Chapter 1
Binary Systems

Digital Systems

Binary Numbers

Number Base Conversion

Octal and Hexadecimal Number

Complements

Signed Binary Numbers

Binary Codes

Binary Storage and Registers

Binary Logic

Digital Logic Design

Digital

- Concerned with the interconnection among digital components and modules
  - General Purpose Computer
  - Embedded Systems : Internet of Things (IoT)

Logic Design

- Deals with the basic concepts and tools used to design digital hardware consisting of logic circuits
  - Circuits to perform Arithmetic and Logic operations

Bits and Pieces of DLD History

- **George Boole**
  - **Mathematical Analysis of Logic(1847)**
  - **An Investigation of Laws of Thoughts; Mathematical Theories of Logic and Probabilities (1854)**
- **Claude Shannon**
  - **Rediscovered the Boole**
  - **“A Symbolic Analysis of Relay and Switching Circuits “**
  - **Boolean Logic and Boolean Algebra were Applied to Digital Circuitry**
- **Beginning of the Digital Age/Computer Age**
  - **World War II**
  - **Computers as Calculating Machines**
  - **Arlington (State Machines) “ Control “**

Motivation

- **Microprocessors/Microelectronics have revolutionized our world**
  - **Cell phones, internet, rapid advances in medicine, etc.**
- **The semiconductor industry has grown tremendously (Moore's Law)**

Digital Systems & Binary Numbers

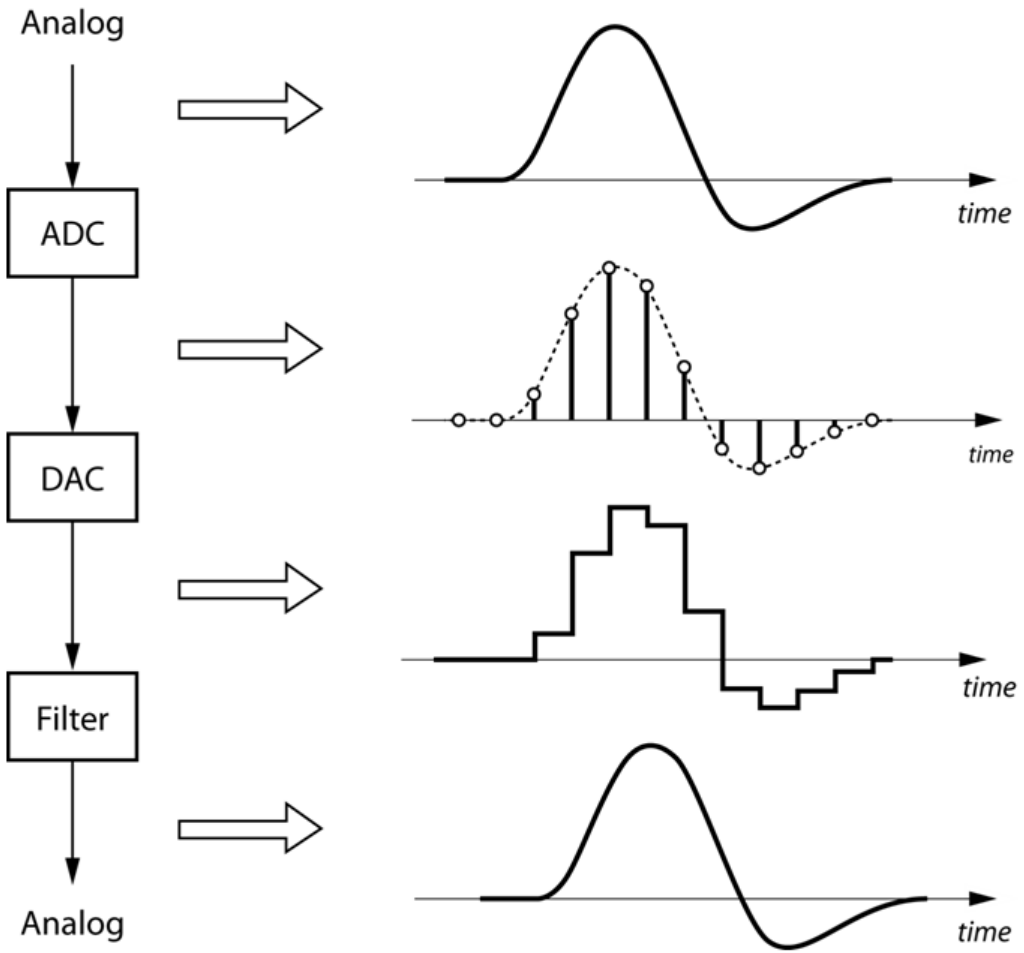
- **Digital/Information Age**
- **Digital Computers**
  - **General Purpose – Personal, Mainframe & Super Computers**
  - **Many scientific, industrial and commercial applications**
  - **Space program**
- **Digital Systems**
  - **Telephone switching exchanges**
  - **Digital Camera, Mobile Phone**
  - **Electronic Calculator, PDA, Tablet, Pad**
  - **Digital TV**
- **Discrete information-processing systems**
- **Why binary?**
  - **Reliability: A transistor circuit is either on or off (two stable states)**

Analog & Digital Signals/Systems

- **Analog Systems**
  - **Physical quantities or signals may vary continuously**
- **Digital/Discreet Information Processing Systems**
  - **Physical quantities or signals can assume discreet values only**
  - **Greater Accuracy**
  - **Higher Noise immunity**
  - **Low Bandwidth requirement**

Analog to Digital & Digital to Analog Conversion

Sampling, Quantization & Coding



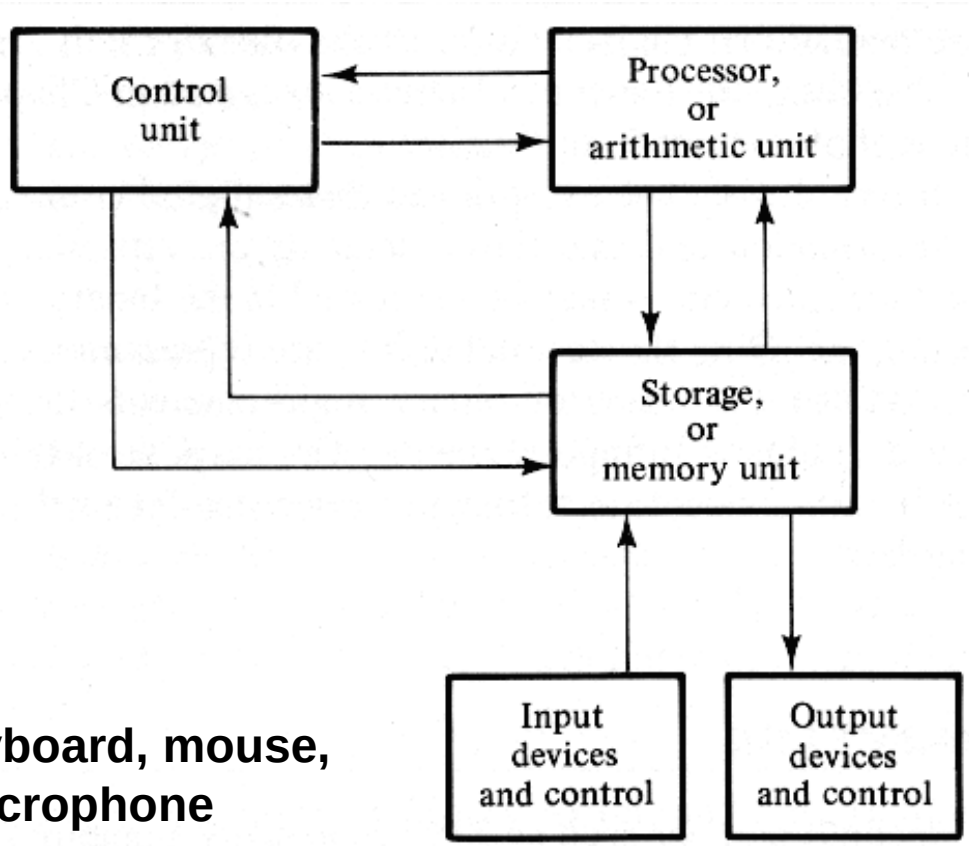
Analog vs Digital Signals/Systems

	Analog	Digital
Technology:	Analog technology records waveforms as they are.	Converts analog waveforms into set of numbers and records them. The numbers are converted into voltage stream for representation.
Uses:	Can be used in various computing platforms and under operating systems like Linux, Unix, Mac OS and Windows.	Computing and electronics
Signal:	Analog signal is a continuous signal which transmits information as a response to changes in physical phenomenon.	Digital signals are discrete time signals generated by digital modulation.
Representation:	Uses continuous range of values to represent information.	Uses discrete or discontinuous values to represent information.
Memory unit:	not required	required
applications:	Thermometer	PCs, PDAs
Data transmissions:	not of high quality	high quality
Result:	not very accurate	accurate
Storage capacity:	limited	high
Process:	processed using OPAMP which uses electronic circuits	using microprocessor which uses logic circuits
Respose to Noise:	More likely to get affected reducing accuracy	Less affected since noise response are analog in nature
Waves:	Denoted by sine waves	Denoted by square waves
Example:	human voice in air	electronic devices

Digital Computer

- **The digital computer is one of the most well known digital systems.**
- **The digital computer consists of the following components:**
  - **Memory unit**
  - **Central processing unit**
  - **Input and output units**
- **The digital computer can perform both arithmetic and logical operations.**

Digital Computer



Inputs: **Keyboard, mouse, modem, microphone**

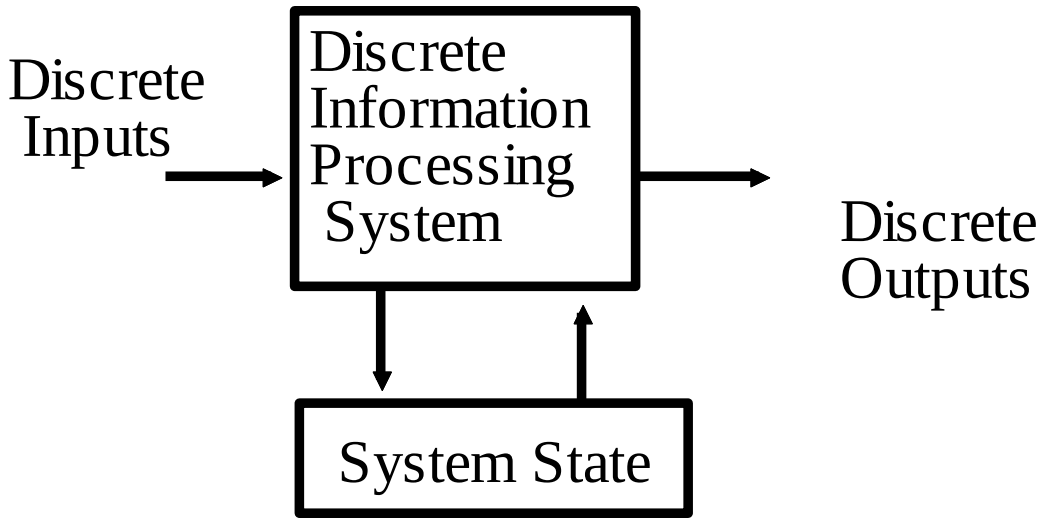
Outputs: **CRT, LCD, modem, speakers**

- **stored program**
- **control unit**
- **arithmetic computations and logical operations**

Digital Systems

- Digital Systems **represent systems that understand, represent and manipulate discrete elements.**
  - **A** discrete element **is any set that has a finite number of elements, for example 10 decimal digits, 26 letters of the alphabet, etc.**
- **Discrete elements are represented by** signals**, such as electrical signals (voltages and currents)**
- **The signals in most electronic digital systems use two discrete values, termed** binary**.**
- Digital System **takes a set of discrete information inputs and discrete internal information (system state) and generates a set of discrete information outputs.**

Digital System



Signals

- A collection of information variables mapped to some physical quantity
- For digital systems, the quantities take on discrete values
- Two level, or binary values are the most prevalent values in digital systems
- The binary values are represented abstractly by digits 0 and 1
- Other physical signals represented by 1 and 0?
  - CPU — Voltage
  - Disk — Magnetic Field Direction
  - CD — Surface Pits/Light
  - Dynamic RAM — Electrical Charge

Why Digital Components?

- **Why do we choose to use digital components?**
  - **The main reason for using digital components is that they can easily be programmed, allowing a single hardware unit to be used for many different purposes**
  - **Advances in circuit technology decrease the price of technology dramatically**
  - **Digital integrated circuits can perform at speeds of hundreds of millions of operations per second**
  - **Error-checking and correction can be used to ensure the reliability of the machine**

Binary Digits

- **A binary digit, called a bit, is represented by one of two values: 0 or 1.**
  - **Discrete elements can be represented by groups of bits called binary codes. For example the decimal digits 0 to 9 are represented as follows:**

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

Differing Bases

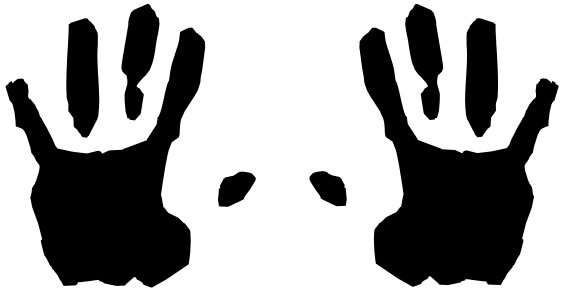
- **In order to represent numbers of different bases, we surround a number in parenthesis and then place a subscript with the base of the number**
  - **A decimal number $(9233)_{10}$**
  - **A binary number $(11011)_{2}$**
  - **A base 5 number $(3024)_{5}$**
- **Decimal number digits are 0 through 9**
- **Binary number digits are 0 through 1**
- **Base (radix) *r* number digits are 0 through *r* - 1**

Commonly Occurring Bases

Name	Radix	Digits (0 through r-1)
Binary	2	0,1
Octal	8	0,1,2,3,4,5,6,7
Decimal	10	0,1,2,3,4,5,6,7,8,9
Hexadecimal	16	0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F

Decimal Numbers

- **A decimal number such as 5723 represents a quantity equal to:**
  - **5 thousands**
  - **7 hundreds**
  - **2 tens**
  - **3 ones**



- **Or, it can be written as:**
  - **$5 \times 10^3 + 7 \times 10^2 + 2 * 10^1 + 3 * 10^0$**
- **The 5, 7, 2, and 3 represent coefficients.**
- **The decimal number system is said to be of base or radix 10 because it uses the 10 digits (0..9) and the coefficients are multiplied by powers of 10.**

Binary Numbers

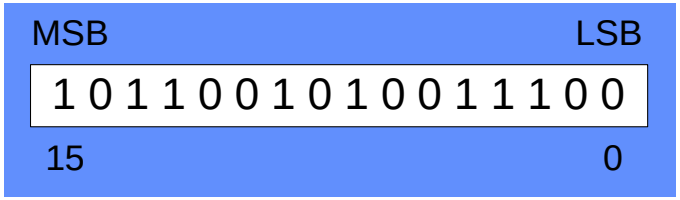
- **The binary system contains only two values in the allowed coefficients (0 and 1).**
- **The binary system uses powers of 2 as the multipliers for the coefficients.**
- **For example, we can represent the binary number 10111.01 as:**
  - $1 \times 2^4 + 0 \times 2^3 + 1 \times 2^2 + 1 \times 2^1 + 1 \times 2^0 + 0 \times 2^{-1} + 1 \times 2^{-2} = 23.25$

Understanding Binary Numbers

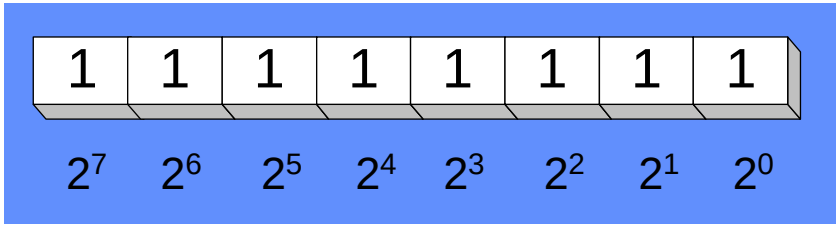
- **Binary numbers are made of binary digits (bits):**
  - **0 and 1**
- **How many items does a binary number represent?**
  - $(1011)_2 = 1x2^3 + 0x2^2 + 1x2^1 + 1x2^0 = (11)_{10}$
- **What about fractions?**
  - $(110.10)_2 = 1x2^2 + 1x2^1 + 0x2^0 + 1x2^{-1} + 0x2^{-2}$
- **Group of eight bits is called a *byte***
  - $(11001001)_2$
- **Group of four bits is called a *nibble***
  - $(1101)_2$
- **Group of sixteen bits is called a *half word***
  - $(1011011010011001)_2$
- **Group of thirty two bits is called a *word***
- **Group of sixty four bits is called a *double word***

Understanding Binary Numbers (Contd..)

- **MSB – most significant bit**
- **LSB – least significant bit**
- **Bit numbering**



- **Each digit (bit) is either 1 or 0**
- **Each bit represents a power of 2**



Why Use Binary Numbers?

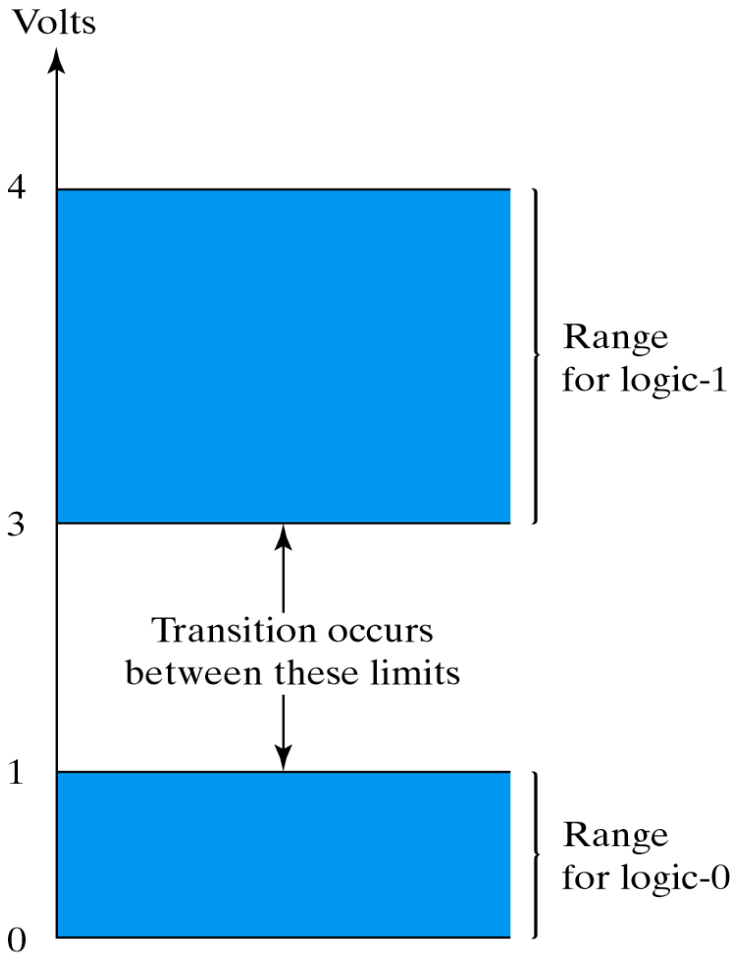
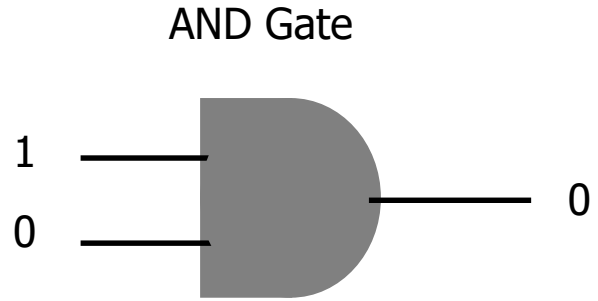


Fig. 1-3 Example of binary signals

- **Easy to represent 0 and 1 using electrical values.**
- **Possible to tolerate noise.**
- **Easy to transmit data**
- **Easy to build binary circuits.**



Powers of Two

n	2^n	n	2^n	n	2^n
0	1	8	256	16	65,536
1	2	9	512	17	131,072
2	4	10	1,024	18	262,144
3	8	11	2,048	19	524,288
4	16	12	4,096	20	1,048,576
5	32	13	8,192	21	2,097,152
6	64	14	16,384	22	4,194,304
7	128	15	32,768	23	8,388,608

n	2^n	n	2^n	n	2^n	n	2^n
0	1	16	65,536	32	4,294,967,296	48	281,474,976,710,656
1	2	17	131,072	33	8,589,934,592	49	562,949,953,421,312
2	4	18	262,144	34	17,179,869,184	**50**	**1,125,899,906,842,620**
3	8	19	524,288	35	34,359,738,368	51	2,251,799,813,685,250
4	16	**20**	**1,048,576**	36	68,719,476,736	52	4,503,599,627,370,500
5	32	21	2,097,152	37	137,438,953,472	53	9,007,199,254,740,990
6	64	22	4,194,304	38	274,877,906,944	54	18,014,398,509,482,000
7	128	23	8,388,608	39	549,755,813,888	55	36,028,797,018,964,000
8	256	24	16,777,216	**40**	**1,099,511,627,776**	56	72,057,594,037,927,900
9	512	25	33,554,432	41	2,199,023,255,552	57	144,115,188,075,856,000
10	**1,024**	26	67,108,864	42	4,398,046,511,104	58	288,230,376,151,712,000
11	2,048	27	134,217,728	43	8,796,093,022,208	59	576,460,752,303,423,000
12	4,096	28	268,435,456	44	17,592,186,044,416	**60**	**1,152,921,504,606,850,000**
13	8,192	29	536,870,912	45	35,184,372,088,832	61	2,305,843,009,213,690,000
14	16,384	**30**	**1,073,741,824**	46	70,368,744,177,664	62	4,611,686,018,427,390,000
15	32,768	31	2,147,483,648	47	140,737,488,355,328	63	9,223,372,036,854,780,000

n	2^n	n	2^n	n	2^n	n	2^n
0	1	16	65,536	32	4,294,967,296	48	281,474,976,710,656
1	2	17	131,072	33	8,589,934,592	49	562,949,953,421,312
2	4	18	262,144	34	17,179,869,184	**50**	**1,125,899,906,842,620**
3	8	19	524,288	35	34,359,738,368	51	2,251,799,813,685,250
4	16	**20**	**1,048,576**	36	68,719,476,736	52	4,503,599,627,370,500
5	32	21	2,097,152	37	137,438,953,472	53	9,007,199,254,740,990
6							
7							
8							
9							
10							
11							
12							
13	8,192	29	536,870,912	45	35,184,372,088,832	61	2,305,843,009,213,690,000
14	16,384	**30**	**1,073,741,824**	46	70,368,744,177,664	62	4,611,686,018,427,390,000
15	32,768	31	2,147,483,648	47	140,737,488,355,328	63	9,223,372,036,854,780,000

32 bits wide word can store an unsigned magnitude
4,294,967,295 = (4 GB -1)
64 bits wide word can store an unsigned magnitude
18,446,744,073,709,600,000 = (16 PB -1)

Important powers of 2

2^{10} is referred to as Kilo, called "K"

2^{20} is referred to as Mega, called "M"

2^{30} is referred to as Giga, called "G"

2^{40} is referred to as Tera, called “T”

2^{50} is referred to as Peta, called “P”

2^{60} is referred to as Exa, called “E”

2^{70} is referred to as Yotta called “Y”

Octal Numbers

- **The octal number system is a base-8 system that contains the coefficient values of 0 to 7**
- **The octal system uses powers of 8 as the multipliers for the coefficients**
- **For example, we can represent the octal number $(72032)_8$ as:**

$$7 \times 8^4 + 2 \times 8^3 + 0 \times 8^2 + 3 \times 8^1 + 2 \times 8^0 = (29722)_{10}$$

Hexadecimal Numbers

- **The hexadecimal number system is a base-16 system that contains the coefficient values of 0 to 9 and A to F**
- **The letters A through F represent the coefficient values of 10, 11, 12, 13, 14, and 15, respectively**
- **The hexadecimal system uses powers of 16 as the multipliers for the coefficients**
- **For example, we can represent the hexadecimal number $(C34D)_{16}$ as:**

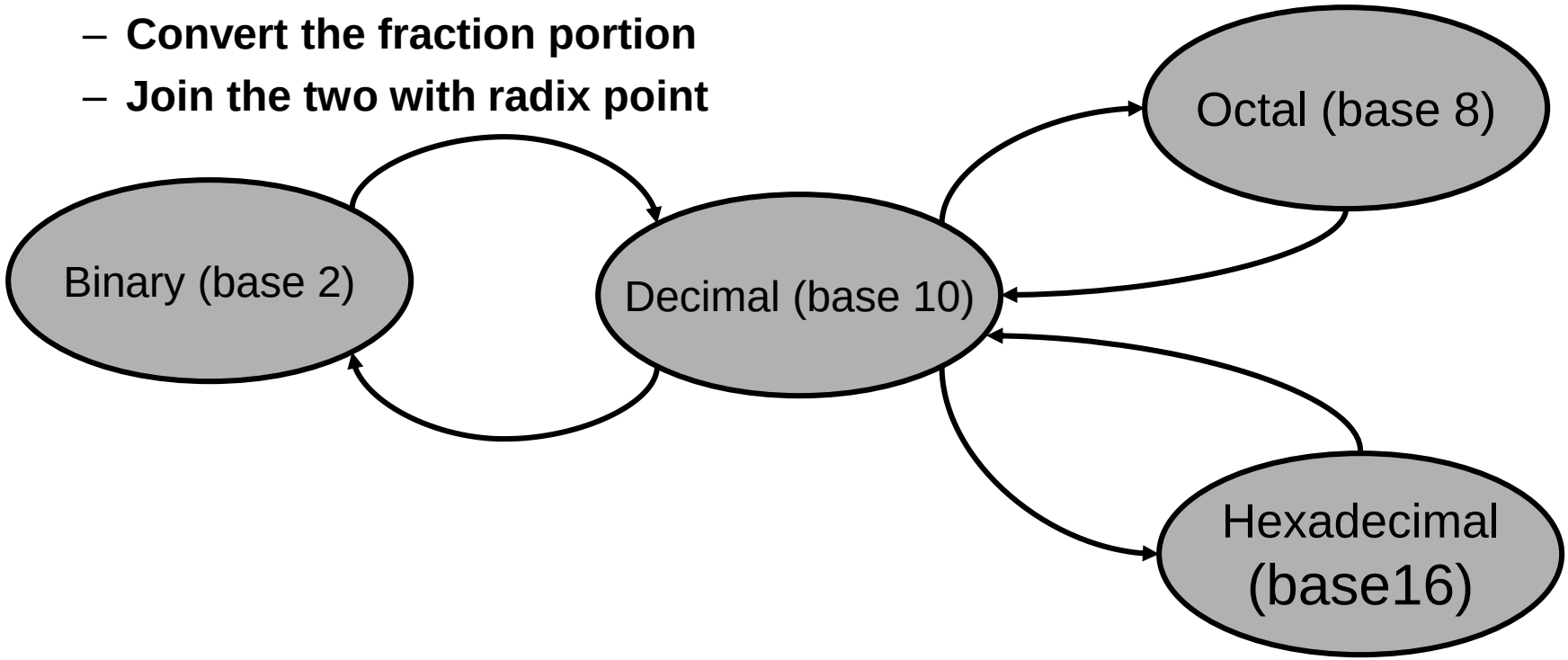
$$12 \times 16^3 + 3 \times 16^2 + 4 \times 16^1 + 13 \times 16^0 = (49997)_{10}$$

Number Examples

Decimal (base 10)	Binary (base 2)	Octal (base 8)	Hexadecimal (base 16)
00	0000	00	0
01	0001	01	1
02	0010	02	2
03	0011	03	3
04	0100	04	4
05	0101	05	5
06	0110	06	6
07	0111	07	7
08	1000	10	8
09	1001	11	9
10	1010	12	A
11	1011	13	B
12	1100	14	C
13	1101	15	D
14	1110	16	E
15	1111	17	F

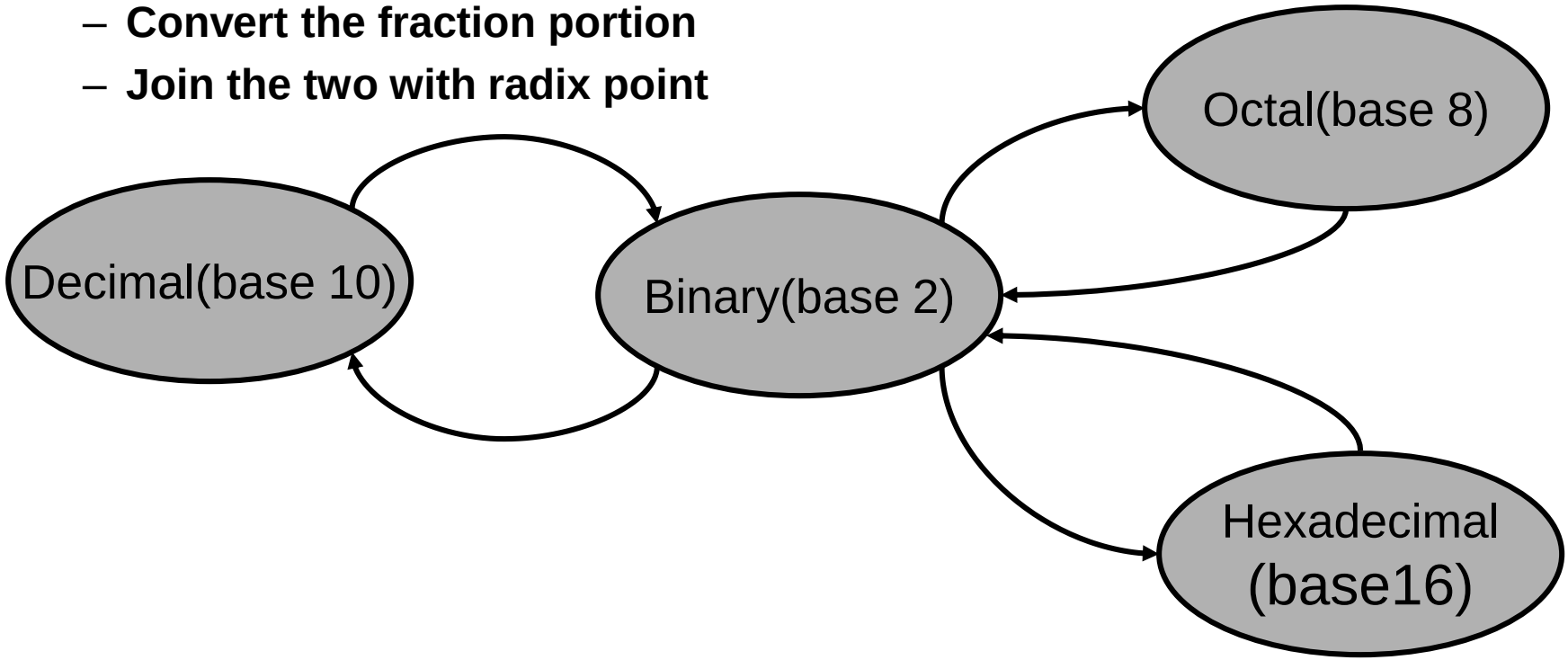
Conversion between bases

- **To convert from one base to other:**
  - **Convert the integer portion**
  - **Convert the fraction portion**
  - **Join the two with radix point**



Conversion between bases

- **To convert from one base to other:**
  - **Convert the integer portion**
  - **Convert the fraction portion**
  - **Join the two with radix point**



r-Decimal Conversion

- **Conversion of a number in base r to decimal is done by expanding the number in a power series and adding all the terms**
- **For example, $(C34D)_{16}$ is converted to decimal:**

  $12 \times 16^3 + 3 \times 16^2 + 4 \times 16^1 + 13 \times 16^0 = (49997)_{10}$
- **$(11010.11)_2$ is converted to decimal:**

  $1 \times 2^4 + 1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 0 \times 2^0 + 1 \times 2^{-1} + 1 \times 2^{-2} = (26.75)_{10}$
- **In general $(Number)_r$ =** $\left(\sum_{i=0}^{i=n-1} a_i \bullet r^i\right) + \left(\sum_{j=-m}^{j=-1} a_j \bullet r^j\right)$

  (Integer Portion) + (Fraction Portion)

Decimal-r Conversion

- **If a decimal number has a radix point, it is necessary to separate the number into an integer part and a fraction part.**
- **The conversion of a decimal integer into a number in base-r is done by dividing the number and all successive quotients by r and accumulating the remainders in reverse order of computation.**
- **For example, to convert decimal 13 to binary:**

	Integer Quotient		Remainder	Coefficient
13/2 =	6	+	1	$a_0 = 1$
6/2 =	3	+	0	$a_1 = 0$
3/2 =	1	+	1	$a_2 = 1$
1/2 =	0	+	1	$a_3 = 1$

Answer $(13)_{10} = (a_3\, a_2\, a_1\, a_0)_2 = (1101)_2$

Example

- **Convert $(37)_{10}$ to binary**

Division	Quotient	Remainder
37 / 2	18	1
18 / 2	9	0
9 / 2	4	1
4 / 2	2	0
2 / 2	1	0
1 / 2	0	1

$(37)_{10} = 100101$

Decimal-r Conversion (converting fraction)

- To convert the fraction portion repeatedly multiply the fraction by the radix and save the integer digits that result
- The process is continued until the fraction becomes 0 or the number of digits have sufficient accuracy
- The new radix fraction digits are the integer digits in computed order
- For example convert fraction $(0.6875)_{10}$ to base 2

0.6875 * 2 = 1.3750 integer = 1
0.3750 * 2 = 0.7500 integer = 0
0.7500 * 2 = 1.5000 integer = 1
0.5000 * 2 = 1.0000 integer = 1

Answer = $(0.1011)_2$

Example

- **When converting fractions, we must use multiplication rather than division**
- **The new radix fraction digits are the integer digits in *computed order***

		Integer		Fraction	Coefficent
0.8432 X 2	=	1	+	0.6864	$a_{-1} = 1$
0.6864 X 2	=	1	+	0.3728	$a_{-2} = 1$
0.3728 X 2	=	0	+	0.7456	$a_{-3} = 0$
0.7456 X 2	=	1	+	0.4912	$a_{-4} = 1$
0.4912 X 2	=	0	+	0.9824	$a_{-5} = 0$
0.9824 X 2	=	1	+	0.9648	$a_{-6} = 1$
0.9648 X 2	=	1	+	0.9296	$a_{-7} = 1$

Continue until fraction becomes 0 or until sufficient accuracy.

$$(0.8432)_{10} = (0.a_{-1}a_{-2}a_{-3}a_{-4}a_{-5}a_{-6}a_{-7})_2 = (0.1101011)_2$$

Example:

- **Convert 0.8125 decimal to binary**
  - **To convert the decimal 0.8125 to binary, we multiply by the radix 2**
  - **$(0.1101)_2$**

$$
\begin{array}{r}
.8125 \\
\times \quad 2 \\
\hline
1.6250
\end{array}
$$

$$
\begin{array}{r}
.6250 \\
\times \quad 2 \\
\hline
1.2500
\end{array}
$$

$$
\begin{array}{r}
.2500 \\
\times \quad 2 \\
\hline
0.5000
\end{array}
$$

$$
\begin{array}{r}
.5000 \\
\times \quad 2 \\
\hline
1.0000
\end{array}
$$

Decimal to Octal Conversion

- **In converting decimal to octal we must divide by 8**

	Integer Quotient		Remainder	Coefficent
35 / 8 =	4	+	3/8	$a_0 = 3$
4 / 8 =	0	+	4/8	$a_1 = 4$

$$(35)_{10} = (a_1 a_0)_8 = (43)_8$$

Converting Fractions (Decimal to Octal)

- **Decimal to Octal fraction conversion takes the same approach but it multiplies by the base 8**

		Integer		Fraction	Coefficent
0.8432 X 8	=	6	+	0.7456	$a_{-1} = 6$
0.7456 X 8	=	5	+	0.9648	$a_{-2} = 5$
0.9648 X 8	=	7	+	0.7184	$a_{-3} = 7$
0.7184 X 8	=	5	+	0.7472	$a_{-4} = 5$
0.7472 X 8	=	5	+	0.9776	$a_{-5} = 5$
0.9776 X 8	=	7	+	0.8208	$a_{-6} = 7$
0.8208 X 8	=	6	+	0.5664	$a_{-7} = 6$

Continue until fraction becomes 0 or until sufficient accuracy.

$$(0.8432)_{10} = (0.a_{-1}a_{-2}a_{-3}a_{-4}a_{-5}a_{-6}a_{-7})_8 = (0.6575576)_8$$

Converting Decimal to Hexadecimal

- **The conversion of a decimal integer into hexadecimal is done by dividing the number and all successive quotients by 16 and accumulating the remainders in reverse order of computation**

Division	Quotient	Remainder
422 / 16	26	6
26 / 16	1	A
1 / 16	0	1

$$(422)_{10} = (1A6)_{16}$$

Binary, Octal and Hexadecimal

- **Conversions between binary, octal and hexadecimal have an easier conversion method.**
  - **Each octal digit represents 3 binary digits**
  - **Each hexadecimal digit represents 4 binary digits**

$$(11\ 010\ 101\ 111\ 111\ .\ 101\ 110\ 01)_2 = (32577.561)_8$$
$$\quad 3 \quad 2 \quad 5 \quad 7 \quad 7 \quad\quad 5 \quad 6 \quad 1$$

$$(11\ 0101\ 0111\ 1111\ .\ 1011\ 1001)_2 = (357F.B9)_{16}$$
$$\quad 3 \quad 5 \quad 7 \quad F \quad\quad B \quad 9$$

Binary to Octal and back

- **Binary to Octal:**
  - Group the binary digits into three bit groups starting at the radix point and going both ways, padding with zeros as needed (at the ends)
  - Convert each group of three bits to an equivalent octal digit
- **Octal to Binary:**
  - It is done by reversing the preceding procedure
  - Restate the octal as three binary digits
  - Start at the radix point and go both ways, padding with zeros as needed.

Examples

- **Convert $(10110001101011.1111000011)_2$ to Octal**

  = 010 110 001 101 011 . 111 100 000 110

  = 2 6 1 5 3 . 7 4 0 6

  = $(26153.7406)_8$

- **Convert $(673.124)_8$ to binary**

  = 110 111 011 . 001 010 100

  = $(110111011.001010100)_2$

- **Convert $(1101010011011)_2$ to Octal**

  = $(32433)_8$

011	010	100	011	011
3	2	4	3	3

Binary to Hexadecimal and back

- **Binary to Hexadecimal:**
  - Group the binary digits into four bit groups starting at the radix point and going both ways, padding with zeros as needed (at the ends)
  - Convert each group of four bits to an equivalent hexadecimal digit
- **Hexadecimal to Binary:**
  - It is done by reversing the preceding procedure
  - Restate the hexadecimal as four binary digits
  - Start at the radix point and go both ways, padding with zeros as needed

Examples

- **Convert $(10110001101011.11110010)_2$ to hexadecimal**

  = 0010 1100 0110 1011 . 1111 0010

  = 2 C 6 B . F 2

  = $(2C6B.F2)_{16}$
- **Convert $(306.D)_{16}$ to binary**

  = 0011 0000 0110. 1101

  = $(001100000110.1101)_2$
- **Convert $(1101010011011)_2$ to hexadecimal**

  = $(351B)_{16}$

0011	0101	0001	1011
3	5	1	B

Base-r Arithmetic

- **Arithmetic operations with numbers in base r follow the same rules as for decimal numbers**
- **When a base other than 10 is used, one must remember to use only the r - allowable digits**
- **The following are some examples:**

```
augend:    110011     minuend:    110101     multiplicand:    1011
addend:   +100011     subtrahend: -100111     multiplier:    X 101
          1010110                  001110                     1011
                                                             0000
                                                            1011
                                                            110111
```

Arithmetic Rules

- **The sum of two digits are calculated as expected but the digits of the sum can only be from the r-allowable coefficients**
- **Any carry in a sum is passed to the next significant digits to be summed**
- **In subtraction the rules are the same but a borrow adds r (where r is the base) to the minuend digit**

Binary Addition

Given two binary digits (X,Y), a carry in (Z) we get the following sum (S) and carry (C):

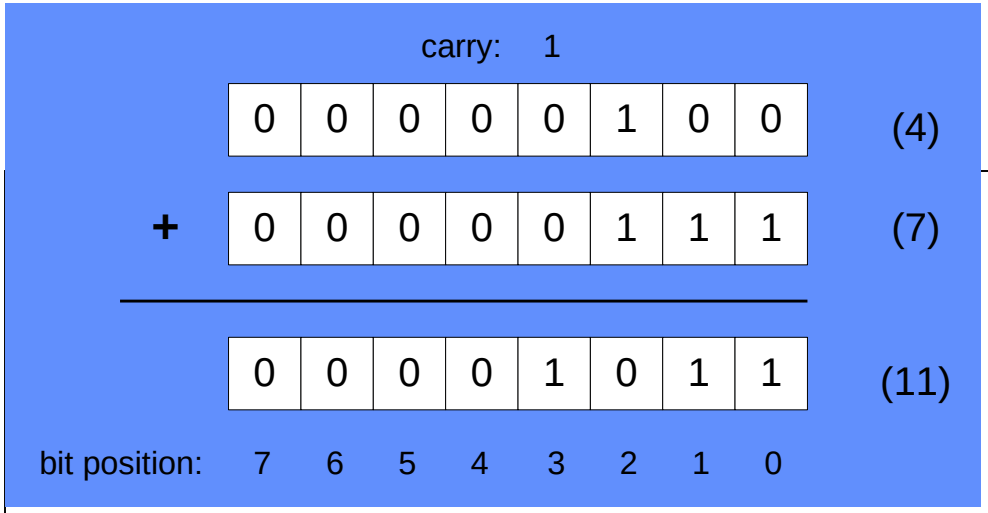
Carry in (Z) of 0:

Z	0	0	0	0
X	0	0	1	1
+ Y	+ 0	+ 1	+ 0	+ 1
C S	0 0	0 1	0 1	1 0

Carry in (Z) of 1:

Z	1	1	1	1
X	0	0	1	1
+ Y	+ 0	+ 1	+ 0	+ 1
C S	0 1	1 0	1 0	1 1

Binary Addition Examples




```
   1   1   1   1   1   1  <-------- carries
       1   1   1   1   0   1      (61)10
+          1   0   1   1   1      (23)10
-------------------------------
   1   0   1   0   1   0   0      (84)10
```

Binary Addition

- **Given two binary digits (X,Y), a carry in (Z) we get the following sum (S) and carry (C):**
  - **Carry in (Z) of 0:**

Z	0	0	0	0
X	0	0	1	1
+ Y	+ 0	+ 1	+ 0	+ 1
C S	0 0	0 1	0 1	1 0

Binary Addition (Contd….)

- **Given two binary digits (X,Y), a carry in (Z) we get the following sum (S) and carry (C):**
  - **Carry in (Z) of 1:**

Z	1	1	1	1
X	0	0	1	1
+ Y	+ 0	+ 1	+ 0	+ 1
C S	0 1	1 0	1 0	1 1

Binary Subtraction

- Given two binary digits (X,Y), a borrow in (Z) we get the following difference (S) and borrow (B):
- Borrow in (Z) of 0:

Z	0	0	0	0
X	0	0	1	1
- Y	-0	-1	-0	-1
BS	0 0	1 1	0 1	0 0

- Borrow in (Z) of 1:

Z	1	1	1	1
X	0	0	1	1
- Y	-0	-1	-0	-1
BS	10	0 1	1 1	1 0

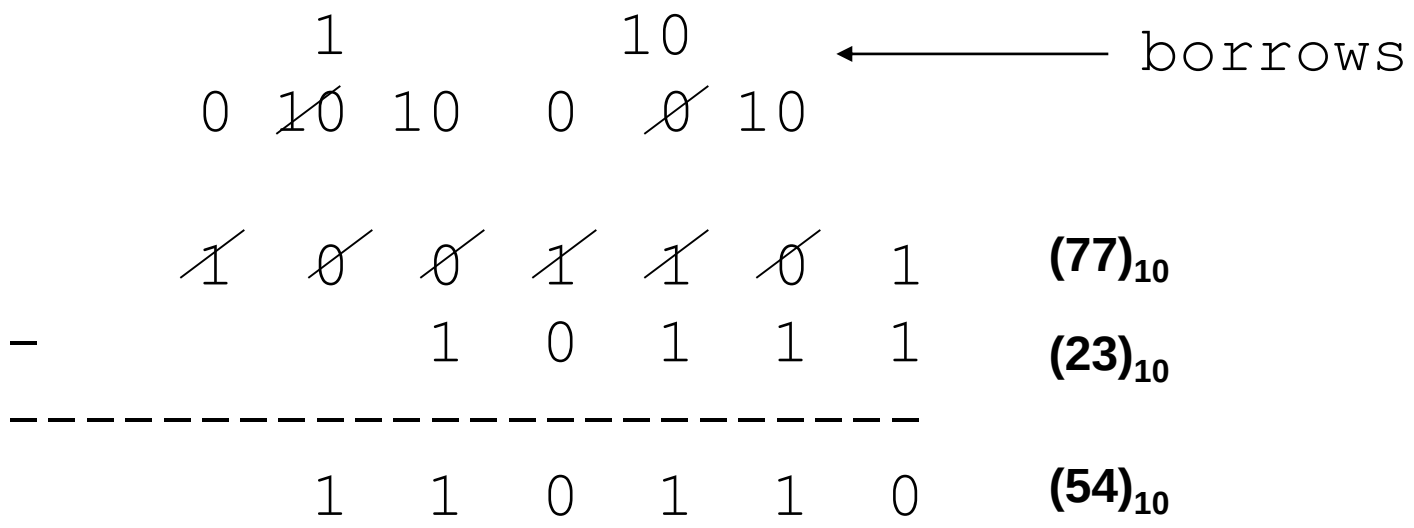
Binary Subtraction

- **Subtraction Table**

0 - 0 = 0
0 - 1 = 1 and borrow 1
1 - 0 = 1
1 - 1 = 0

- **The borrow process works:**
- **Decimal subtraction:**
  - **$1 \times 10^n = 10 \times 10^{n-1}$**
- **Binary subtraction:**
  - **$1 \times 2^n = 2 \times 2^{n-1}$**

Binary Subtraction Example



Binary Multiplication and Division

- **Multiplication table**

  0 x 0 = 0
  0 x 1 = 0
  1 x 0 = 0
  1 x 1 = 1

```
                    1   0   1   1   1
X                       1   0   1   0
-------------------------------------
                    0   0   0   0   0
                1   0   1   1   1
            0   0   0   0   0
        1   0   1   1   1
-------------------------------------
    1   1   1   0   0   1   1   0
```

- **Binary division is similar to decimal division**

Complements

- Complements **are used to simplify subtraction operations**
- **We do subtraction by adding**

  A – B = A + (-B)
- **There are two types:**
  - **The** radix complement**, called the** r's complement
  - **The** diminished radix complement**, called the** (r-1)'s complement

Diminished Radix Complement (DRC)

- **Given a number *N* in base *r* having *n* digits, the (r-1)'s complement of N is defined as:**

  $(r^n - 1) - N$

- **Decimal numbers are in base-10**

  (r-1) = (10-1) = 9.

- **The 9's complement would be defined as:**

  $(10^n - 1) - N$

- **So, to determine the 9's complement of 52:**

  $(10^2 - 1) - 52 = 47$

- **Another example is to determine the 9's complement of 3124:**

  $(10^4 - 1) - 3124 = 6875$

Finding Diminished Radix Complement (DRC)

- **The DRC or (r-1)'s complement of decimal number is obtained by subtracting each digit from 9**
- **The (r-1)'s complement of octal or hexadecimal number is obtained by subtracting each digit from 7 or F, respectively**
- **The DRC (1's complement) of a binary number is obtained by subtracting each digit from 1**
- **It can also be formed by changing 1s to 0s and 0s to 1s**

DRC for Binary Numbers

- For binary numbers r = 2 and (r-1) = 1
- So, the 1's complement would be defined as:

  $(2^n - 1) - N$
- To determine the 1's complement of 1000101:

  $(2^7 - 1) - 1000101 = 0111010$
- To determine the 1's complement of 11110111101:

  $(2^{11} - 1) - 11110111101 = 00001000010$

- Note that 1's complement can be done by switching all 0s to 1s and 1s to 0s

Radix Complement

- **The r's complement of an n-digit number N in base - r is defined as:**

  $r^n - N$ - for $N \neq 0$

  0 - for $N = 0$

- **We may obtain r's complement by adding 1 to (r-1)'s complement**
- **Since $r^n - N = [(r^n - 1) - N] + 1$**
- **10's complement of 3229 is:**

  $10^4 - 3229 = 6771$

- **2's complement of 101101 is:**

  $2^6 - 101101 = 010011$

- **Note: to determine 2's complement, leave the least significant 0s and the first 1 unchanged and then switch the remaining 1s to 0 and 0s to 1**

2's Complement

- **Another method to find 2's complement is**
  - **Complement (reverse) each bit**
  - **Add 1**
- **Example:**

Starting value	00000001
Step 1: reverse the bits	11111110
Step 2: add 1 to the value from Step 1	11111110 +00000001
Sum: two's complement representation	11111111

Note that 00000001 + 11111111 = 100000000

Notes on Complements

- **A couple of notes on complements to keep in mind:**
  - **If you are trying to determine the complement of a value that contains a radix point:**
    - » **Remove the radix point**
    - » **Determine the complement**
    - » **Replace the radix point in the same relative position**
  - **The complement of a complement will restore the original number**

Subtraction with Complements

- **In digital computers the use of borrows to complete subtraction is inefficient**
- **Complements are used to overcome this inefficiency**
- **The subtraction of two *n* - digit unsigned numbers *M* – *N* in base *r* can be done as follows:**
  - **Add the minuend, M, to the r's complement of the subtrahend, N:**
    - » $M + (r^n - N) = M - N + r^n$
  - **If $M \geq N$, the sum will produce an end carry, r^n, which can be discarded; what is left is the result of M – N**
  - **If $M < N$, the sum does not produce an end carry and is equal to $r^n - (N - M)$, 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**

10’s Complement Subtraction

- **Using 10’s complement, subtract 62513 – 2140**
- **Taking 10’s complement of 2140: 10000 – 2140 = 97860**

M =	62513
10’s complement of N =	97860
Sum	160373
Discard end carry	-100000
Answer	60373

- **Note that the extra 9 in the 10’s complement of N is to fill the space holder 0**

10’ Complement Subtraction

- **Using 10’s complement, subtract 2140 – 62513**
- **Taking 10’s complement of 62513: 100000 – 62513 = 37487**

M =	02140
10’s complement of N =	37487
Sum	39627
There is no end carry.	
10’s complement of 39627	60373
(Add - sign) Answer	-60373

2's Complement Subtraction

- **Using 2's complement, subtract 1001001 - 1000110**

M =	1001001
2's complement of N =	0111010
Sum	10000011
Discard end carry 2^7	-10000000
Answer	0000011

2’s Complement Subtraction

- **Using 2’s complement, subtract 1000110 - 1001001**

M =	1000110
2’s complement of N =	0110111
Sum	1111101
There is no end carry.	
2’s complement of 1111101	0000011
(Add – sign) Answer	-0000011

Using 1's Complement

- **You can also use the 1's complement for performing subtraction**
- **You can add the minuend M to the (r-1)'s complement of subtrahend N, then inspect the result**
  - **If an end carry occurs add 1**
  - **If there is no end carry take (r-1)'s complement of the result obtained and place a negative sign**
  - **Note: Remember that 1's complement is 1 less than 2's complement**
  - **This means we must compensate by adding 1 when an end carry occurs**
  - **Removing an end-carry and adding one is called an** end-around carry

1’s Complement Subtraction

- **Using 1’s complement, subtract 1001001 - 1000110**

M =	1001001
1’s complement of N =	0111001
Sum	10000010
Discard end carry 2^7	-10000000
	0000010
Add 1 to compensate	+0000001
Answer	0000011

1’s Complement Subtraction

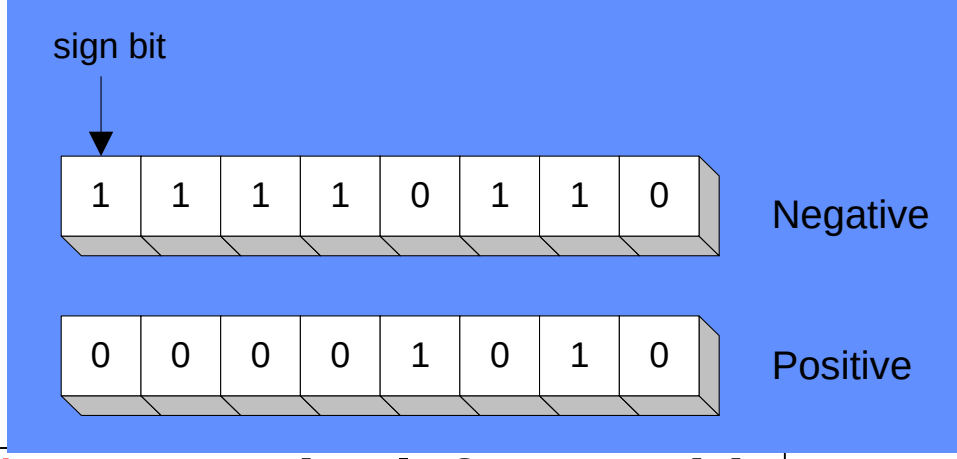
- **Using 1’s complement, subtract 1000110 - 1001001**

M =	1000110
1’s complement of N =	0110110
Sum	1111100
There is no end carry.	
1’s complement of 1111100	0000011
(Add – sign) Answer	-0000011

Signed Binary Numbers

- **In ordinary arithmetic a negative number is indicated by minus sign and positive number by plus sign**
- **Not possible in computers, because of hardware limitation computers must represent everything with binary digits**
- **There are two methods to do this:**
  - **The** signed magnitude convention **uses the left-most bit to represent the sign (0 for positive and 1 for negative)**
  - **The** signed complement system **negates a number by taking its complement**
    - » **It could be either,** 1's complement **representation**
    - » **or** 2's complement **representation.**

Signed Magnitude Convention



- **The signed magnitude convention uses the left-most bit to represent the sign (0 for positive and 1 for negative)**
  - **The user determines whether the number is signed or unsigned**
  - **If the binary number is signed then the leftmost bit represents the sign and the rest of the bits represents the number**
  - **If the binary number is unsigned then the leftmost bit is the most significant bit of the number**
- **Example:**
  - » **01001 can be considered as 9 (unsigned binary) or a +9 because the left most bit is zero**
  - » **On the other hand, the string of bits 11001 represents binary equivalent of 25 when considered as an unsigned number or as –9 when considered as signed number**

Signed Complement System

- **The signed Complement System negative number is indicated by its complement (Complement of positive number)**
  - **Positive numbers always start with 0 (plus), its complement (representing negative number) will always start with 1**
  - **Signed complement system can use either 1's complement or 2's complement.**
- **Example:**
  - » **+9 is represented only as 00001001 but –9 can be represented as:**
    - **11110110** **Signed 1's complement representation**
    - **11110111** **Signed 2's complement representation**

Number Representations

- **The following is the representation for +11:**
  - **00001011**
- **The following are different methods for representing -11:**
  - **Signed magnitude: 10001011**
  - **Signed-1's-complement:11110100**
  - **Signed-2's-complement:11110101**

Number Representations

- **The string of bits : 1011**
  may represent a decimal value in binary form as a:
  - **Unsigned Number: 11**
  - **Signed magnitude: - 3**
  - **Signed-1's-complement: - 4**
  - **Signed-2's-complement: - 5**

Signed Binary Numbers

Decimal	Signed-2's complement	Signed-1's complement	Signed Magnitude
+7	0111	0111	0111
+6	0110	0110	0110
+5	0101	0101	0101
+4	0100	0100	0100
+3	0011	0011	0011
+2	0010	0010	0010
+1	0001	0001	0001
+0	0000	0000	0000
-0	------	1111	1000
-1	1111	1110	1001
-2	1110	1101	1010
-3	1101	1100	1011
-4	1100	1011	1100
-5	1011	1010	1101
-6	1010	1001	1110
-7	1001	1000	1111
-8	1000	------	------

Arithmetic Addition (Signed-Magnitude System)

- **The addition of two signed binary numbers in the signed-magnitude system follows the rules of ordinary arithmetic**
- **If the signs are the same we add the two magnitudes and give the sum the common sign**
- **If the signs are different we subtract the smaller magnitude from the larger and give the result the sign of the larger magnitude**

Arithmetic Addition (Signed 2's Complement system)

- **This system doesn't require the comparison of the signs and the magnitudes (as in signed-magnitude system), but only addition.**
- **The addition of two signed binary numbers with negative numbers represented in signed-2's complement form is obtained from addition of the two numbers, including their sign bits.**
- **A carry out of the sign-bit position is discarded.**
- **If the sum is negative, it will be in 2's complement form.**

Arithmetic Subtraction

- **Subtraction can be performed by simply converting the equation into an addition formula**
  - **Take the 2's complement of the subtrahend (including the sign bit) and add it to the minuend (including the sign bit)**
  - **A carry out of the sign bit position is discarded**
  - **Note: Subtraction operation can be changed to an addition operation if the sign of the subtrahend is changed**
  - **This is easily done by taking it's 2's complement**

Example Arithmetic (Signed 2's Complement)

+ 9		00001001		- 9		11110111
+11		00001011		+11		00001011
+20		00010100		+ 2		00000010
+ 9		00001001		- 9		11110111
-11		11110101		-11		11110101
- 2		11111110		-20		11101100

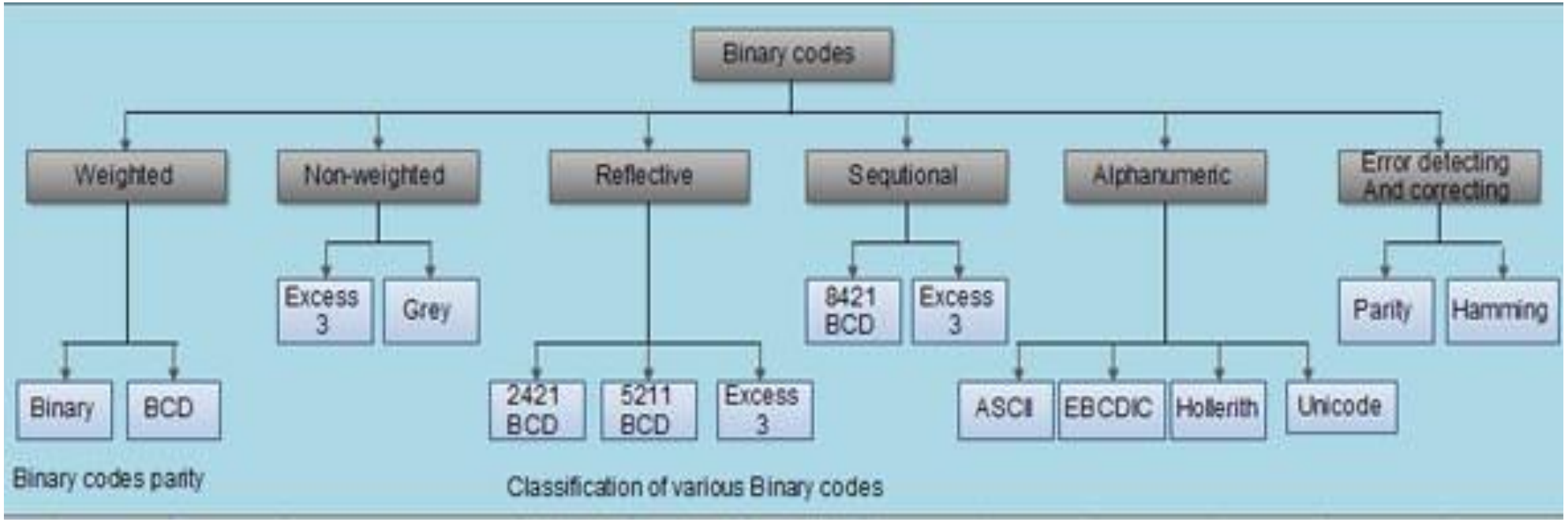
Example

- **Consider the subtraction (–6) – (–13) = +7**
- **In binary with eight bits the same is written as (11111010 – 11110011)**
- **This subtraction is changed to addition by taking 2's complement of the subtrahend (– 13) to give (+ 13)**
- **In binary this is 11111010 + 00001101 = 100000111**
- **Removing the end carry, we obtain the correct answer: 00000111(+7)**

Binary Codes

- **All symbols in a computer must be represented by a binary code (binary representation)**
- **An n-bit binary code is a group of n bits that can represent up to 2^n distinct combinations of 1's and 0's**
- **Each distinct combination represents a single symbol in the computer**

Binary Codes



- **Weighted codes:** In weighted codes, each digit is assigned a specific weight according to its position.
- **Non-weighted codes:** In non-weighted codes are not appositionally weighted.
- **Reflective codes**: A code is reflective when the code is self complementing. In other words, when the code for 9 is the complement the code for 0, 8 for 1, 7 for 2, 6 for 3 and 5 for 4.
- **Sequential codes**: In sequential codes, each succeeding 'code is one binary number greater than its preceding code.
- **Alphanumeric codes**: Codes used to represent numbers, alphabetic characters, symbols
- **Error defecting and correcting codes**: Codes which allow error defection and correction are called error detecting and' correcting codes.

BCD Code (8 4 2 1)

- **The most common representation for binary digits is the Binary Coded Decimal (BCD) form which is a binary assignment of the decimal numbers**
  - **This code is the simplest, most intuitive binary code for decimal digits and uses the same weights as a binary number, but only encodes the first ten values from 0 to 9 (6 out of 16 possible combinations remains unassigned)**
  - **A number with k distinct decimal digits will require 4k bits in BCD**
  - **Each digit of a decimal value is converted to its respective binary representation**
  - **BCD number needs more bits than its equivalent binary value?**

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

Multi-Digit BCD

Decimal Symbol	BCD Representation
10	0001 0000
11	0001 0001
212	0010 0001 0010
213	0010 0001 0011
5673	0101 0110 0111 0011
5684	0101 0110 1000 0100

BCD Addition

- **BCD only represents each of the decimal digitals 0 through 9 as a single 4-bit binary value**
- **When adding two BCD values, if the sum is equal to or less than 1001 (9), the corresponding BCD value is correct**
- **However, when the binary sum is greater or equal to 1010 (10), the result is an invalid BCD value**
  - **To overcome the invalid BCD value add 0110 (6) to the result to obtain the BCD representation and also produces a carry as required**
  - **The use of 0110 (6) works because the difference between a carry in the most significant bit position of the binary sum and a decimal carry differ by 16-10 = 6**

BCD Addition Examples

4		0100		3		0011		9		1001
+5		0101		+7		0111		+9		1001
9		1001		10		1010		18		10010
						+0110				+0110
						0001 0000				0001 1000

Multi-Digit BCD Addition

BCD Carry	1	1		
	0010	1001	0101	295
	0110	0011	0101	+635
Binary Sum	1001	1101	1010	
Add 6		0110	0110	
BCD Sum	1001	0011	0000	930

BCD Arithmetic

- **BCD arithmetic involving negative numbers uses the 10's complement for representing the negative numbers including the sign digit**
  - **0 (0000) represents a positive sign and 9 (1001) represents a negative sign**
- **As an example, imagine we want to add**

  (+257) + (-160) = +97

	1
0 257	0000 0010 0101 0111
9 840	1001 1000 0100 0000
	1010 1010 1001 0111
	0110 0110
0 097	0000 0000 1001 0111

- **Note: To obtain 10's complement of a BCD number, we first take the 9's complement (by subtraction of each digit from 9) and then add one to least significant digit**

Other Decimal Codes

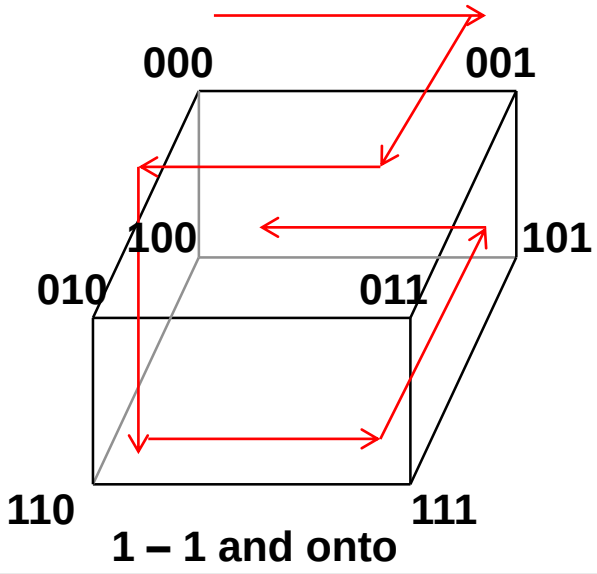
- **There are various other decimal codes that can be used:**
  - **BCD (8 4 2 1)**
  - **2 4 2 1**
  - **Excess-3 code. (adds binary 0011 to the BCD code)**
  - **8 4 -2 -1**
- **Each bit has a "weight" associated with it and you can compute the decimal value by adding the weights where a 1 exists in the code-word**

Four Different (Decimal) Binary Codes

Decimal digit	BCD 8421	2421	Excess-3	8 4 -2 -1
0	0000	0000	0011	0000
1	0001	0001	0100	0111
2	0010	0010	0101	0110
3	0011	0011	0110	0101
4	0100	0100	0111	0100
5	0101	1011	1000	1011
6	0110	1100	1001	1010
7	0111	1101	1010	1001
8	1000	1110	1011	1000
9	1001	1111	1100	1111

Gray Code

- It is sometimes convenient to use the Gray code to represent the digital data when it is converted from analog data
- The advantage of Gray code over straight binary number sequence is that only one bit in the code group changes when going from one number to the next
  - Error Detection
  - Representation of Analog Data
  - Low Power Design



Gray Code

Decimal digit	Gray code
0	0000
1	0001
2	0011
3	0010
4	0110
5	0111
6	0101
7	0100

Decimal digit	Gray code
8	1100
9	1101
10	1111
11	1110
12	1010
13	1011
14	1001
15	1000

Gray Code Vs Binary Code

- **Compare the number of bits changing when going from one number to the next:**
- **In Gray code it is always 1 bit.**

Binary Code	Bit Changes	Gray Code	Bit Changes
000		000	
	1		1
001		001	
	2		1
010		011	
	1		1
011		010	
	3		1
100		110	
	1		1
101		111	
	2		1
110		101	
	1		1
111		100	
	3		1
000		000	

ASCII Character Code & Properties

- **The American Standard Code for Information Interchange (ASCII) (Refer to Table 1.7)**
  - **Popular code to represent information sent as character-based data**
  - **Uses seven bits to code 128 characters, representing the alphabet, decimal numbers (94 Graphic printing characters), and various other symbols (34 Non-printing characters)**
  - **For example, the character 5 is represented in binary as 0110101**
- **Properties**
  - **Digits 0 to 9 span Hexadecimal values 30_{16} to 39_{16}**
  - **Upper case A-Z span 41_{16} to $5A_{16}$**
  - **Lower case a-z span 61_{16} to $7A_{16}$**
    - » **Lower to upper case translation (and vice versa) occurs by flipping bit 6**

ASCII Table

The following ASCII chart allows you to specify the characters in decimal representat by concatenating the column headings to the row headings

	$B_7B_6B_5$							
$B_4B_3B_2B_1$	000	001	010	011	100	101	110	111
0000	NULL	DLE	SP	0	@	P	`	p
0001	SOH	DC1	!	1	A	Q	a	q
0010	STX	DC2	"	2	B	R	b	r
0011	ETX	DC3	#	3	C	S	c	s
0100	EOT	DC4	$	4	D	T	d	t
0101	ENQ	NAK	%	5	E	U	e	u
0110	ACK	SYN	&	6	F	V	f	v
0111	BEL	ETB	'	7	G	W	g	w
1000	BS	CAN	(	8	H	X	h	x
1001	HT	EM	)	9	I	Y	i	y
1010	LF	SUB	*	:	J	Z	j	z
1011	VT	ESC	+	;	K	[	k	{
1100	FF	FS	,	<	L	\	l	|
1101	CR	GS	-	=	M	]	m	}
1110	SO	RS	.	>	N	^	n	~
1111	SI	US	/	?	O	_	o	DEL

ASCII Table (Contd ……)

Control Characters:

NULL	NULL	DLE	Data link escape
SOH	Start of heading	DC1	Device control 1
STX	Start of text	DC2	Device control 2
ETX	End of text	DC3	Device control 3
EOT	End of transmission	DC4	Device control 4
ENQ	Enquiry	NAK	Negative acknowledge
ACK	Acknowledge	SYN	Synchronous idle
BEL	Bell	ETB	End of transmission block
BS	Backspace	CAN	Cancel
HT	Horizontal tab	EM	End of medium
LF	Line feed	SUB	Substitute
VT	Vertical tab	ESC	Escape
FF	Form feed	FS	File separator
CR	Carriage return	GS	Group separator
SO	Shift out	RS	Record separator
SI	Shift in	US	Unit separator
SP	Space	DEL	Delete

Error-Detecting Code

- Error-Detecting **uses an eighth bit (added to 7-bit ASCII character) to indicate** parity
  - **A** parity bit **is an extra bit that is set to 0 or 1 as needed to make the total number of 1's either even or odd**
  - **In an odd-parity code, the parity bit is specified so that the total number of ones is odd**
  - **In an even-parity code, the parity bit is specified so that the total number of ones is even**
  - **It detects one, three or any odd combination of errors but even combination of errors is undetected**


```
1 1 0 0 0 0 1 1          0 1 0 0 0 0 1 1
↑                        ↑
Added even parity bit    Added odd parity bit
```

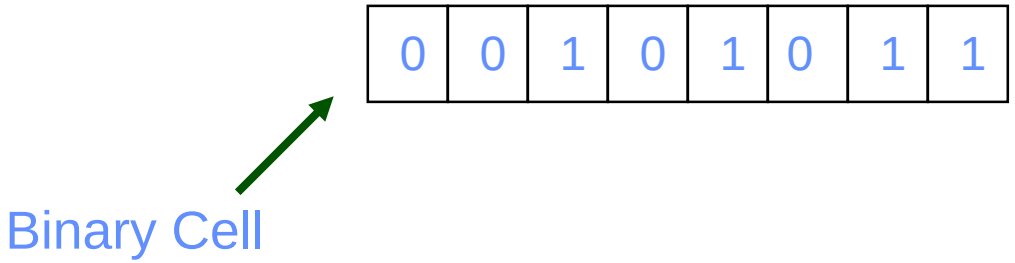
Parity Code Example

- Concatenate **a parity bit to the ASCII code for the characters 0, X, and = to produce both odd-parity and even-parity codes.**

Character	ASCII	Odd-Parity ASCII	Even-Parity ASCII
0	0110000	10110000	00110000
X	1011000	01011000	11011000
=	0111100	10111100	00111100

Binary Storage

- Binary storage **represents the storage mechanisms for binary data stored in a computer**
  - **A** binary cell **is a device that possesses two stable states and it can store a single state value (0 or 1)**
  - **It stores single bit of data. examples: flip-flop circuits, ferrite cores, capacitor**
  - **A** register **is a group of binary cells**
  - **n cells allows the register to store n bits and thus 2^n possible states**
    - » **The type of information (BCD, ASCII, etc.) stored in a register has to be agreed upon by the users of the register**



Register Transfer

- **A register transfer operation involves the transfer of binary information from one set of binary registers into other set of binary registers**
- **The capture and storage of information requires:**
  - **An input register to store the key inputs from the keyboard**
  - **A processor register to store the data when processed by the CPU**
  - **A memory register in the memory unit to store the values**

Register Transfer

- **Data input at keyboard**
- **Shifted into place**
- **Stored in memory**
  - NOTE: Data input in ASCII

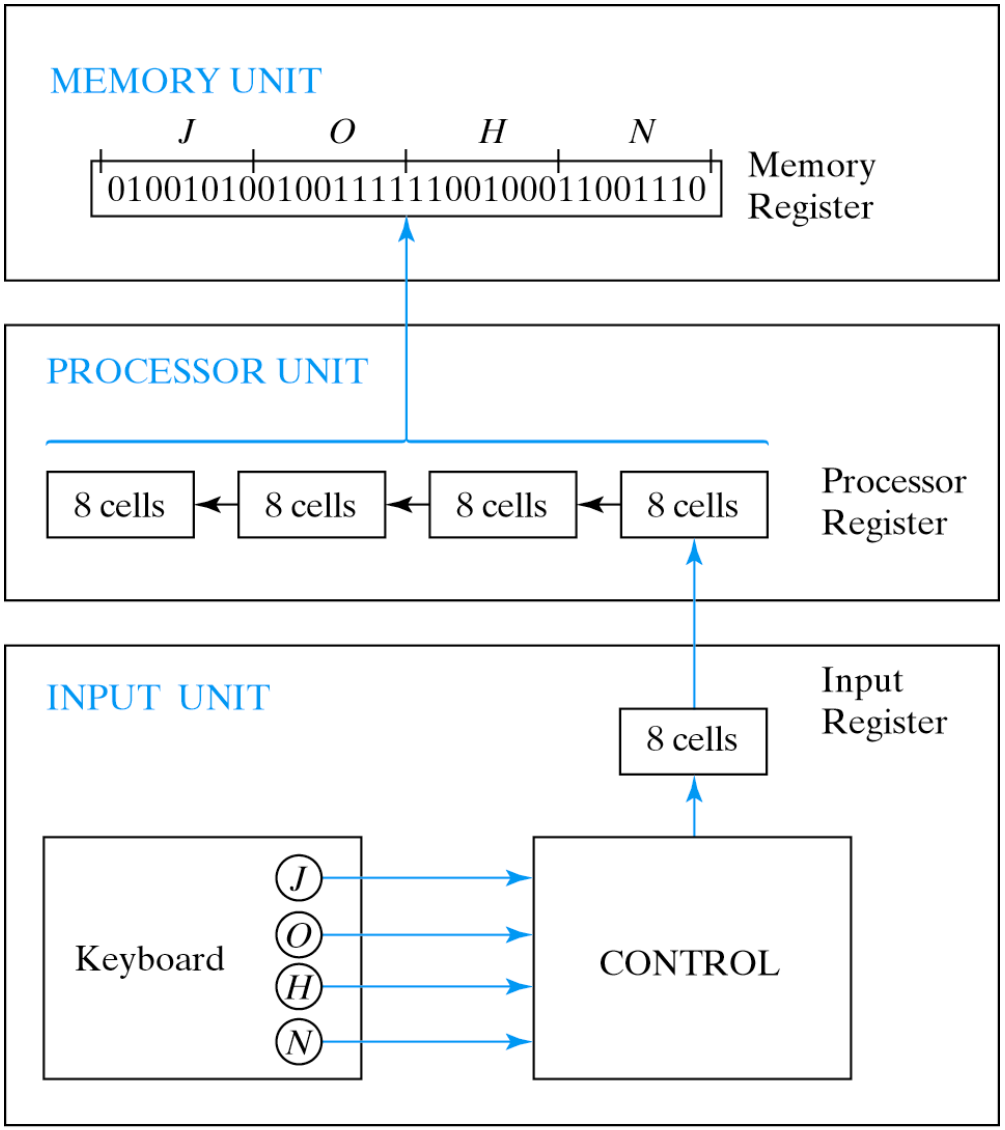


Fig. 1-1 Transfer of information with registers

Binary Information Processing

- **The actual processing of binary information in a computer is completed by digital logic circuits which have been implemented to serve a specific purpose (i.e. addition)**
  - **The registers are accessed (read and write) when they are needed to complete an operation**
  - **For example we need two register sets to store two values to be added and a register set to store the result of the sum**
    - » **Furthermore, we need three registers in both the memory unit and in the processor**

Example Binary Information Processing

- We need processing
- We need storage
- We need communication

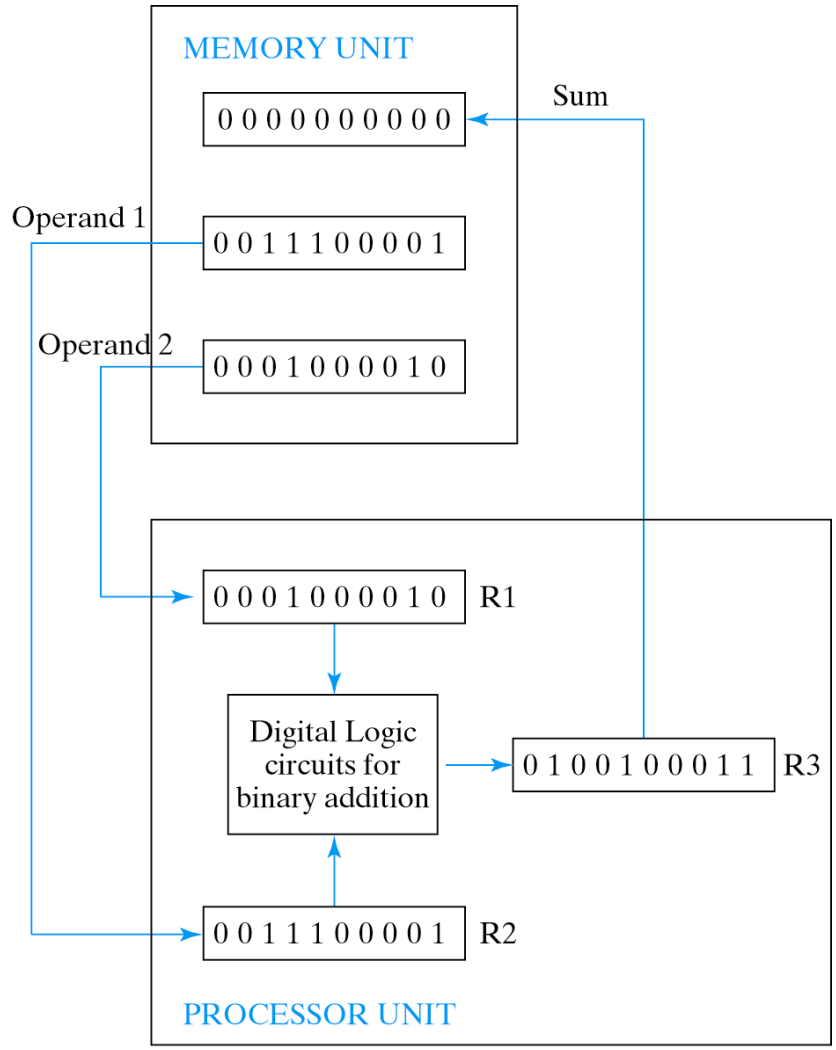


Fig. 1-2 Example of binary information processing

Binary Logic

- **Binary logic consists of binary variables and logical operations**
  - **The variables are designated by letters of the alphabet (A, B, C, x, y, z, etc.)**
  - **Each variable can have only two and only two distinct possible values: 1 and 0**
  - **There are three basic logical operations:**
    - **AND**
    - **OR**
    - **NOT**

Logical Operations

- AND **is represented by a dot or the absence of an operator.**
  - **x·y = z or xy = z**
  - **Read as "x and y is equal to z"**
  - **Means that z=1 if and only if x=1 and y=1; otherwise z=0**
- OR **is represented by a plus sign.**
  - **x+y = z**
  - **Read as "x or y is equal to z"**
  - **Means that z=1 if x=1 or y=1 or either x=1 or y=1. if both x=0 and y=0, then z=0**
- NOT **is represented by a prime or an overbar.**
  - **x'=z or $\bar{x}$ = z**
  - **Read as not x is equal to z**
  - **Means that if x=1 then z=0 or if x=0 then z=1**
  - **Also referred to as the complement operation**

Truth Tables

- **Since each binary variable consists of value of 0 or 1, each combination of values for the variables involved in a binary operation has a specific result value**
- **A truth table is a method of visualizing all possible combinations of the input values and the respective output values that occur due to the operation on the specified combination**

AND Truth Table

z = xy

x	y	x·y
0	0	0
0	1	0
1	0	0
1	1	1

AND

A	B	A.B
0	0	0
0	1	0
1	0	0
1	1	1

OR Truth Table

z = x+y

x	y	x+y
0	0	0
0	1	1
1	0	1
1	1	1

OR

A	B	A+B
0	0	0
0	1	1
1	0	1
1	1	1

NOT Truth Table

z = x’

x	x’
0	1
1	0

NOT

A	A'
0	1
1	0

Binary Signal

- **Two separated voltage levels represents a binary variable equal to logic 1 or logic 0**
- **Logic 0 is equal to 0 volt and logic 1 equal to 4 volt (with acceptable range as shown in figure)**

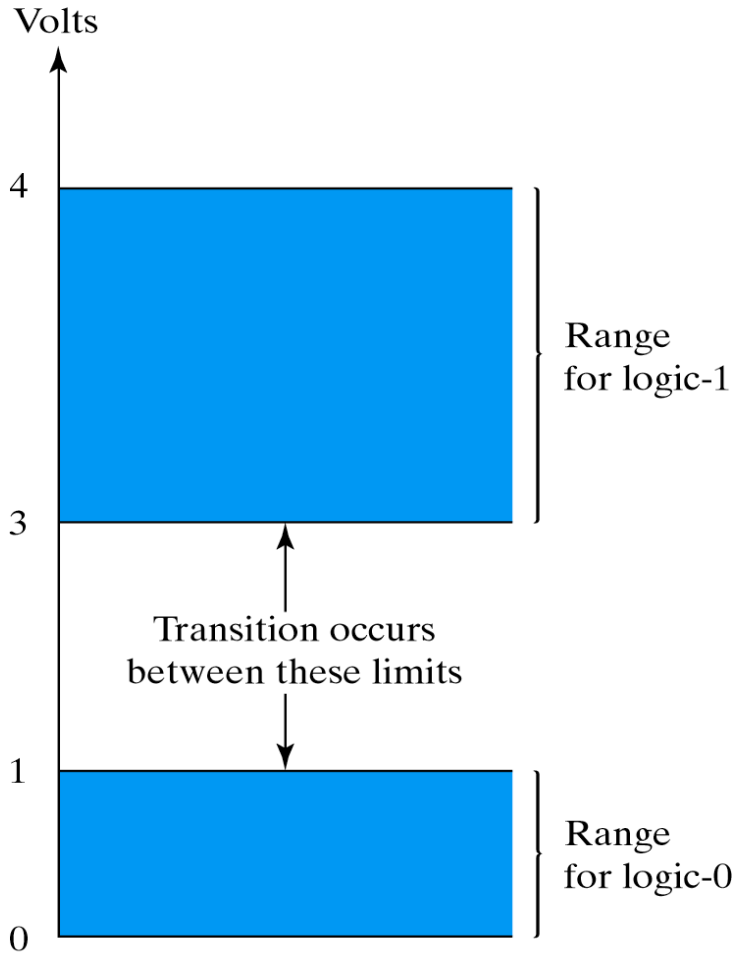
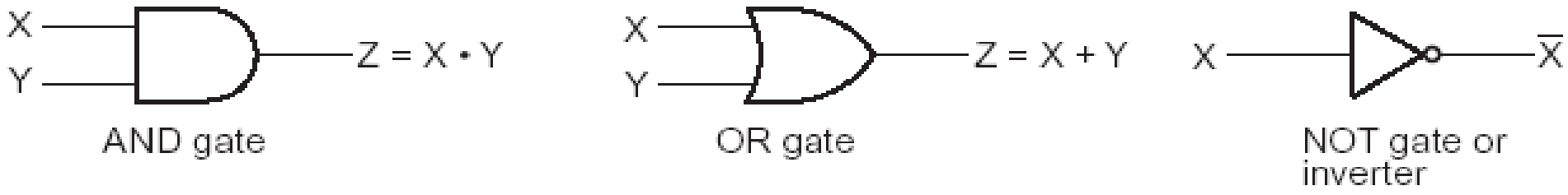


Fig. 1-3 Example of binary signals

Logic Gates

- Logic gates **are electronic circuits that operate on one or more input signals to produce an output signal.**
  - **The state (high-low, on-off) of electricity on a line represents each of the two states for binary representation (1 or 0)**

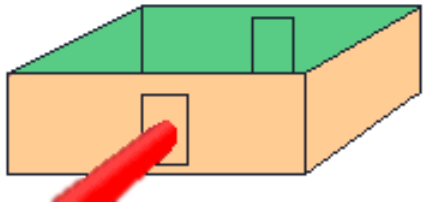
- **Logic Gate Notation**



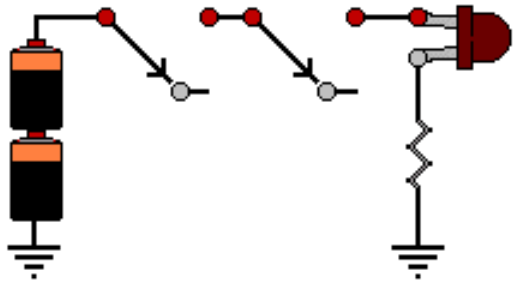
AND Logic Function

A	B	C
0	0	0
0	1	0
1	0	0
1	1	1

- **Using door**
  - **Both doors are opened to pass the light**



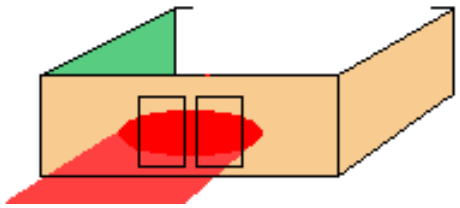
- **Using Switches**
  - **Switches are input and LED is output**
  - **Both switches closed (ON) to give output**



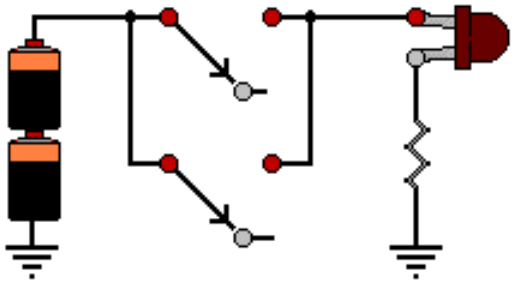
OR Logic Function

A	B	C
0	0	0
0	1	1
1	0	1
1	1	1

- **Using door**
  - **Any one or both doors are opened to pass the light**



- **Using Switches**
  - **Switches are input and LED is output**
  - **Any one switch or both closed to give output**



Timing Diagram

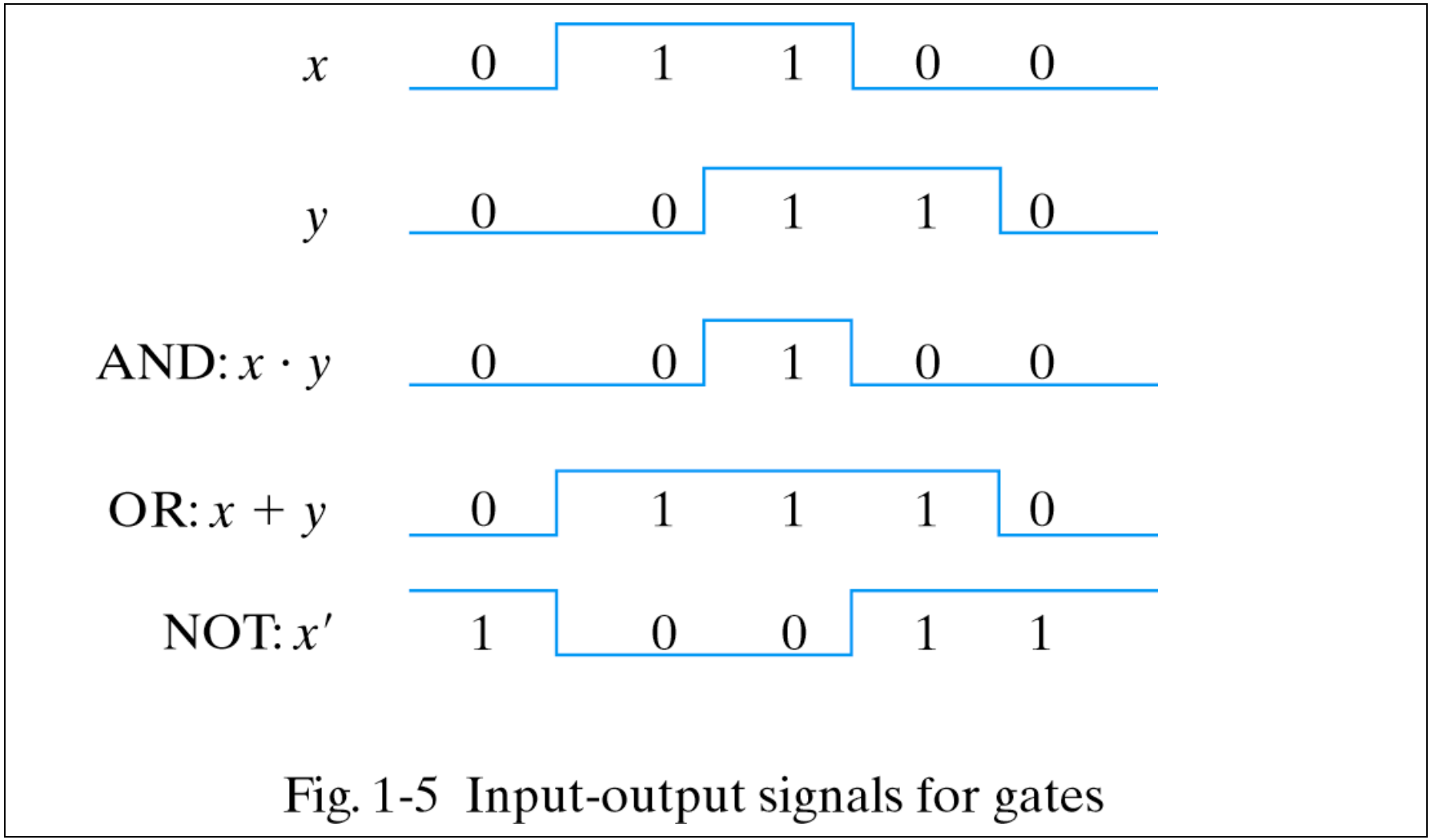
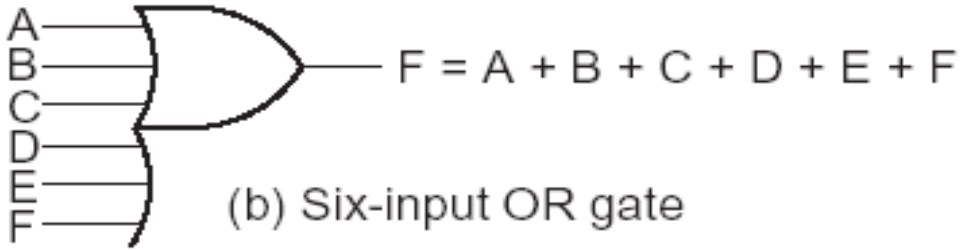


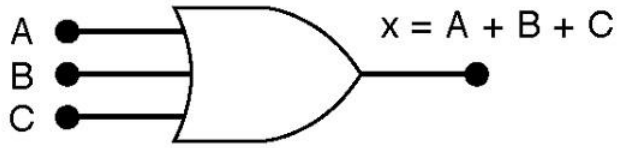
Fig. 1-5 Input-output signals for gates

Multi-Input Circuits

(a) Three-input AND gate



(b) Six-input OR gate



A	B	C	x = A + B + C
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

End of Chapter 1