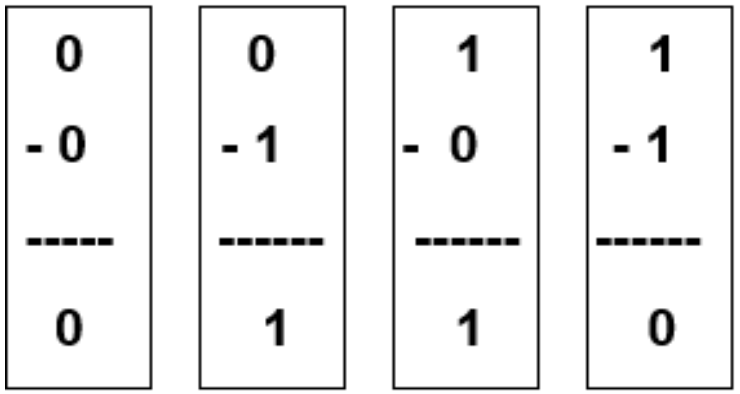
Q: How to verify?

A: Convert to decimal

```
  783
+ 490
-----
 1273
```

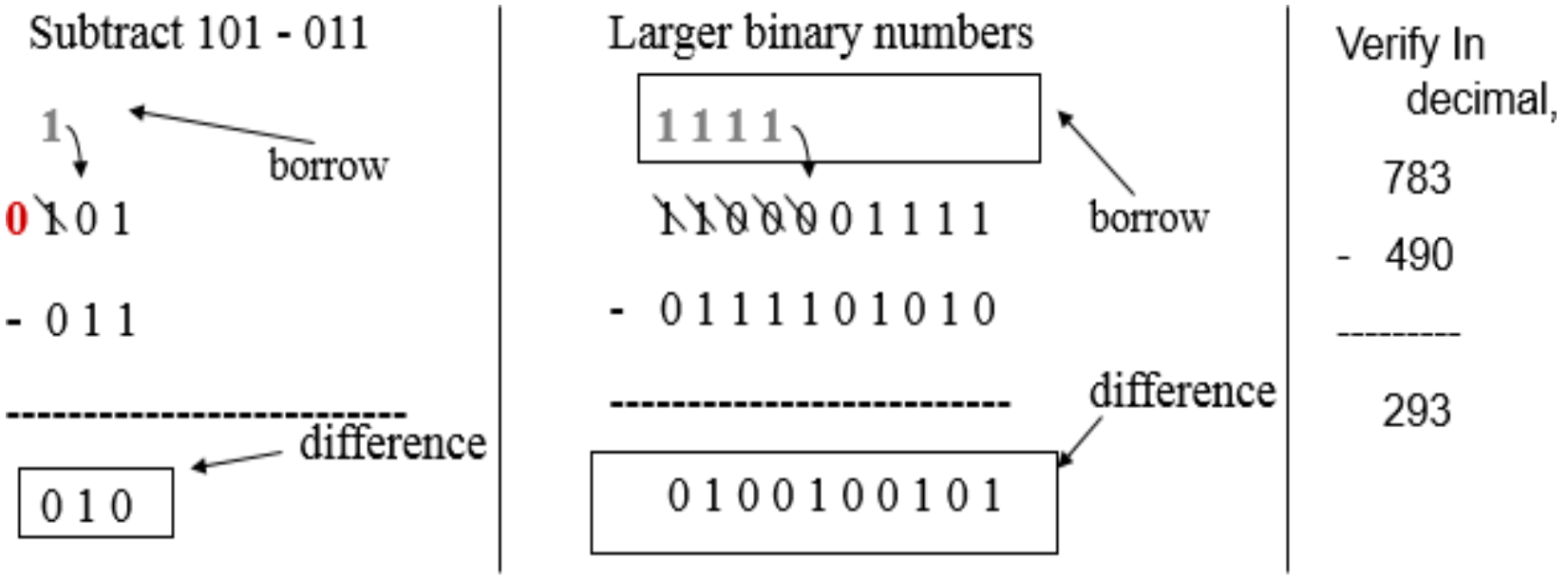
Binary Subtraction

One bit subtraction:



- In binary addition, there is a **sum** and a **carry**.
- In binary subtraction, there is a **difference** and a **borrow**
- **Note: 0 – 1 = 1 borrow 1**

Binary Subtraction (cont.)



- In Decimal subtraction, the borrow is equal to 10.
- In Binary, the borrow is equal to 2. Therefore, a '1' borrowed in binary will generate a $(10)_2$, which equals to $(2)_{10}$ in decimal

One’s Complement

- The additive inverse of a one’s complement representation is found by inverting each bit.

- Inverting each bit is also called taking the one’s complement

Example

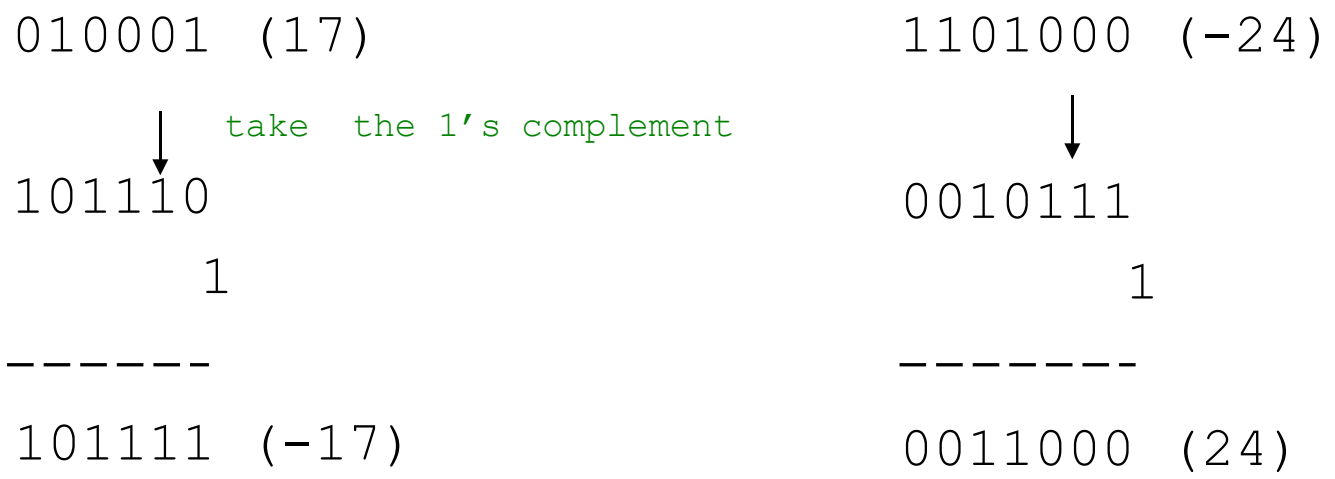
```
0000 0011 (3)
1111 1100 (-3)

1110 1000 (-23)
0001 0111 (23)

0000 0000 (0)
1111 1111 (0)
```

Note: There are two representations of zero

Two's complement

- The additive inverse of a two's complement integer can be obtained by adding 1 to its one's complement

Example



Subtracting the large number from the small number

```
Let's subtract 61 from 45:

    45
  - 61
  ----
   -24

I just followed the usual rules for subtraction: 5-1 is 4; 4-6 is -2.
So is the answer -24? No, because if we add 61 to -24 we get

    61
  - 24
  ----
    37

not 45. What went wrong?
```

Subtracting the large number from the small number

The problem is that when we got that negative sign, it meant only that the 2 in the tens place was negative; the 4 is still positive! So what we realy found was that

$$(40 + 5) - (60 + 1) = (40 - 60) + (5 - 1) = -20 + 4$$

That is not -24, but -16, which is the right answer to the problem.

So we can't subtract a larger number from a smaller one columnwise, because the sign gets mixed up.

The usual way to do this is to apply the fact that

$$a - b = -(b - a)$$

to say that

$$45 - 61 = -(61 - 45) = -16$$

Subtracting the large number from the small number

You would do the same thing in binary (and these are in fact the same numbers):

```
   101101      111101
 - 111101    - 101101
 --------    --------
         = -(010000)
```

That is, you reverse the order of the numbers, subtract, and take the negative.

Group Activity

$$\begin{array}{rrrrrr} & 1 & 0 & 0 & 0 & 1 \\ + & 1 & 1 & 1 & 0 & 1 \\ \hline \end{array}$$

$$\begin{array}{rrrrrr} & 1 & 0 & 1 & 1 & 1 \\ + & 1 & 1 & 0 & 1 & 0 & 1 \\ \hline \end{array}$$

- 10001 + 11101 = 101110:

$$\begin{array}{rrrrrr} 1 & & & & 1 & \\ & 1 & 0 & 0 & 0 & 1 \\ + & 1 & 1 & 1 & 0 & 1 \\ \hline 1 & 0 & 1 & 1 & 1 & 0 \end{array}$$

- 10111 + 110101 = 1001100:

$$\begin{array}{rrrrrrr} & 1 & 1 & & 1 & 1 & 1 \\ & & & 1 & 0 & 1 & 1 & 1 \\ + & & 1 & 1 & 0 & 1 & 0 & 1 \\ \hline & 1 & 0 & 0 & 1 & 1 & 0 & 0 \end{array}$$

Group Activity

$$\begin{array}{rccccccc} & 1 & 0 & 1 & 1 & 0 & 1 & 1 \\ - & & & 1 & 0 & 0 & 1 & 0 \\ \hline \end{array}$$

- $1011011 - 10010 = 1001001$:

$$\begin{array}{rccccccc} & 1 & 0 & 1 & 1 & 0 & 1 & 1 \\ - & & & 1 & 0 & 0 & 1 & 0 \\ \hline & 1 & 0 & 0 & 1 & 0 & 0 & 1 \end{array}$$

Multiplication (decimal)

$$
\begin{array}{r}
13 \\
\times 11 \\
\hline
13 \\
+130 \\
\hline
143
\end{array}
$$

Binary Multiplication

The binary multiplication table is simple:

$0 * 0 = 0$ | $1 * 0 = 0$ | $0 * 1 = 0$ | $1 * 1 = 1$

Extending multiplication to multiple digits:

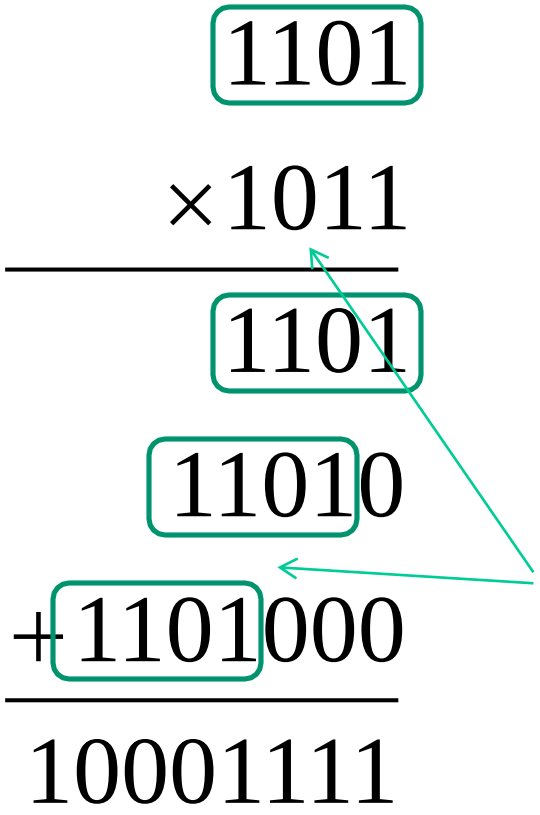
Multiplicand	**1011**
Multiplier	**x**
Partial Products	**10**
	1
	101
	1
	0000

Multiplication (binary)

$$
\begin{array}{r}
1101 \\
\times 1011 \\
\hline
1101 \\
11010 \\
+1101000 \\
\hline
10001111
\end{array}
$$

Multiplication (binary)

It's interesting to note that binary multiplication is a sequence of shifts and adds of the first term (depending on the bits in the second term.



110100 is missing here because the corresponding bit in the second terms is 0.

BASE CONVERSION - Positive Powers of 2

- Useful for Base Conversion

Exponent	Value
0	1
1	2
2	4
3	8
4	16
5	32
6	64
7	128
8	256
9	512
10	1024

Exponent	Value
11	2,048
12	4,096
13	8,192
14	16,384
15	32,768
16	65,536
17	131,072
18	262,144
19	524,288
20	1,048,576
21	2,097,152

Converting Binary to Decimal

- **To convert to decimal, use decimal arithmetic to form Σ (digit × respective power of 2).**
- **Example: Convert 11010_2 to N_{10}:**

Decimal to Binary Conversion

- Converting to binary – can't use polynomial evaluation!

- Whole part and fractional parts must be handled separately!
  - Whole part: Use *repeated division*.
  - Fractional part: Use *repeated multiplication*.
  - Combine results when finished.

Decimal to Binary Conversion (Whole Part: Repeated Division)

- Divide by target radix (2 in this case)
- Remainders become digits in the new representation (0 <= digit < R)
- Digits produced in right to left order.
- Quotient is used as next dividend.
- Stop when the quotient becomes zero, but use the corresponding remainder.

Decimal to Binary Conversion (Whole Part: Repeated Division)

$97 \div 2 \rightarrow$	quotient = 48,	remainder = 1 (LSB)
$48 \div 2 \rightarrow$	quotient = 24,	remainder = 0.
$24 \div 2 \rightarrow$	quotient = 12,	remainder = 0.
$12 \div 2 \rightarrow$	quotient = 6,	remainder = 0.
$6 \div 2 \rightarrow$	quotient = 3,	remainder = 0.
$3 \div 2 \rightarrow$	quotient = 1,	remainder = 1.
$1 \div 2 \rightarrow$	quotient = 0 (Stop)	remainder = 1 (MSB)

Result = $1\ 1\ 0\ 0\ 0\ 0\ 1_2$

Decimal to Binary Conversion (Fractional Part: Repeated Multiplication)

- Multiply by target radix (2 in this case)
- Whole part of product becomes digit in the new representation (0 <= digit < R)
- Digits produced in left to right order.
- Fractional part of product is used as next multiplicand.
- Stop when the fractional part becomes zero (sometimes it won't).

Decimal to Binary Conversion (Fractional Part: Repeated Multiplication)

.1 × 2 ➔ 0.2 (fractional part = .2, whole part = 0)

.2 × 2 ➔ 0.4 (fractional part = .4, whole part = 0)

.4 × 2 ➔ 0.8 (fractional part = .8, whole part = 0)

.8 × 2 ➔ 1.6 (fractional part = .6, whole part = 1)

.6 × 2 ➔ 1.2 (fractional part = .2, whole part = 1)

Result = $.0001100011000110011_2$…..

(How much should we keep?)

Regardless of the number of digits , multiply them by 2

Counting in Binary

Dec	Binary
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111

Note the pattern!

- LSB (bit 0) toggles on *every* count.
- Bit 1 toggles on every *second* count.
- Bit 2 toggles on every *fourth* count.
- Etc….

Question

- Do you trust the used car salesman that tells you that the 1966 Mustang he wants to sell you has only the 13,000 miles that it's odometer shows?
- If not, what has happened?
- Why?

Representation Rollover

- Consequence of *fixed precision.*
- Computers use fixed precision!
- Digits are lost on the left-hand end.
- Remaining digits are still correct.
- Rollover while counting . . .

  Up: “999999” ➔ “000000” (R^n-1 ➔ 0)

  Down: “000000” ➔ “999999” (0 ➔ R^n-1)

Conversion Summary

- We need to know how to convert a number in one system to the equivalent number in another system.
- Since the decimal system is more familiar than the other systems, we first show how to covert from any base to decimal.
- Then we show how to convert from decimal to any base.
- Finally, we show how we can easily convert from binary to hexadecimal or octal and vice versa.

Any base to decimal conversion

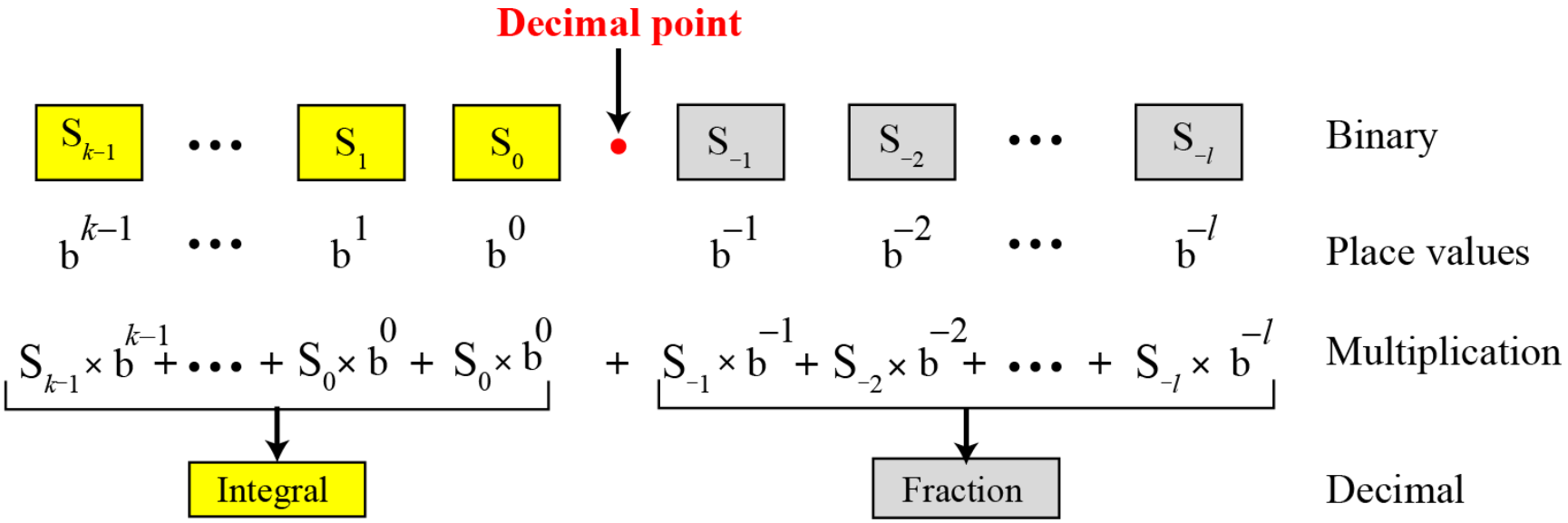


Figure 2.5 Converting other bases to decimal

Example

The following shows how to convert the binary number (110.11)2 to decimal: (110.11)2 = 6.75.

Binary	1		1		0	•	1	1
Place values	2^2		2^1		2^0		2^{-1}	2^{-2}
Partial results	4	+	2	+	0	+	0.5	+ 0.25
Decimal: 6.75								

Example

The following shows how to convert the hexadecimal number (1A.23)16 to decimal.

Hexadecimal	1		A	•	2		3
Place values	16^1		16^0		16^{-1}		16^{-2}
Partial result	16	+	10	+	0.125	+	0.012

Decimal: 26.137

Note that the result in the decimal notation is not exact, because

$3 \times 16^{-2} = 0.01171875$. We have rounded this value to three digits (0.012).

Example

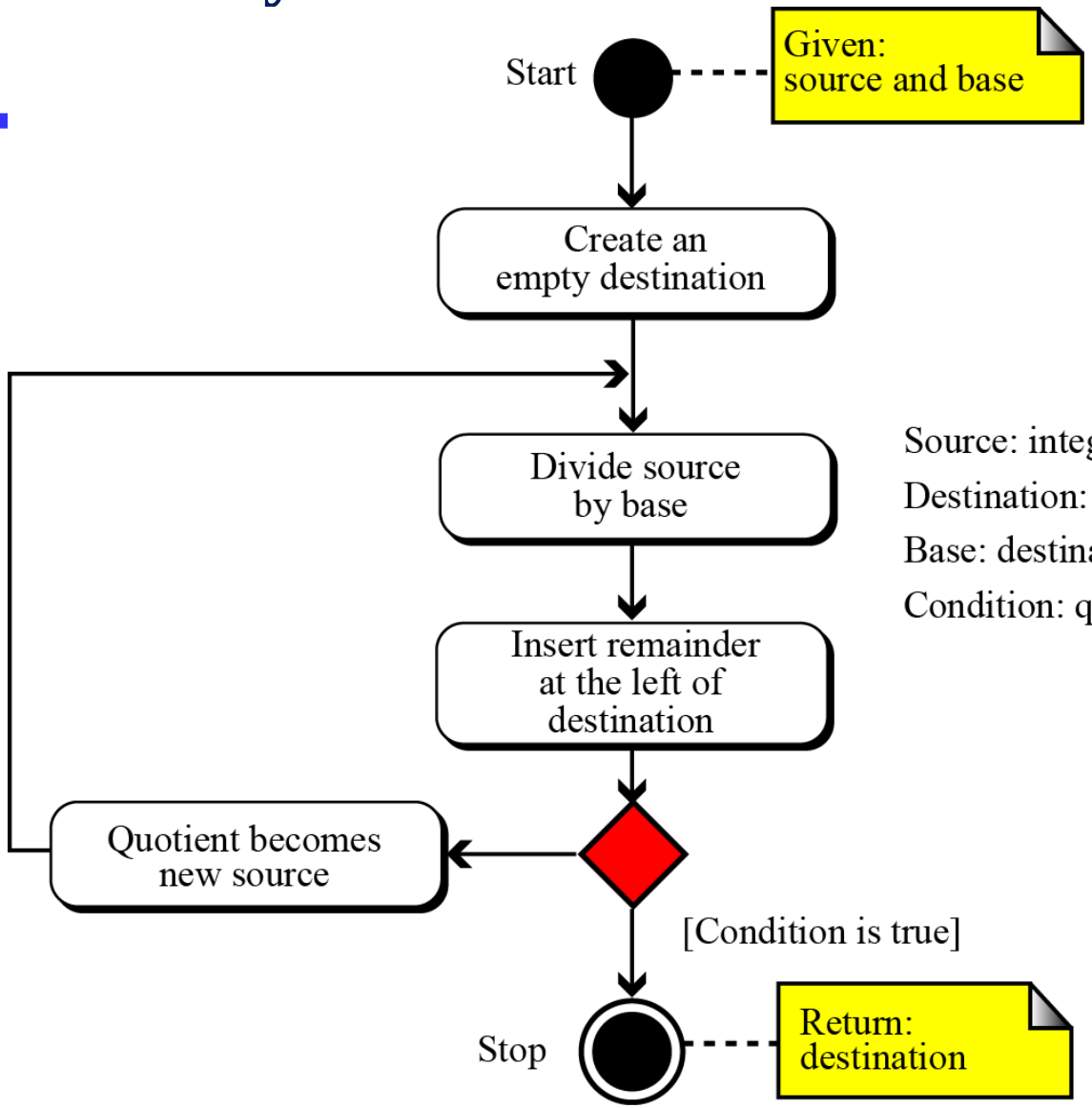
The following shows how to convert $(23.17)_8$ to decimal.

Octal	2		3	•	1		7
Place values	8^1		8^0		8^{-1}		8^{-2}
Partial result	16	+	3	+	0.125	+	0.109

Decimal: 19.234

This means that $(23.17)_8 \approx 19.234$ in decimal. Again, we have rounded up $7 \times 8^{-2} = 0.109375$.

Decimal to any base



Source: integral part of decimal number

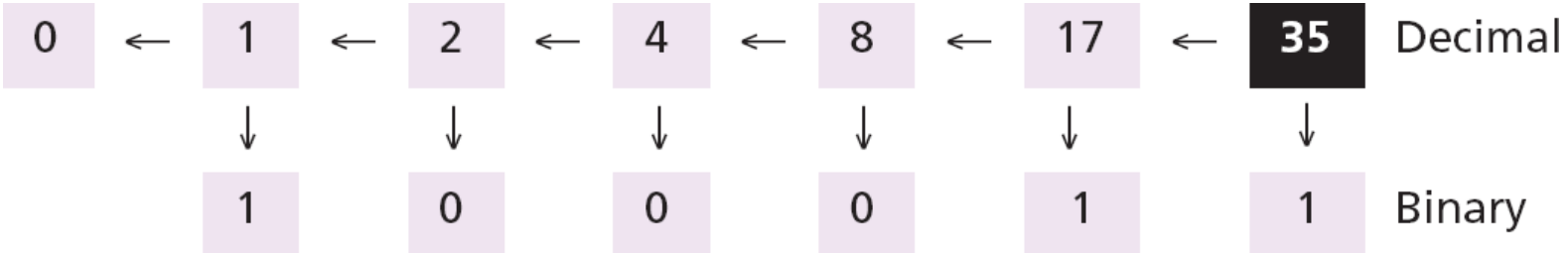
Destination: integral part of converted number

Base: destination base

Condition: quotient is zero

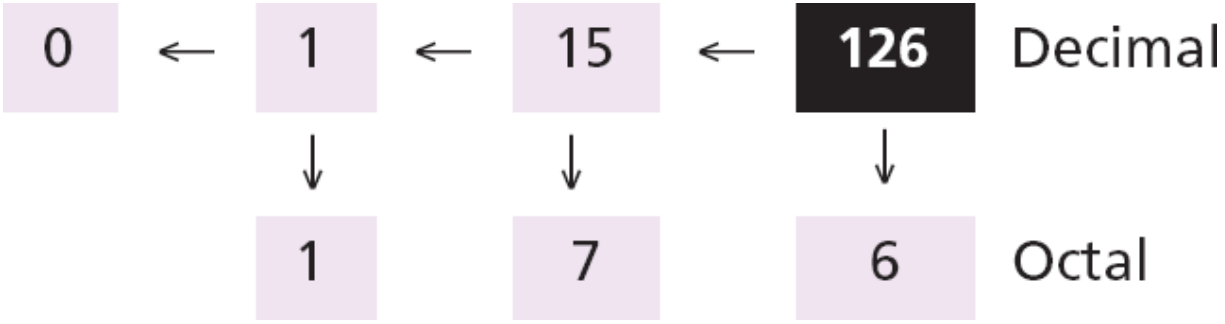
Example

The following shows how to convert 35 in decimal to binary. We start with the number in decimal, we move to the left while continuously finding the quotients and the remainder of division by 2. The result is 35 = (100011)2.



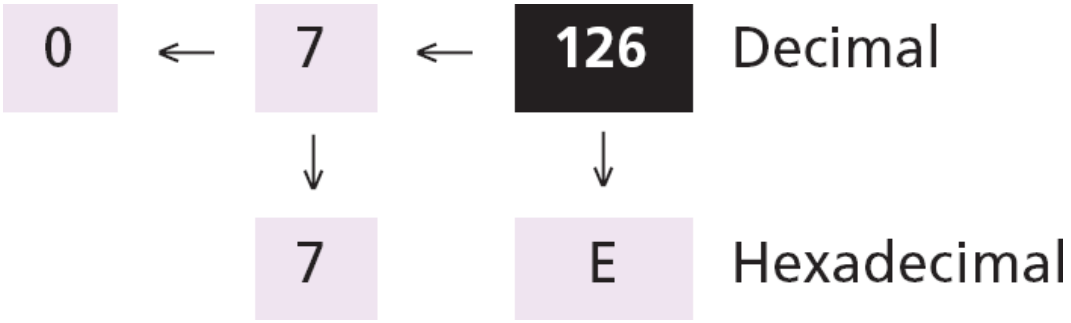
Example

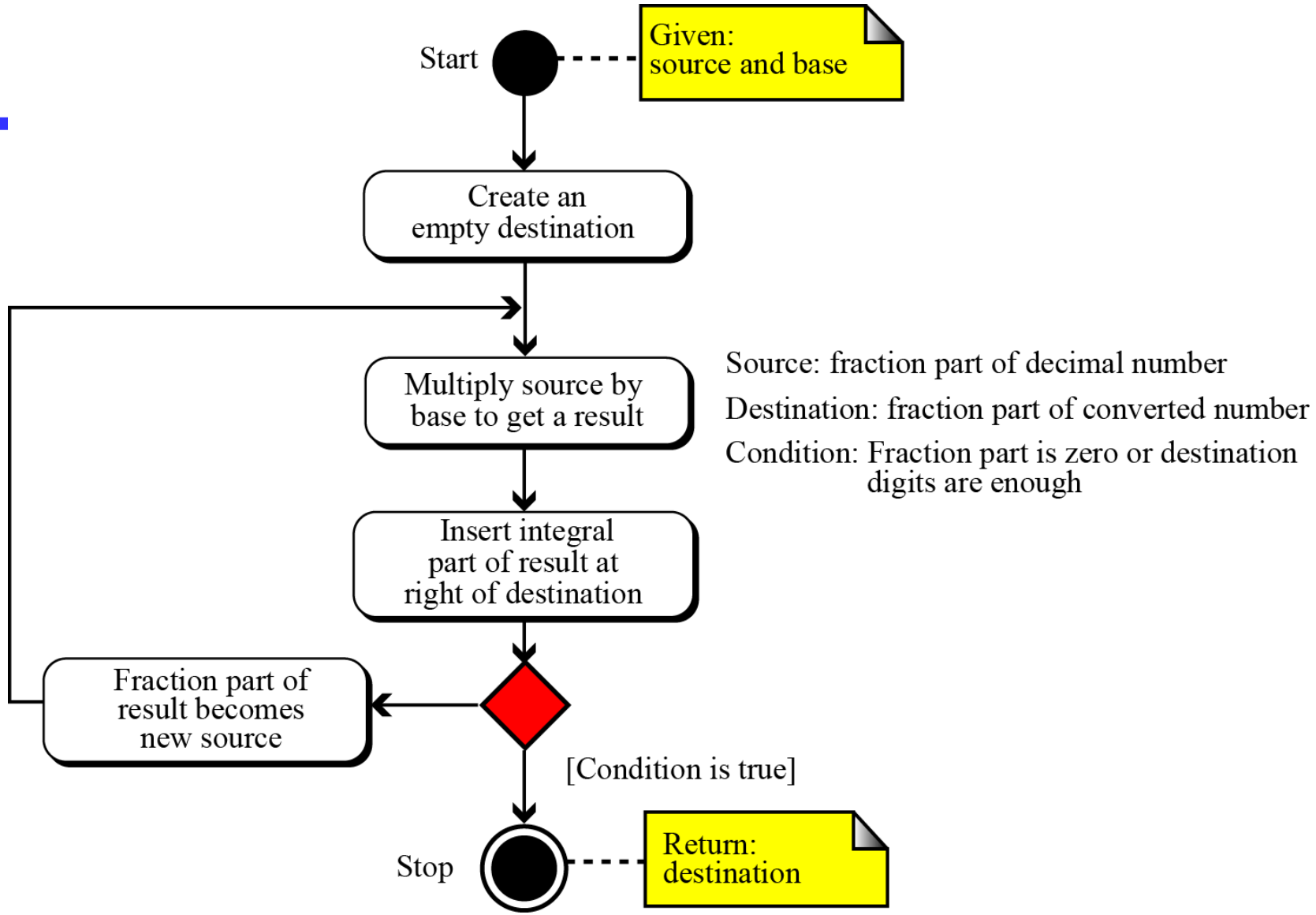
The following shows how to convert 126 in decimal to its equivalent in the octal system. We move to the right while continuously finding the quotients and the remainder of division by 8. The result is 126 = (176)8.



Example

The following shows how we convert 126 in decimal to its equivalent in the hexadecimal system. We move to the right while continuously finding the quotients and the remainder of division by 16. The result is 126 = (7E)16

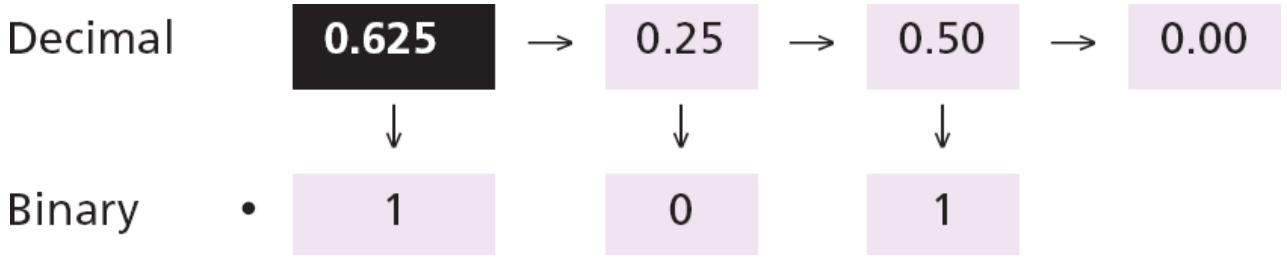




Converting the fractional part of a number in decimal to other bases

Example

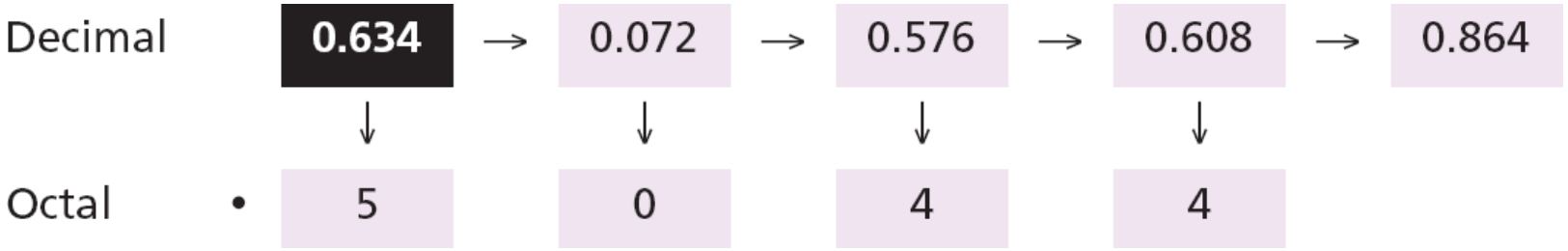
Convert the decimal number 0.625 to binary.



Since the number 0.625 = (0.101)2 has no integral part, the example shows how the fractional part is calculated.

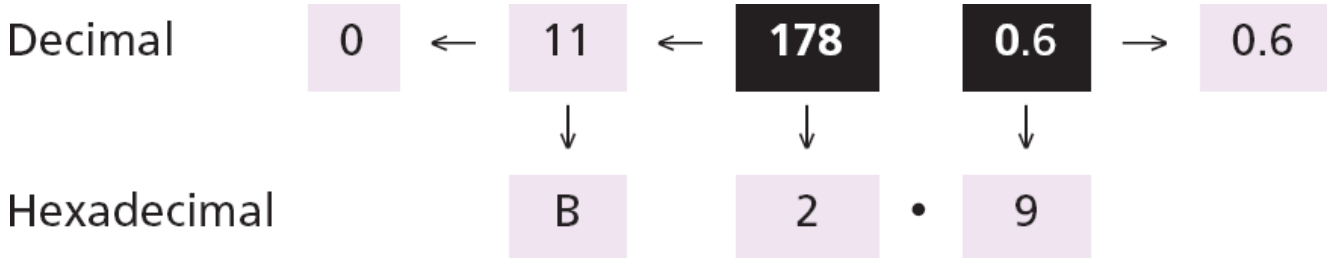
Example

The following shows how to convert 0.634 to octal using a maximum of four digits. The result is 0.634 = (0.5044)8. Note that we multiple by 8 (base octal).



Example

The following shows how to convert 178.6 in decimal to hexadecimal using only one digit to the right of the decimal point. The result is 178.6 = (B2.9)16 Note that we divide or multiple by 16 (base hexadecimal).



Example

An alternative method for converting a small decimal integer (usually less than 256) to binary is to break the number as the sum of numbers that are equivalent to the binary place values shown:

Place values	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
Decimal equivalent	128	64	32	16	8	4	2	1

Decimal 165 =	128	+	0	+	32	+	0	+	0	+	4	+	0	+	1
Binary	1		0		1		0		0		1		0		1

Example

A similar method can be used to convert a decimal fraction to binary when the denominator is a power of two:

Place values	2^{-1}	2^{-2}	2^{-3}	2^{-4}	2^{-5}	2^{-6}	2^{-7}
Decimal equivalent	1/2	1/4	1/8	1/16	1/32	1/64	1/128

Decimal = 27/64 = 16/64 + 8/64 + 2/64 + 1/64

= 1/4 + 1/8 + 1/32 + 1/64

Decimal 27/64 =	0	+	1/4	+	1/8	+	0	+	1/32	+	1/64
Binary	0		1		1		0		1		1

The answer is then $(0.011011)_2$

Example

Show the hexadecimal equivalent of the binary number $(110011100010)_2$.

Solution

We first arrange the binary number in 4-bit patterns:

1100 1110 0010

Note that the leftmost pattern can have one to four bits.

Example

What is the binary equivalent of (24C)16?

Solution

Each hexadecimal digit is converted to 4-bit patterns:

$2 \rightarrow 0010$, $4 \rightarrow 0100$, and $C \rightarrow 1100$

The result is $(001001001100)_2$.

Example

Show the octal equivalent of the binary number (101110010)2.

Solution

Each group of three bits is translated into one octal digit.

101 110 010

The result is (562)8.

Example

What is the binary equivalent of for (24)8?

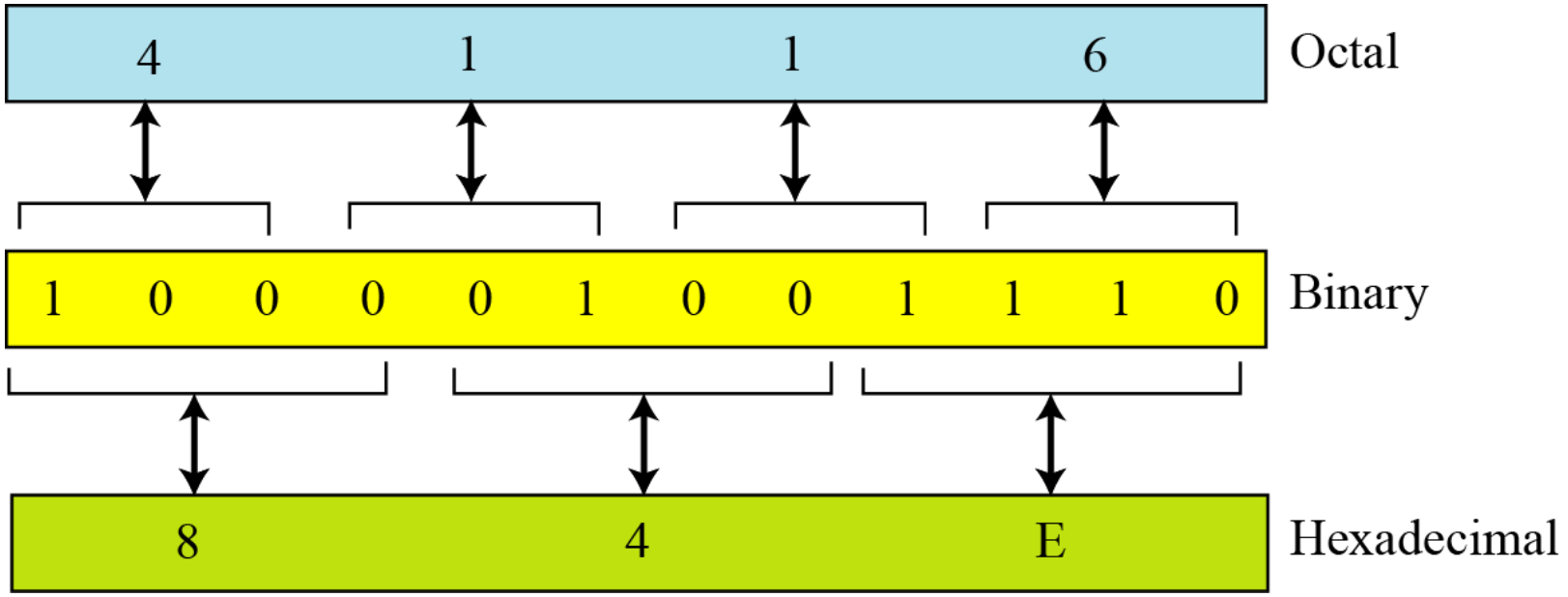
Solution

Write each octal digit as its equivalent bit pattern to get

$$2 \rightarrow 010 \quad \text{and} \quad 4 \rightarrow 100$$

The result is (010100)2.

Octal-hexadecimal conversion



Octal to hexadecimal and hexadecimal to octal conversion

Binary Numbers and Binary Coding

- **Flexibility of representation**
  - **Within constraints below, can assign any binary combination (called a code word) to any data as long as data is uniquely encoded.**
- **Information Types**
  - **Numeric**
    - **Must represent range of data needed**
    - **Very desirable to represent data such that simple, straightforward computation for common arithmetic operations permitted**
    - **Tight relation to binary numbers**
  - **Non-numeric**
    - **Greater flexibility since arithmetic operations not applied.**
    - **Not tied to binary numbers**

Non-numeric Binary Codes

- Given *n* binary digits (called bits), a binary code is a mapping from a set of represented elements to a subset of the 2^n binary numbers.
- Example: A binary code for the seven colors of the rainbow
- Code 100 is not used

Color	Binary Number
Red	000
Orange	001
Yellow	010
Green	011
Blue	101
Indigo	110
Violet	111

Number of Elements Represented

- Given n digits in radix r, there are r^n distinct elements that can be represented.
- But, you can represent m elements, m < r^n
- Examples:
  - You can represent 4 elements in radix $r = 2$ with $n = 2$ digits: (00, 01, 10, 11).
  - You can represent 4 elements in radix $r = 2$ with $n = 4$ digits: (0001, 0010, 0100, 1000).
  - This second code is called a "one hot" code.

DECIMAL CODES - Binary Codes for Decimal Digits

- There are over 8,000 ways that you can chose 10 elements from the 16 binary numbers of 4 bits. A few are useful:

Decimal	8,4,2,1	Excess3	8,4,-2,-1	Gray
0	0000	0011	0000	0000
1	0001	0100	0111	0100
2	0010	0101	0110	0101
3	0011	0110	0101	0111
4	0100	0111	0100	0110
5	0101	1000	1011	0010
6	0110	1001	1010	0011
7	0111	1010	1001	0001
8	1000	1011	1000	1001
9	1001	1100	1111	1000

GRAY CODE – Decimal

Decimal	8,4,2,1	Gray
0	0000	0000
1	0001	0100
2	0010	0101
3	0011	0111
4	0100	0110
5	0101	0010
6	0110	0011
7	0111	0001
8	1000	1001
9	1001	1000

- **What special property does the Gray code have in relation to adjacent decimal digits?**

UNICODE

- **UNICODE extends ASCII to 65,536 universal characters codes**
  - **For encoding characters in world languages**
  - **Available in many modern applications**
  - **2 byte (16-bit) code words**
  - **See Reading Supplement – Unicode on the Companion Website http://www.prenhall.com/mano**

PARITY BIT Error-Detection Codes

- **Redundancy (e.g. extra information), in the form of extra bits, can be incorporated into binary code words to detect and correct errors.**
- **A simple form of redundancy is parity, an extra bit appended onto the code word to make the number of 1's odd or even. Parity can detect all single-bit errors and some multiple-bit errors.**
- **A code word has even parity if the number of 1's in the code word is even.**
- **A code word has odd parity if the number of 1's in the code word is odd.**

4-Bit Parity Code Example

- **Fill in the even and odd parity bits:**

Even Parity Message- Parity	Odd Parity Message - Parity
000 -	000 -
001 -	001 -
010 -	010 -
011 -	011 -
100 -	100 -
101 -	101 -
110 -	110 -
111 -	111 -

- **The codeword "1111" has <u>even parity</u> and the codeword "1110" has <u>odd parity</u>. Both can be used to represent 3-bit data.**

Binary-Coded Decimal (BCD)

- a BCD number is just a natural binary encoding of the decimal digits from 0 to 9 on four bits.

0101 0111

59 in BCD (0 ~ 99)

because there are unused code words

87 in normal unsigned binary number (0 ~ 255)

Binary-Coded Decimal (BCD)

- Binary-Coded Decimal is a weighted code because each decimal digit can be obtained from its code word by assigning a fixed weight to each code-word bit.

- The weights for the BCD bits are 8, 4, 2, and 1, and for this reason the code is sometimes called the 8421 code.

Excess-3 code

- This code is also self-complementing like 2421 code.
- Although this code is not weighted, it has an arithmetic relationship with the BCD code.
- The code word for each decimal digit is the corresponding BCD code word plus 0011_2.

$$\begin{aligned} 0010 &= 2 \quad \text{in BCD} \\ +\ 0011_2 & \\ = 0101 &= 2 \quad \text{in excess-3} \end{aligned}$$

Gray Code

- Gray code is a code where only one bit changes at a time while traversing from 0 to any decimal number in sequence.

- This is a useful property when converting analog values into digital values, since it eliminates the problem of misinterpreting asynchronous changes to bits between valid values.

Gray Code

Table 2-10
A comparison of 3-bit binary code and Gray code.

Decimal Number	*Binary Code*	*Gray Code*
0	000	000
1	001	001
2	010	011
3	011	010
4	100	110
5	101	111
6	110	101
7	111	100

Parity check

- One of the most common ways to achieve error detection is by means of a parity bit.
- A parity bit is an extra bit included with a message to make the total number of 1's transmitted either odd or even.
- If an odd parity is adopted, the P bit is chosen such that the total number of 1's is odd.

Information Bits	*Even-parity Code*	*Odd-parity Code*
000	000 0	000 1
001	001 1	001 0
010	010 1	010 0
011	011 0	011 1
100	100 1	100 0
101	101 0	101 1
110	110 0	110 1
111	111 1	111 0

Table 2-13
Distance-2 codes with three information bits.

DIGITAL LOGIC CIRCUITS

Logic Gates

Boolean Algebra

Map Specification

Combinational Circuits

Flip-Flops

Sequential Circuits

Memory Components

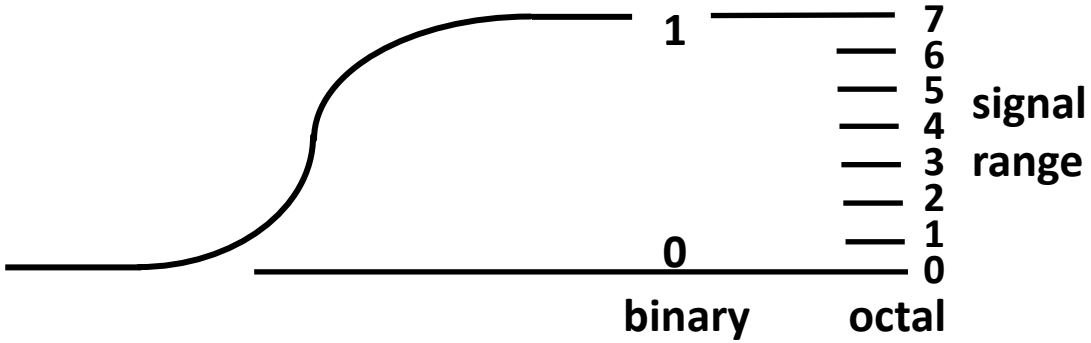
Integrated Circuits

LOGIC GATES

Digital Computers

- Imply that the computer deals with digital information, i.e., it deals with the information that is represented by binary digits
- Why *BINARY* ? instead of Decimal or other number system ?

* Consider electronic signal

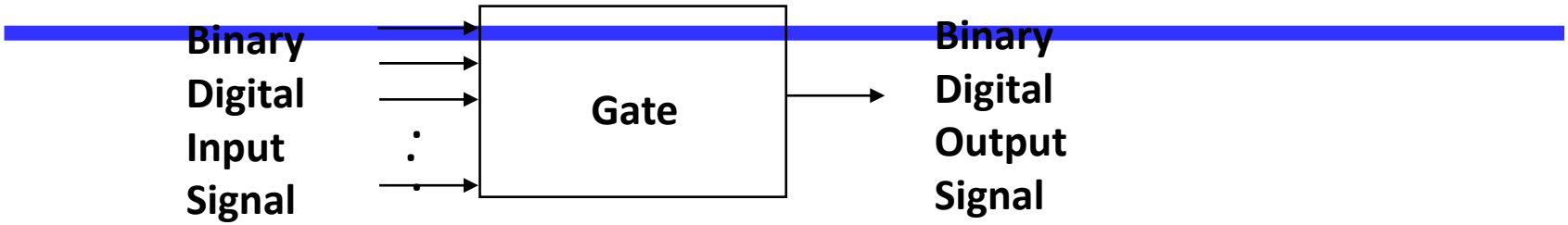


* Consider the calculation cost - Add

	0	1
0	0	1
1	1	10

	0	1	2	3	4	5	6	7	8	9
0	0	1	2	3	4	5	6	7	8	9
1	1	2	3	4	5	6	7	8	9	10
2	2	3	4	5	6	7	8	9	1011	
3	3	4	5	6	7	8	9	101112		
4	4	5	6	7	8	9	10111213			
5	5	6	7	8	9	1011121314				
6	6	7	8	9	101112131415					
7	7	8	9	10111213141516						
8	8	9	1011121314151617							
9	9	101112131415161718								

BASIC LOGIC BLOCK - GATE -



Types of Basic Logic Blocks

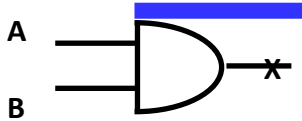
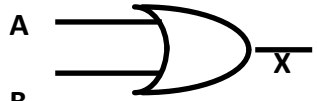
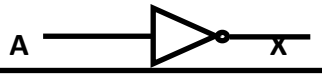
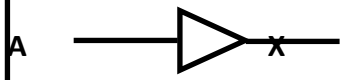
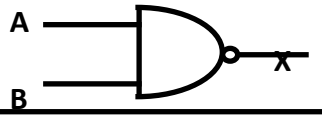
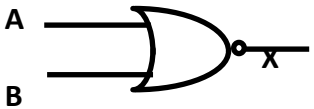
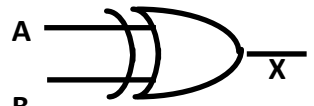
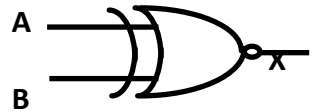
- **Combinational Logic Block**
  Logic Blocks whose output logic value depends only on the input logic values

- **Sequential Logic Block**
  Logic Blocks whose output logic value depends on the input values and the state (stored information) of the blocks

Functions of Gates can be described by

- **Truth Table**
- **Boolean Function**
- **Karnaugh Map**

COMBINATIONAL 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
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

BOOLEAN ALGEBRA

Boolean Algebra

* Algebra with Binary(Boolean) Variable and Logic Operations
* Boolean Algebra is useful in Analysis and Synthesis of Digital Logic Circuits

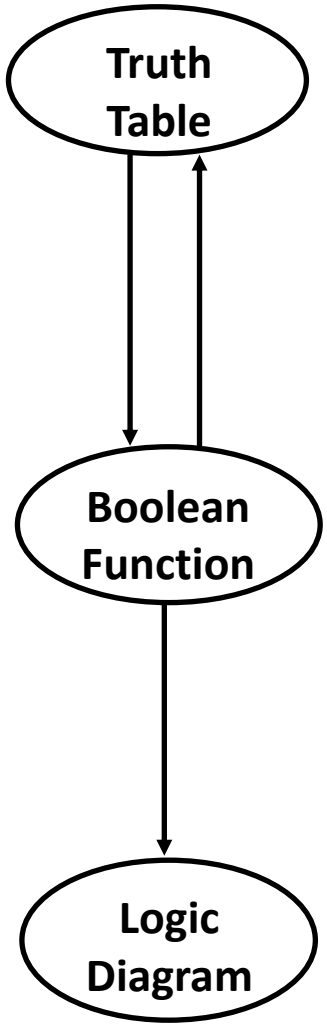
- Input and Output signals can be represented by Boolean Variables, and
- Function of the Digital Logic Circuits can be represented by Logic Operations, i.e., Boolean Function(s)
- From a Boolean function, a logic diagram can be constructed using AND, OR, and I

Truth Table

* The most elementary specification of the function of a Digital Logic Circuit is the Truth Table

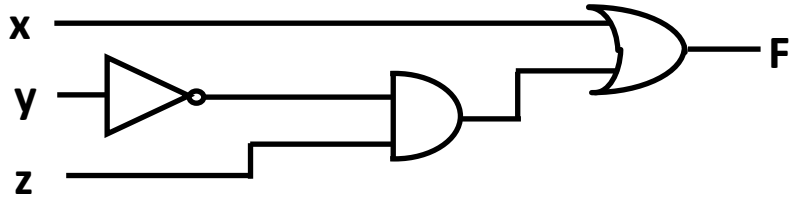
- Table that describes the Output Values for all the combinations of the Input Values, called *MINTERMS*
- n input variables → 2^n minterms

LOGIC CIRCUIT DESIGN



x	y	z	F
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

$F = x + y'z$



BASIC IDENTITIES OF BOOLEAN ALGEBRA

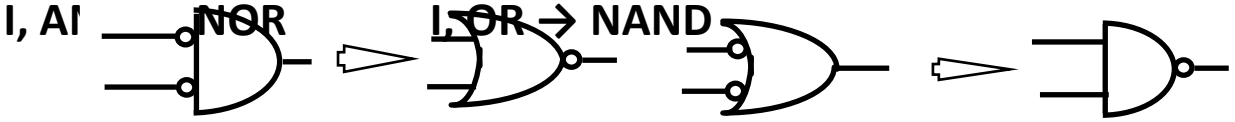
[1] x + 0 = x	[2] x • 0 = 0
[3] x + 1 = 1	[4] x • 1 = x
[5] x + x = x	[6] x • x = x
[7] x + x' = 1	[8] x • X' = 0
[9] x + y = y + x	[10] xy = yx
[11] x + (y + z) = (x + y) + z	[12] x(yz) = (xy)z
[13] x(y + z) = xy +xz	[14] x + yz = (x + y)(x + z)
[15] (x + y)' = x'y'	[16] (xy)' = x' + y'
[17] (x')' = x	

[15] and [16] : De Morgan's Theorem

Usefulness of this Table

- Simplification of the Boolean function
- Derivation of equivalent Boolean functions
  to obtain logic diagrams utilizing different logic gates
  -- Ordinarily ANDs, ORs, and Inverters
  -- But a certain different form of Boolean function may be convenient
  to obtain circuits with NANDs or NORs
  → Applications of De Morgans Theorem

x'y' = (x + y)' x'+ y'= (xy)'



EQUIVALENT CIRCUITS

Many different logic diagrams are possible for a given Function

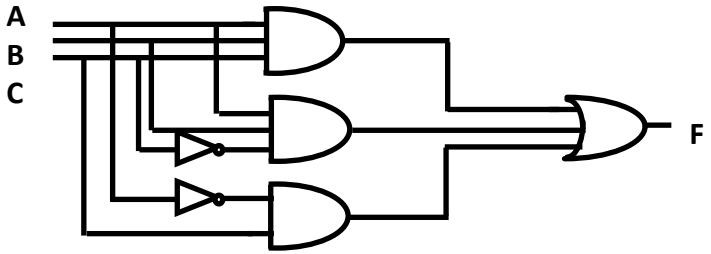
$F = ABC + ABC' + A'C$ ..... (1)

$= AB(C + C') + A'C$ [13] ..... (2)

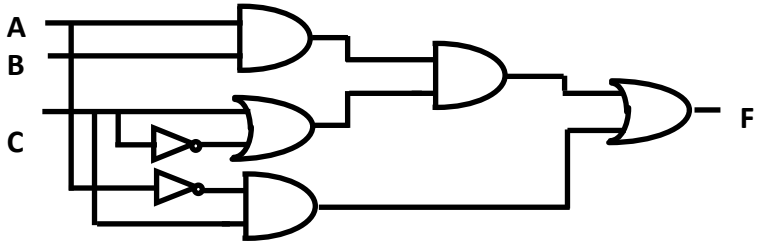
$= AB \bullet 1 + A'C$ [7]

$= AB + A'C$ [4] ..... (3)

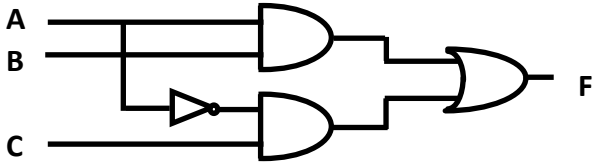
(1)



(2)



(3)



COMPLEMENT OF FUNCTIONS

A Boolean function of a digital logic circuit is represented by only using logical variables and AND, OR, and Invert operators.

→ Complement of a Boolean function

- Replace all the variables and subexpressions in the parentheses appearing in the function expression with their respective complements

$$A,B,\ldots,Z,a,b,\ldots,z \Rightarrow A',B',\ldots,Z',a',b',\ldots,z'$$
$$(p + q) \Rightarrow (p + q)'$$

- Replace all the operators with their respective complementary operators

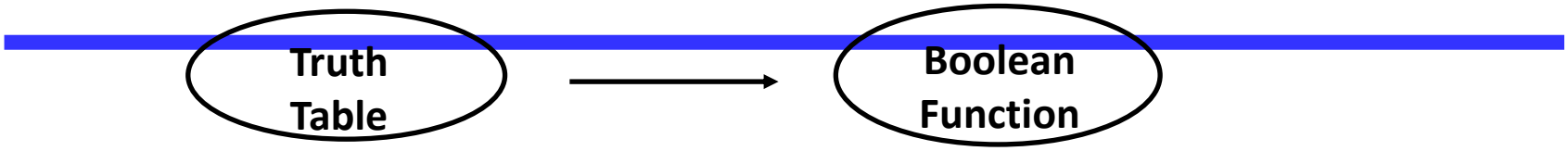
$$\text{AND} \Rightarrow \text{OR}$$
$$\text{OR} \Rightarrow \text{AND}$$

- Basically, extensive applications of the De Morgan's theorem

$$(x_1 + x_2 + \ldots + x_n)' \Rightarrow x_1'x_2'\ldots x_n'$$

$$(x_1x_2 \ldots x_n)' \Rightarrow x_1' + x_2' + \ldots + x_n'$$

SIMPLIFICATION



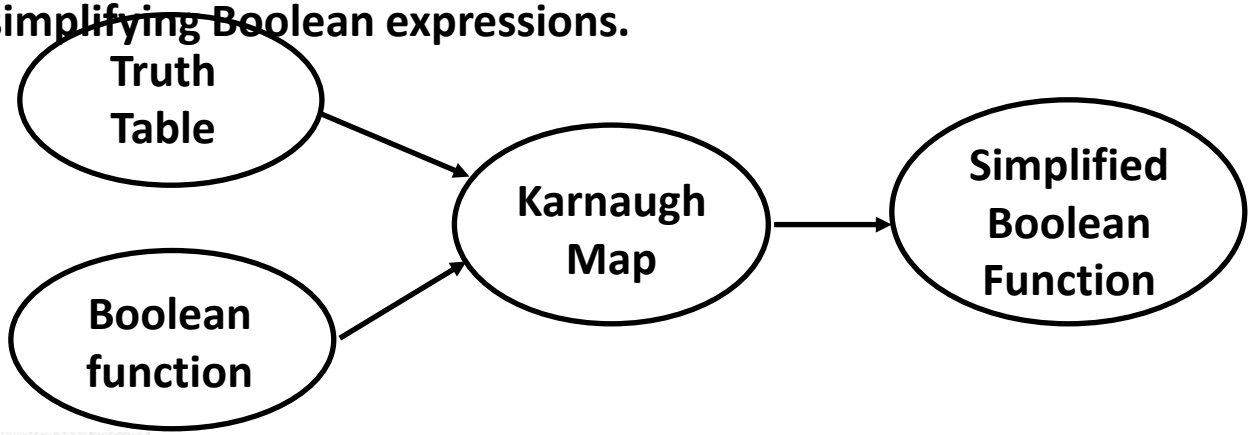
Unique

Many different expressions exist

Simplification from Boolean function

- **Finding an equivalent expression that is least expensive to implement**
- **For a simple function, it is possible to obtain a simple expression for low cost implementation**
- **But, with complex functions, it is a very difficult task**

Karnaugh Map (K-map) is a simple procedure for simplifying Boolean expressions.



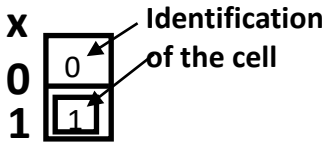
KARNAUGH MAP

Karnaugh Map for an n-input digital logic circuit (n-variable sum-of-products form of Boolean Function, or Truth Table) is

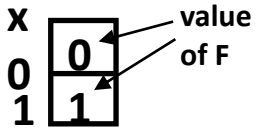
- **Rectangle divided into 2^n cells**
- **Each cell is associated with a *Minterm***
- **An output(function) value for each input value associated with a mintern is written in the cell representing the minterm**
  → 1-cell, 0-cell

Each Minterm is identified by a decimal number whose binary representation is identical to the binary interpretation of the input values of the minterm.

x	F
0	1
1	0



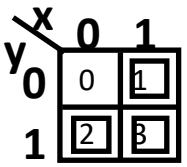
Karnaugh Map

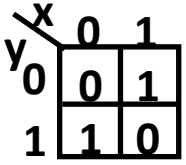


$F(x) = \sum (1)$

1-cell

x	y	F
0	0	0
0	1	1
1	0	1
1	1	1

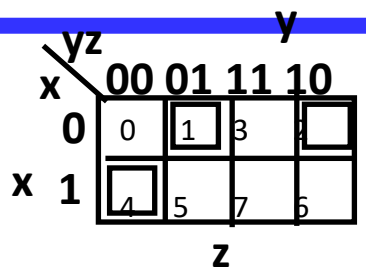


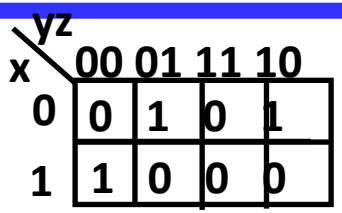


$F(x,y) = \sum (1,2)$

KARNAUGH MAP

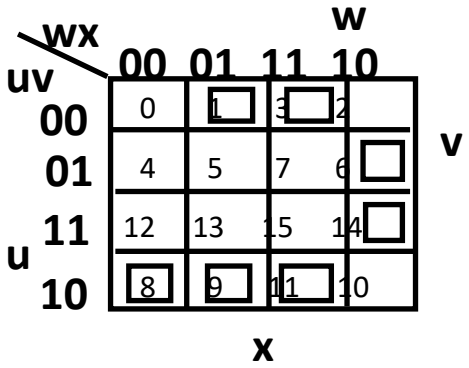
x	y	z	F
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	0

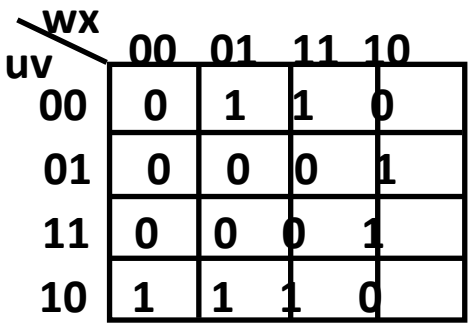




$F(x,y,z) = \sum (1,2,4)$

u	v	w	x	F
0	0	0	0	0
0	0	0	1	1
0	0	1	0	0
0	0	1	1	1
0	1	0	0	0
0	1	0	1	0
0	1	1	0	1
0	1	1	1	0
1	0	0	0	1
1	0	0	1	1
1	0	1	0	0
1	0	1	1	1
1	1	0	0	0
1	1	0	1	0
1	1	1	0	1
1	1	1	1	0





$F(u,v,w,x) = \sum (1,3,6,8,9,11,14)$

MAP SIMPLIFICATION - 2 ADJACENT CELLS -

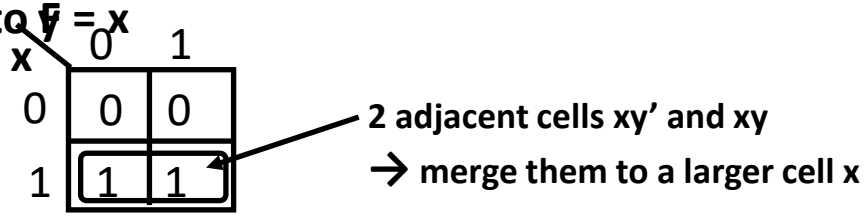
Rule: xy' +xy = x(y+y') = x

Adjacent cells

- binary identifications are different in one bit
→ minterms associated with the adjacent cells have one variable complemented each other

Cells (1,0) and (1,1) are adjacent
Minterms for (1,0) and (1,1) are
x • y' --> x=1, y=0
x • y --> x=1, y=1

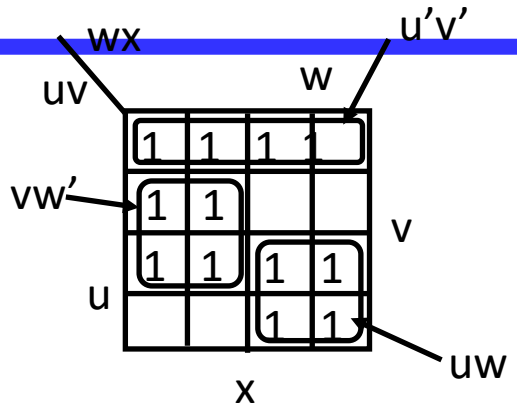
F = xy'+ xy can be reduced to F = x
From the map

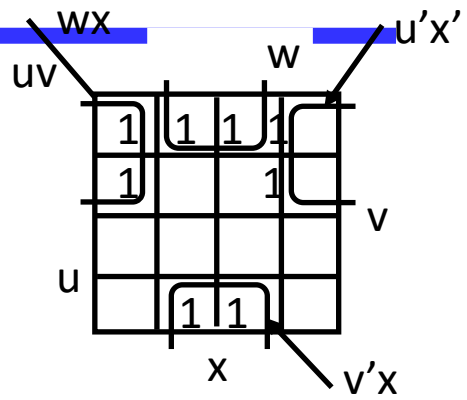


$F(x,y) = \sum(2,3)$
$= xy' + xy$
$= x$

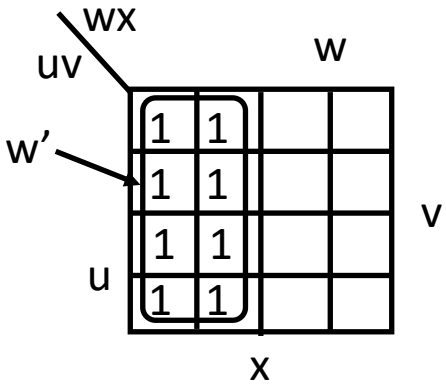
MAP SIMPLIFICATION - MORE THAN 2 CELLS -

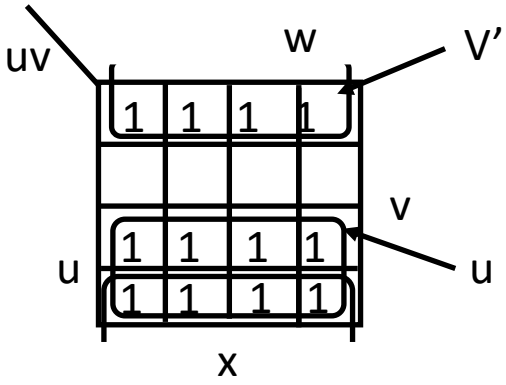
u'v'w'x' + u'v'w'x + u'v'wx + u'v'wx'
= u'v'w'(x'+x) + u'v'w(x+x')
= u'v'w' + u'v'w
= u'v'(w'+w)
= u'v'



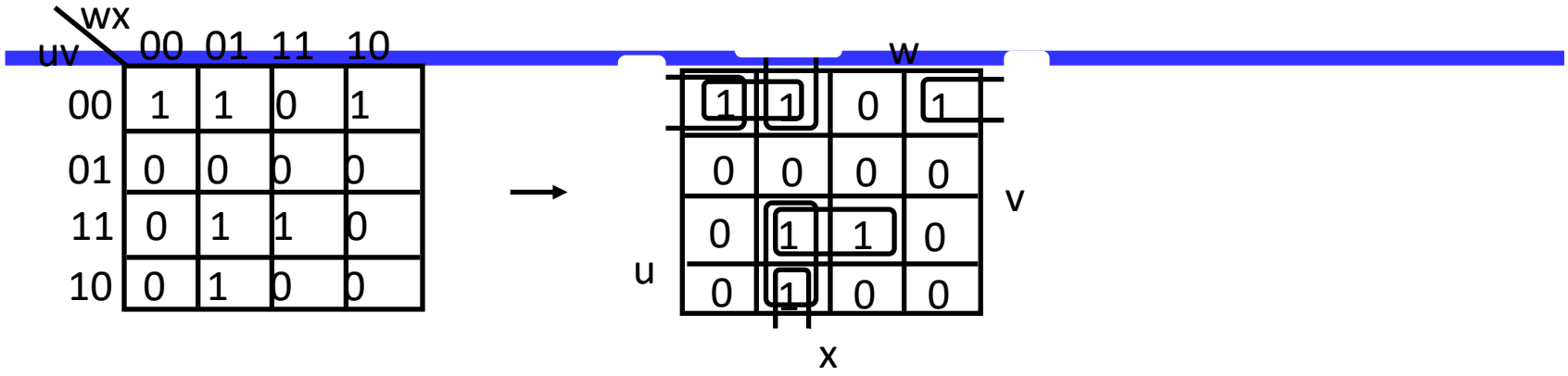


u'v'w'x'+u'v'w'x+u'vw'x'+u'vw'x+uvw'x'+uvw'x+uv'w'x'+uv'w'x
= u'v'w'(x'+x) + u'vw'(x'+x) + uvw'(x'+x) + uv'w'(x'+x)
= u'(v'+v)w' + u(v'+v)w'
= (u'+u)w' = w'





MAP SIMPLIFICATION



$F(u,v,w,x) = \sum (0,1,2,9,13,15)$

(0,1), (0,2), (0,4), (0,8)

Adjacent Cells of 1

Adjacent Cells of 0

(1,0), (1,3), (1,5), (1,9)

...

...

Adjacent Cells of 15

(15,7), (15,11), (15,13), (15,14)

→

Merge (0,1) and (0,2)

--> u'v'w' + u'v'x'

Merge (1,9)

--> v'w'x

Merge (9,13)

--> uw'x

Merge (13,15)

--> uvx

F = u'v'w' + u'v'x' + v'w'x + uw'x + uvx

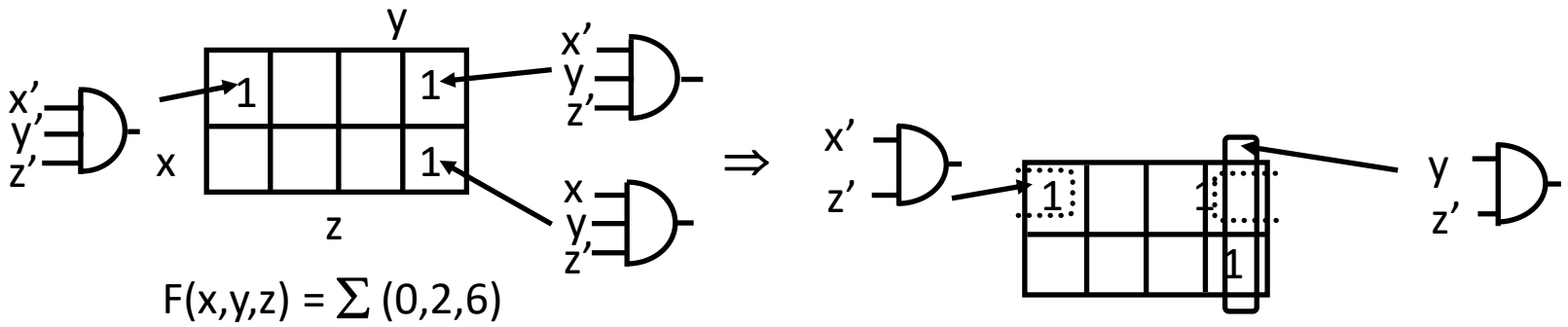
But (9,13) is covered by (1,9) and (13,15)

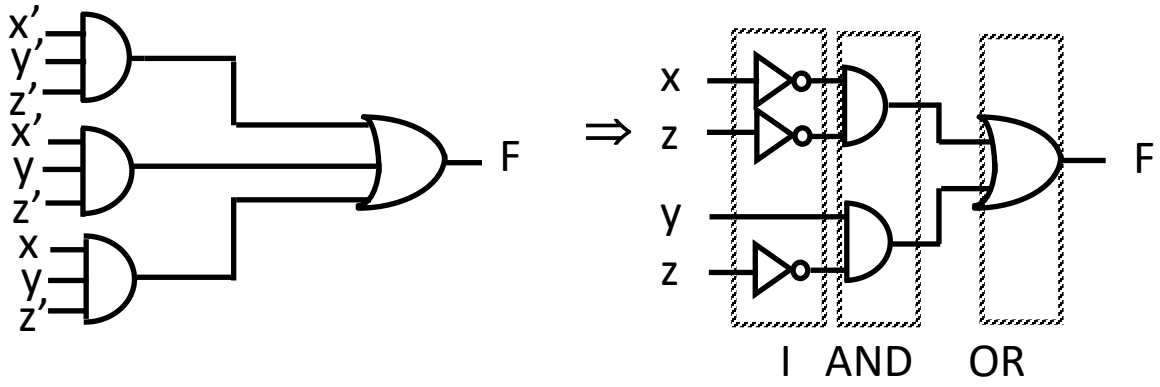
F = u'v'w' + u'v'x' + v'w'x + uvx

IMPLEMENTATION OF K-MAPS - Sum-of-Products Form -

Logic function represented by a Karnaugh map can be implemented in the form of I-AND-OR

A cell or a collection of the adjacent 1-cells can be realized by an AND gate, with some inversion of the input variables.



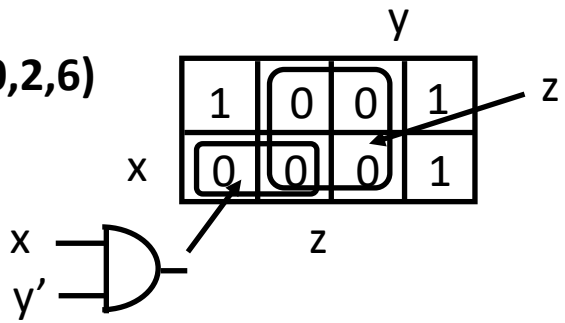


IMPLEMENTATION OF K-MAPS - Product-of-Sums Form -

Logic function represented by a Karnaugh map can be implemented in the form of I-OR-AND

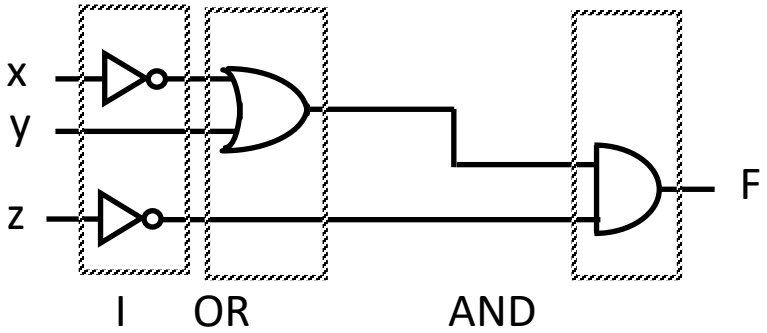
If we implement a Karnaugh map using 0-cells, the complement of F, i.e., F', can be obtained. Thus, by complementing F' using DeMorgan's theorem F can be obtained

F(x,y,z) = (0,2,6)



F' = xy' + z

F = (xy')z'
= (x' + y)z'



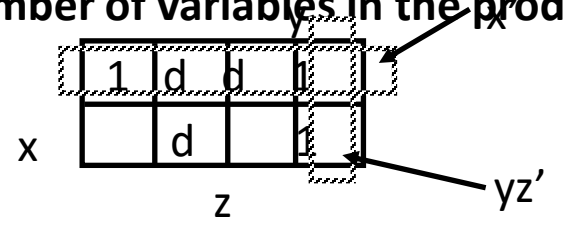
IMPLEMENTATION OF K-MAPS
- Don't-Care Conditions -

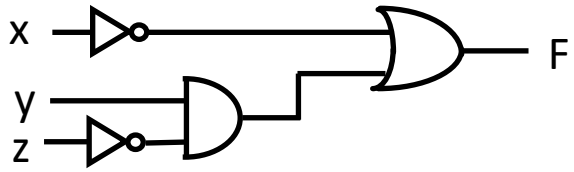
In some logic circuits, the output responses for some input conditions are don't care whether they are 1 or 0.

In K-maps, don't-care conditions are represented by d's in the corresponding cells.

Don't-care conditions are useful in minimizing the logic functions using K-map.

- **Can be considered either 1 or 0**
- **Thus increases the chances of merging cells into the larger cells**
  --> Reduce the number of variables in the product terms

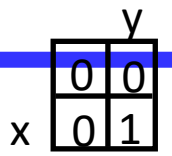




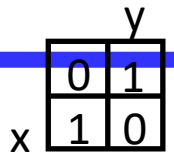
COMBINATIONAL LOGIC CIRCUITS

Half Adder

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

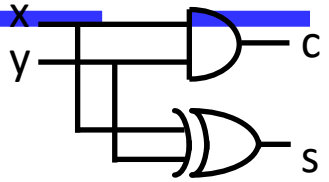


$c = xy$



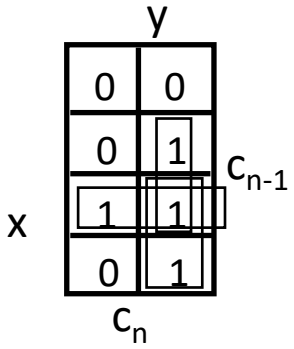
$s = xy' + x'y$

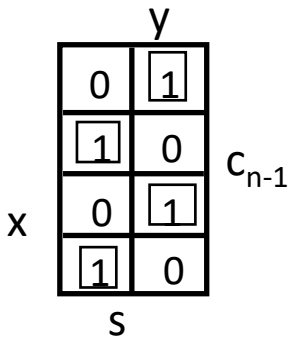
$= x \oplus y$



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



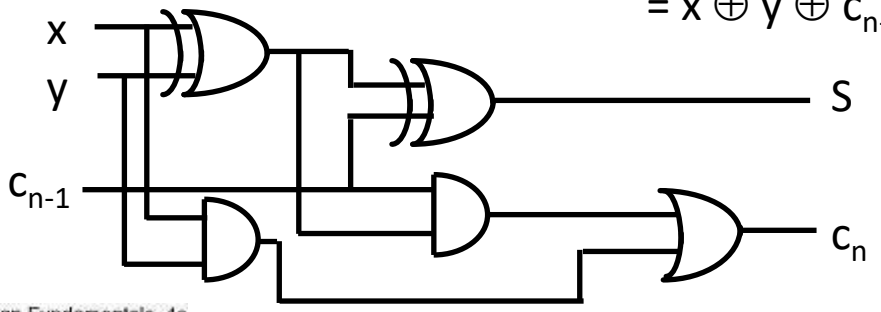


$c_n = xy + xc_{n-1} + yc_{n-1}$

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

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

$= x \oplus y \oplus c_{n-1} = (x \oplus y) \oplus c_{n-1}$



COMBINATIONAL LOGIC CIRCUITS

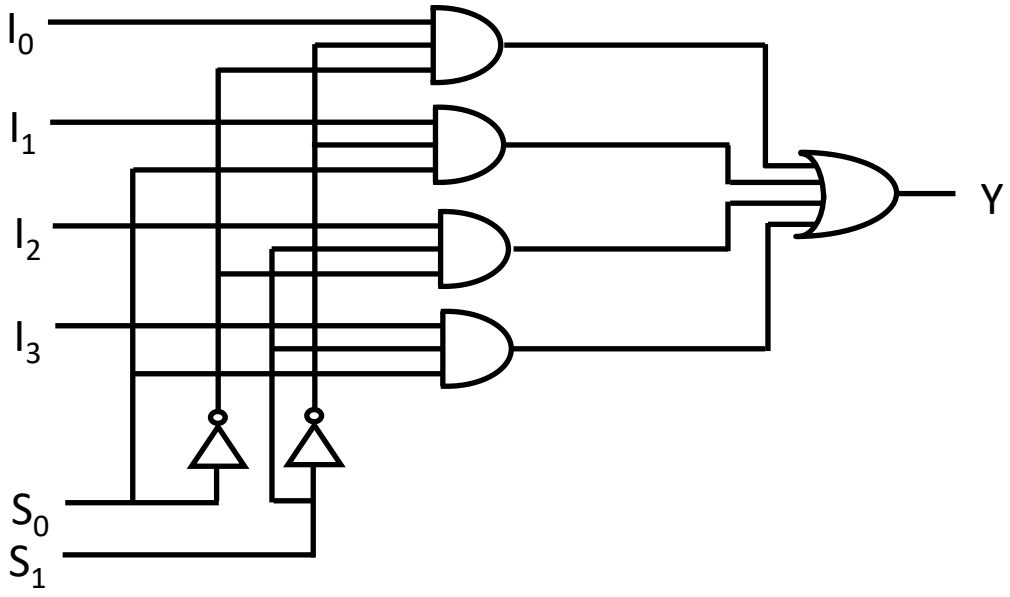
Other Combinational Circuits

- Multiplexer
- Encoder
- Decoder
- Parity Checker
- Parity Generator
- etc

MULTIPLEXER

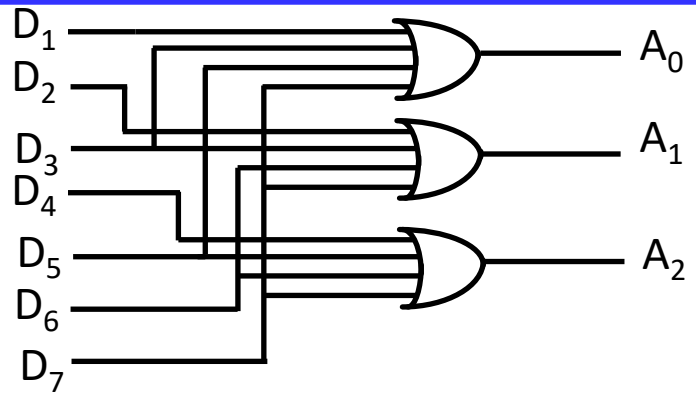
4-to-1 Multiplexer

Select		Output
S_1	S_0	Y
0	0	I_0
0	1	I_1
1	0	I_2
1	1	I_3



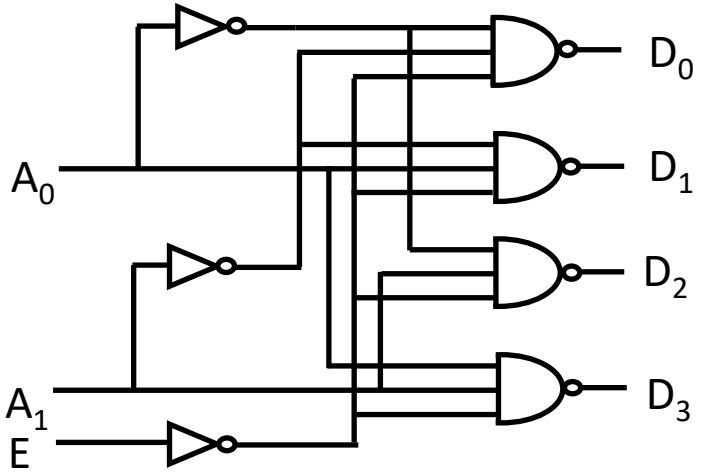
ENCODER/DECODER

Octal-to-Binary Encoder



2-to-4 Decoder

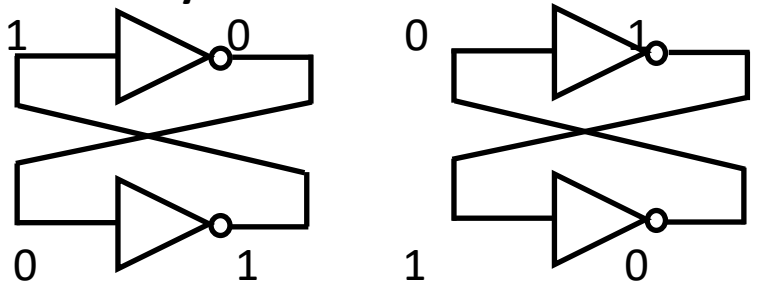
E	A_1	A_0	D_0	D_1	D_2	D_3
0	0	0	0	1	1	1
0	0	1	1	0	1	1
0	1	0	1	1	0	1
0	1	1	1	1	1	0
1	d	d	1	1	1	1



FLIP FLOPS

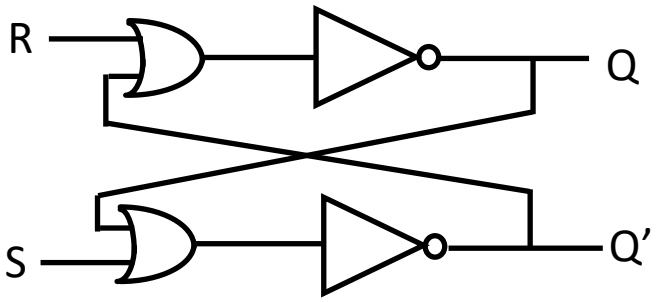
Characteristics

- 2 stable states
- Memory capability
- Operation is specified by a Characteristic Table



0-state **1-state**

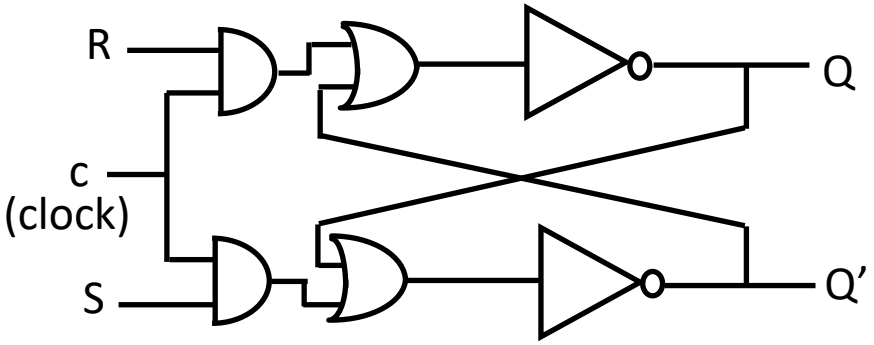
In order to be used in the computer circuits, state of the flip flop should have input terminals and output terminals so that it can be set to a certain state, and its state can be read externally.



S	R	Q(t+1)
0	0	Q(t)
0	1	0
1	0	1
1	1	indeterminate (forbidden)

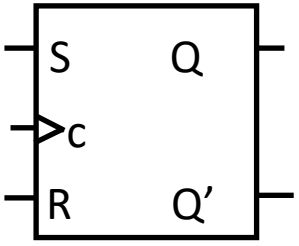
CLOCKED FLIP FLOPS

In a large digital system with many flip flops, operations of individual flip flops are required to be synchronized to a clock pulse. Otherwise, the operations of the system may be unpredictable.

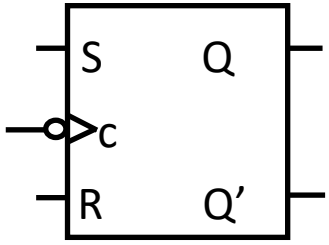


Clock pulse allows the flip flop to change state only when there is a clock pulse appearing at the c terminal.

We call above flip flop a Clocked RS Latch, and symbolically as

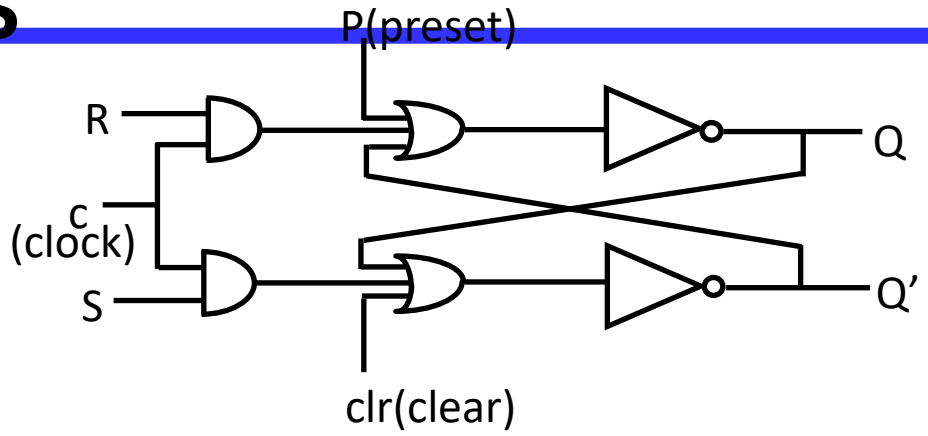


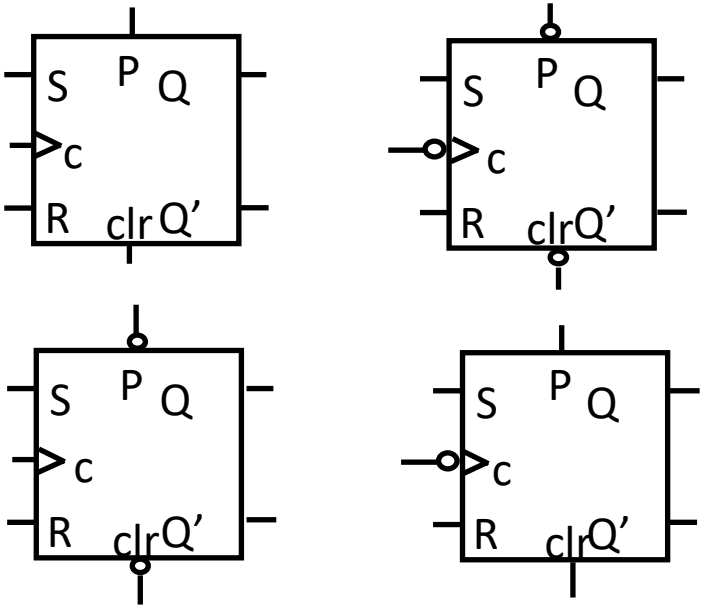
operates when clock is high



operates when clock is low

RS-LATCH WITH PRESET AND CLEAR INPUTS

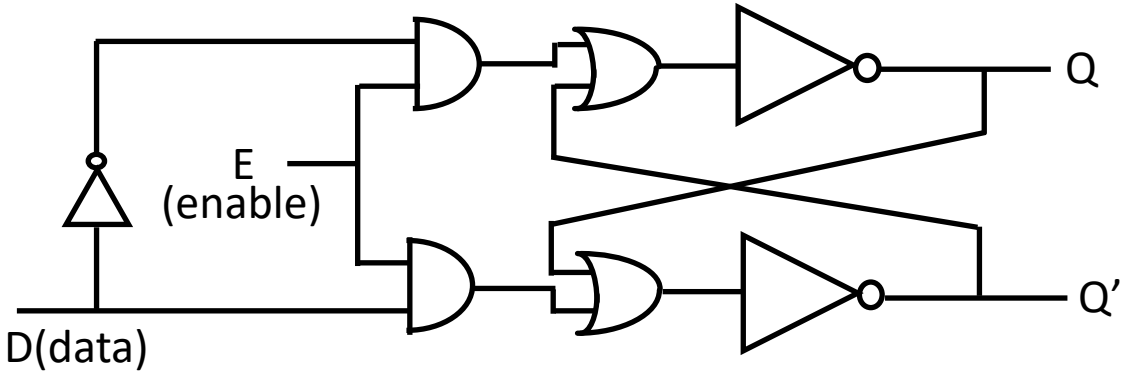


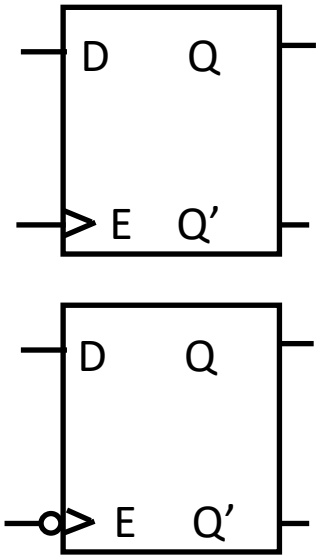


D-LATCH

D-Latch

Forbidden input values are forced not to occur by using an inverter between the inputs





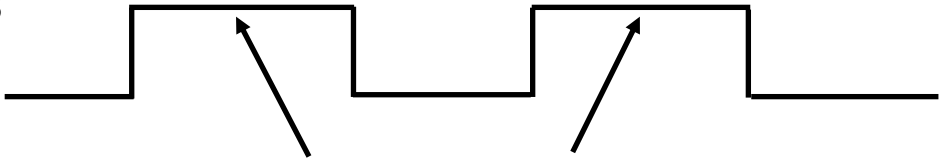
D	Q(t+1)
0	0
1	1

EDGE-TRIGGERED FLIP FLOPS

Characteristics

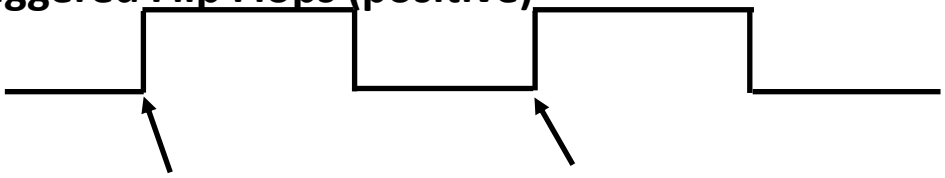
- State transition occurs at the rising edge or falling edge of the clock pulse

Latches



respond to the input only during these periods

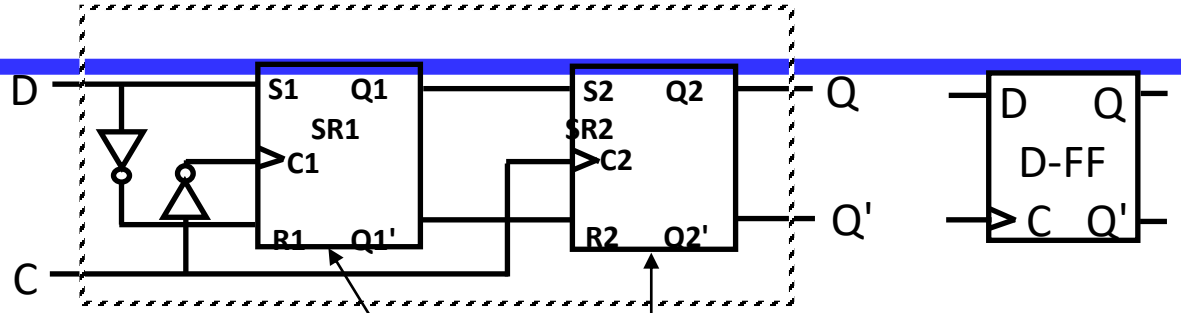
Edge-triggered Flip Flops (positive)

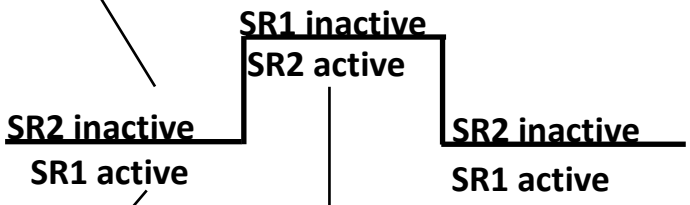


respond to the input only at this time

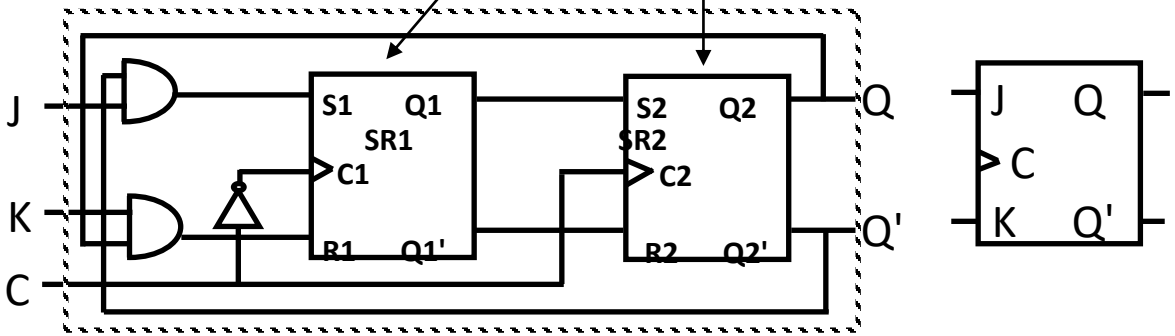
POSITIVE EDGE-TRIGGERED

D-Flip Flop





JK-Flip Flop



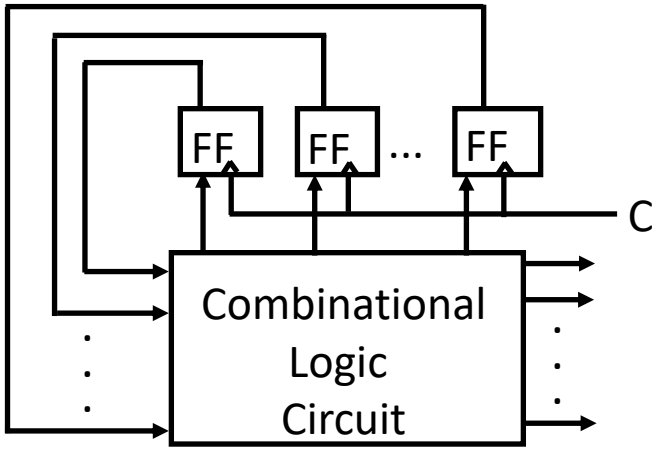
T-Flip Flop: JK-Flip Flop whose J and K inputs are tied together to make T input. Toggles whenever there is a pulse on T input.

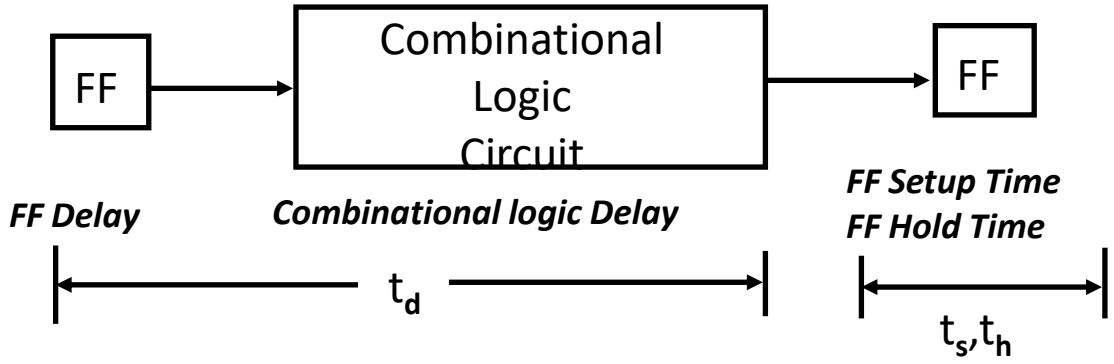
CLOCK PERIOD

Clock period determines how fast the digital circuit operates.

How can we determine the clock period ?

Usually, digital circuits are sequential circuits which has some flip flops





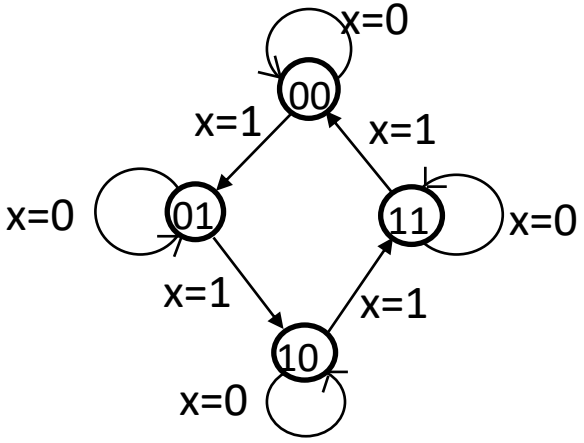
clock period $T = t_d + t_s + t_h$

DESIGN EXAMPLE

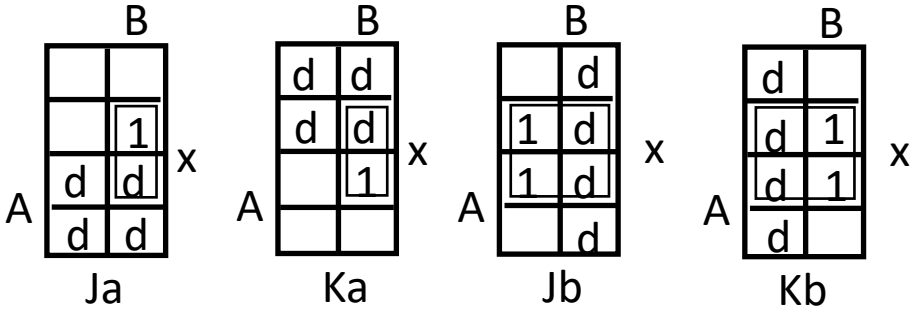
Design Procedure:

Specification ⇒ State Diagram ⇒ State Table ⇒
Excitation Table ⇒ Karnaugh Map ⇒ Circuit Diagram

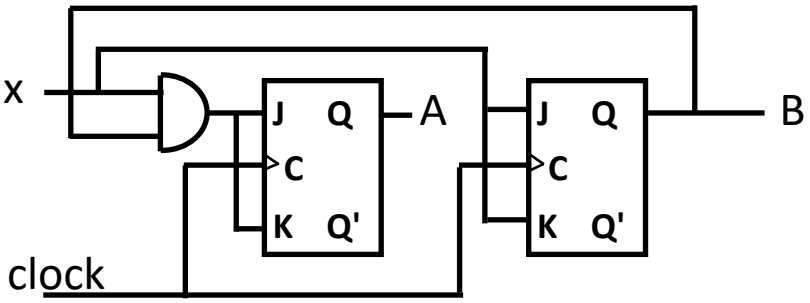
Example: 2-bit Counter -> 2 FF's



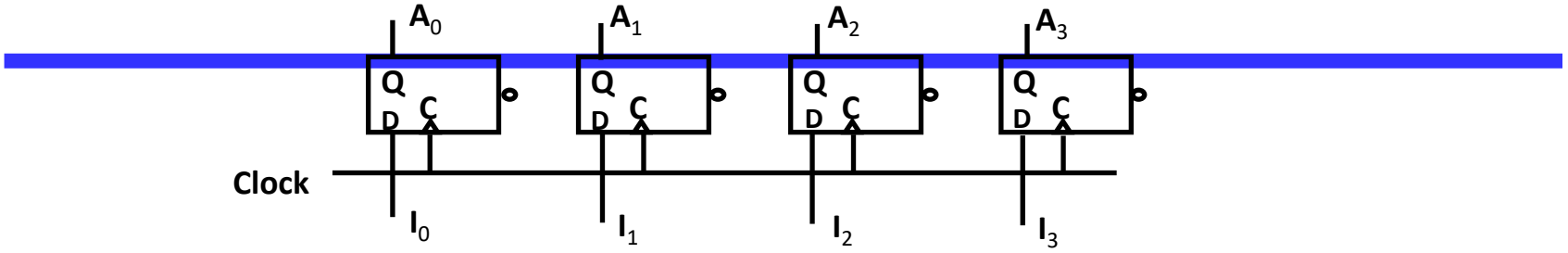
current state		input	next state		FF inputs			
A	B	x	A	B	Ja	Ka	Jb	Kb
0	0	0	0	0	0	d	0	d
0	0	1	0	1	0	d	1	d
0	1	0	0	1	0	d	d	0
0	1	1	1	0	1	d	d	1
1	0	0	1	0	d	0	0	d
1	0	1	1	1	d	0	1	d
1	1	0	1	1	d	0	d	0
1	1	1	0	0	d	1	d	1



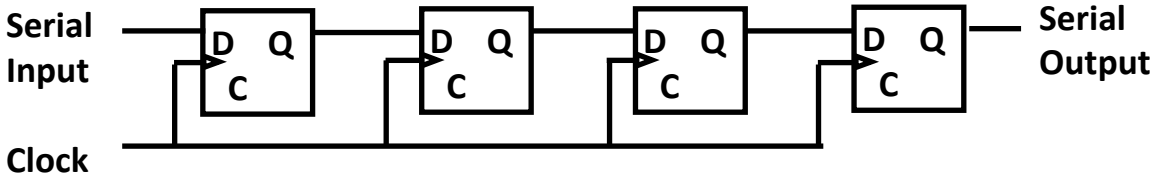
Ja = Bx Ka = Bx Jb = x Kb = x



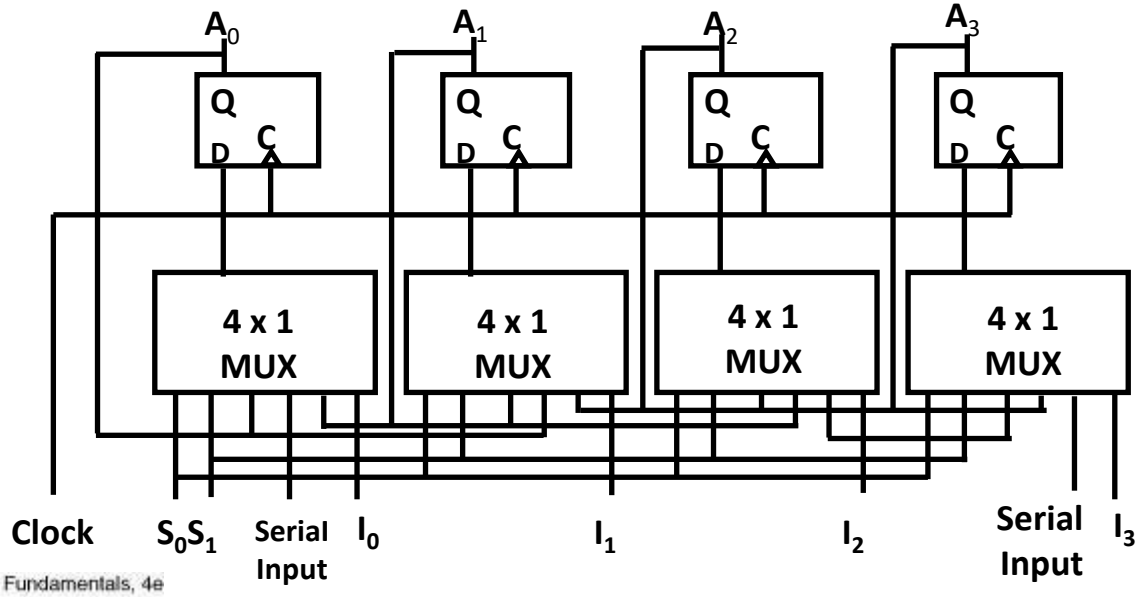
SEQUENTIAL CIRCUITS - Registers



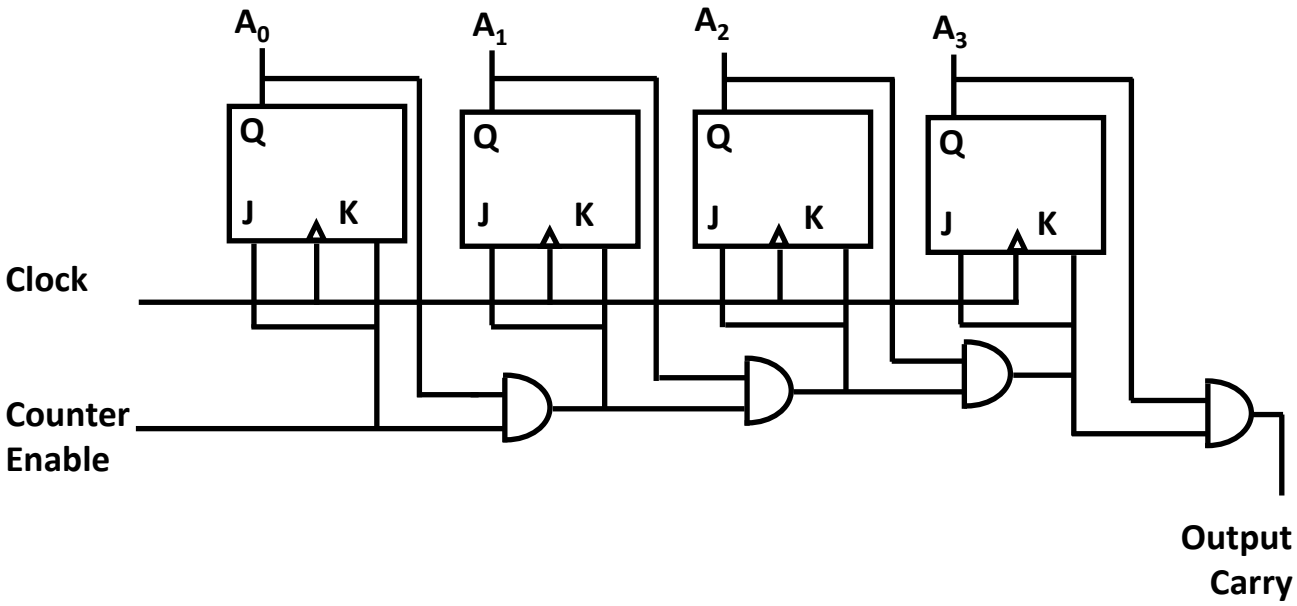
Shift Registers



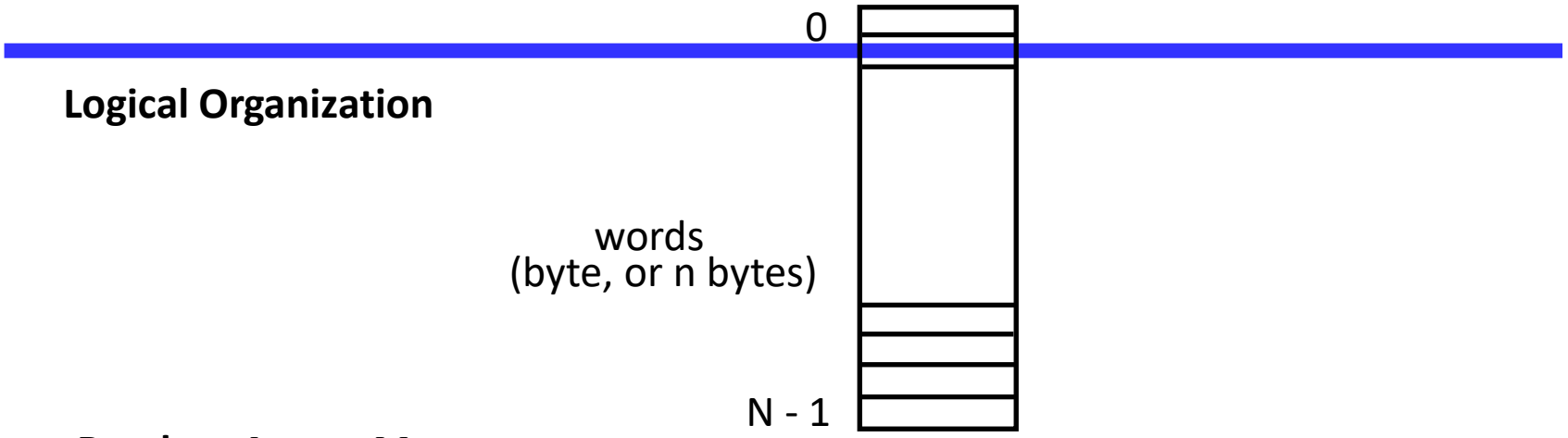
Bidirectional Shift Register with Parallel Load



SEQUENTIUAL CIRCUITS - Counters



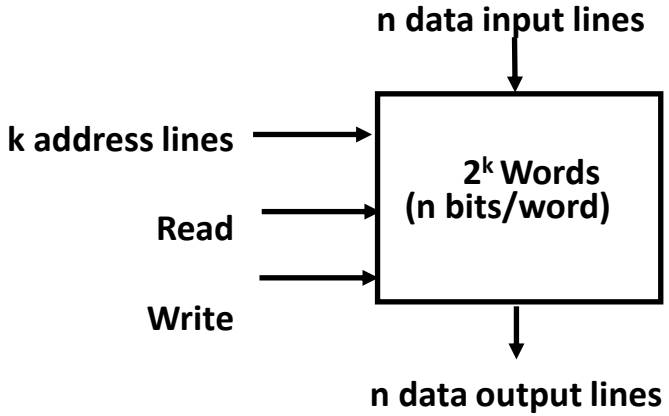
MEMORY COMPONENTS



Logical Organization

Random Access Memory

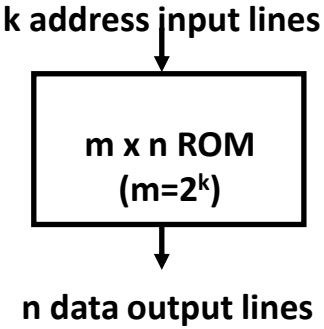
- **Each word has a unique address**
- **Access to a word requires the same time independent of the location of the word**
- **Organization**



READ ONLY MEMORY(ROM)

Characteristics

- **Perform read operation only, write operation is not possible**
- **Information stored in a ROM is made permanent during production, and cannot be changed**
- **Organization**



Information on the data output line depends only on the information on the address input lines.

--> Combinational Logic Circuit

$X_0 = A'B' + B'C$
$X_1 = A'B'C + A'BC'$
$X_2 = BC + AB'C'$
$X_3 = A'BC' + AB'$
$X_4 = AB$

Canonical minterms

$X_0 = A'B'C' + A'B'C + AB'C$
$X_1 = A'B'C + A'BC'$
$X_2 = A'BC + AB'C' + ABC$
$X_3 = A'BC' + AB'C' + AB'C$
$X_4 = ABC' + ABC$

address ***Output***

ABC	X_0	X_1	X_2	X_3	X_4
000	1	0	0	0	0
001	1	1	0	0	0
010	0	1	0	1	0
011	0	0	1	0	0
100	0	0	1	1	0
101	1	0	0	1	0
110	0	0	0	0	1
111	0	0	1	0	1

TYPES OF ROM

ROM

- Store information (function) during production
- Mask is used in the production process
- Unalterable
- Low cost for large quantity production --> used in the final products

PROM (Programmable ROM)

- Store info electrically using PROM programmer at the user's site
- Unalterable
- Higher cost than ROM -> used in the system development phase
  -> Can be used in small quantity system

EPROM (Erasable PROM)

- Store info electrically using PROM programmer at the user's site
- Stored info is erasable (alterable) using UV light (electrically in some devices) and rewriteable
- Higher cost than PROM but reusable --> used in the system development phase. Not used in the system production due to eras ability

INTEGRATED CIRCUITS

Classification by the Circuit Density

SSI - several (less than 10) independent gates
MSI - 10 to 200 gates; Perform elementary digital functions;
Decoder, adder, register, parity checker, etc
LSI - 200 to few thousand gates; Digital subsystem
Processor, memory, etc
VLSI - Thousands of gates; Digital system
Microprocessor, memory module

Classification by Technology

TTL - Transistor-Transistor Logic
Bipolar transistors
NAND
ECL - Emitter-coupled Logic
Bipolar transistor
NOR
MOS - Metal-Oxide Semiconductor
Unipolar transistor
High density
CMOS - Complementary MOS
Low power consumption