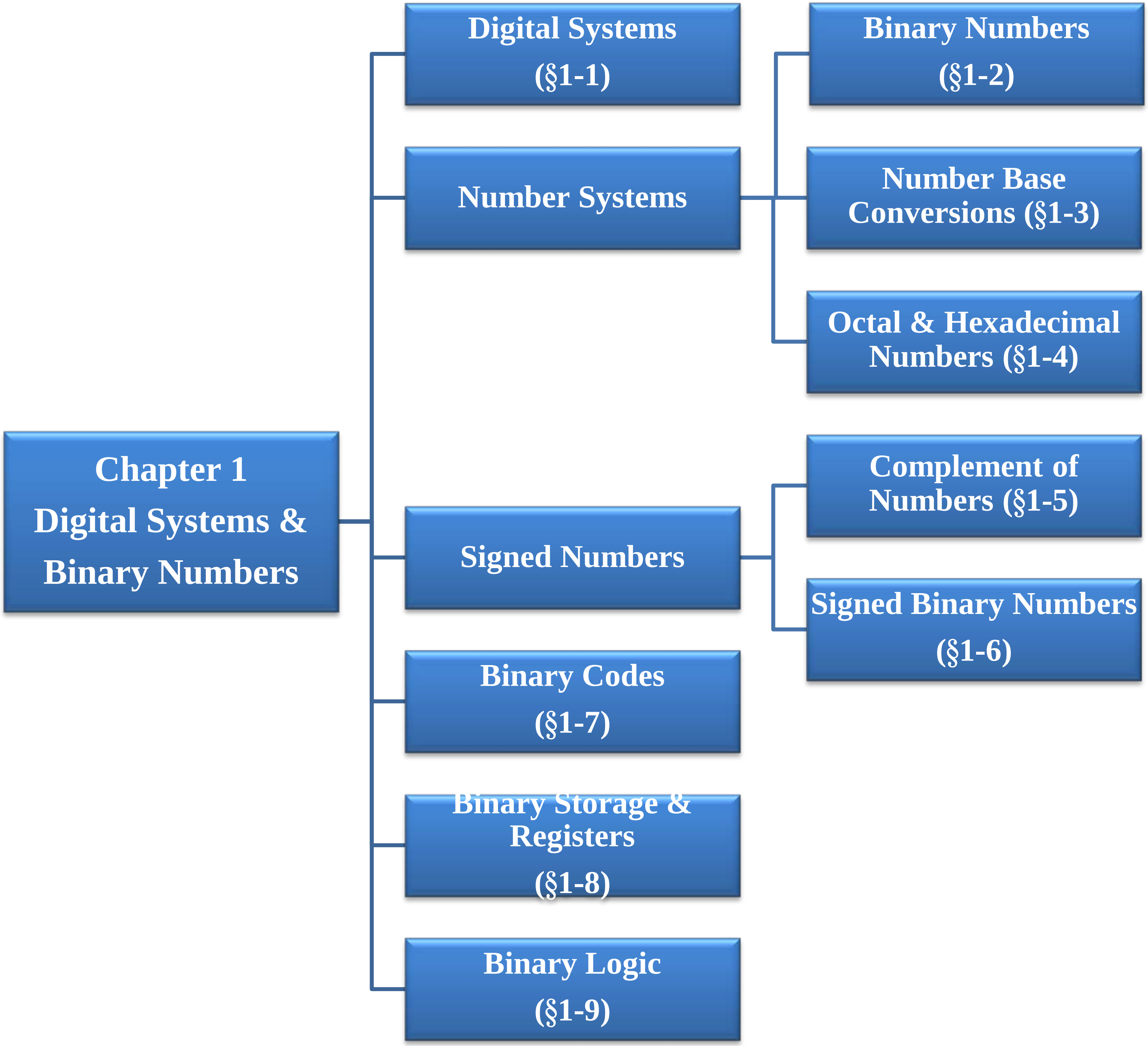
Chapter 1

Digital Systems and Binary Numbers

J.J. Shann

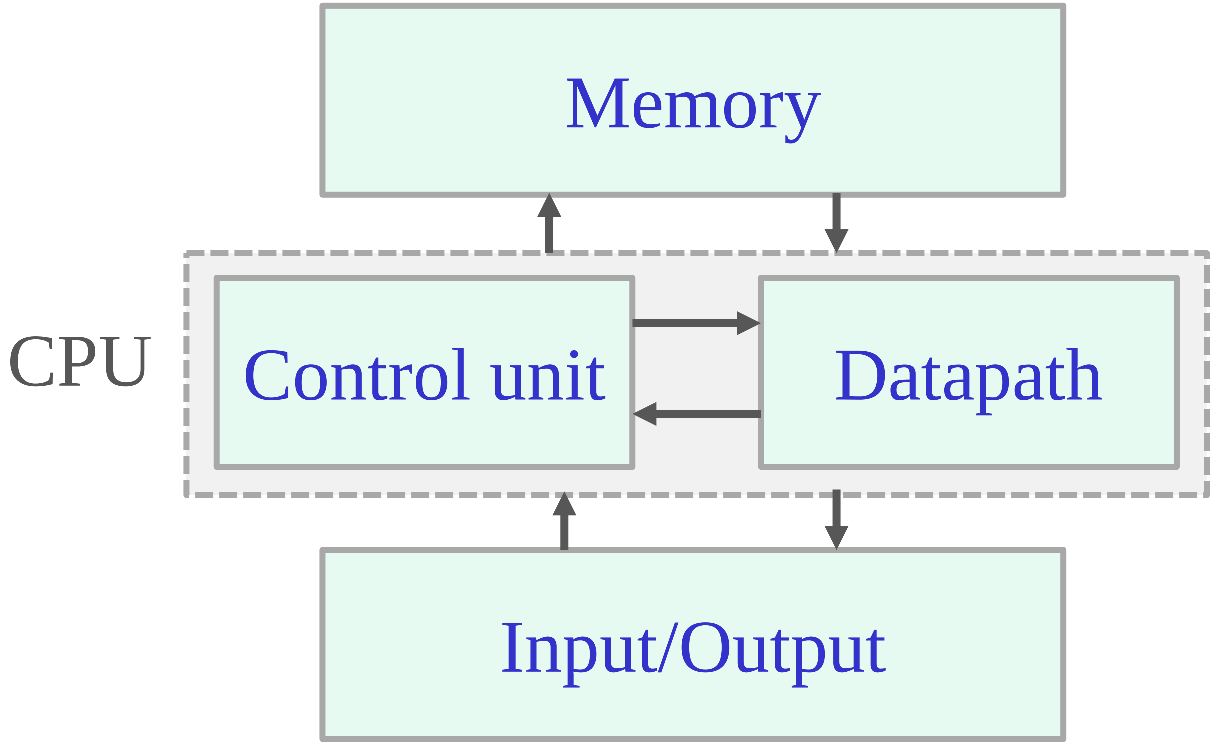
Chapter Overview

1-1 Digital Systems
1-2 Binary Numbers
1-3 Number Base Conversions
1-4 Octal and Hexadecimal Numbers
1-5 Complements of Numbers
1-6 Signed Binary Numbers
1-7 Binary Codes
1-8 Binary Storage and Registers
1-9 Binary Logic


Chapter 1
Digital Systems &
Binary Numbers
Digital Systems
(§1-1)
Number Systems
Binary Numbers
(§1-2)
Number Base
Conversions (§1-3)
Octal & Hexadecimal
Numbers (§1-4)
Signed Numbers
Complement of
Numbers (§1-5)
Signed Binary Numbers
(§1-6)
Binary Codes
(§1-7)
Binary Storage &
Registers
(§1-8)
Binary Logic
(§1-9)

1-1 Digital Systems

- Digital system:
  - manipulates discrete elements of information
  - E.g.: general-purpose digital computer



Discrete Information

- Discrete information:
  - any set that is restricted to a finite # of elements
  - E.g.: 10 decimal digits (0 ~ 9),
    26 letters of the alphabet,
    52 playing cards,
    64 squares of a chessboard
- Binary information: 2 discrete elements
  - are used in most present-day electronic digital systems
    ⇐ The resulting transistor ckt w/ an output that is either HIGH or LOW is simple, easy to design, and extremely reliable.
  - *bit*: a binary digit

* *byte*: 8 bits

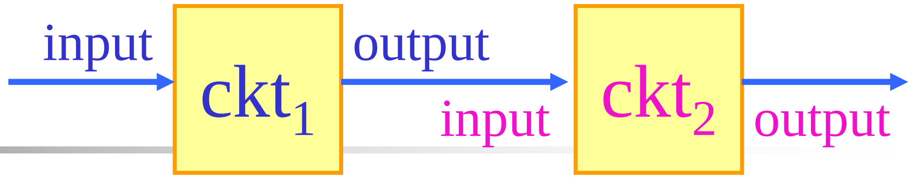
- Abstract representation of binary values:
  - HIGH (H), LOW (L)
  - TRUE (T), FALSE (F)
  - ON, OFF
  - 0, 1

Signal

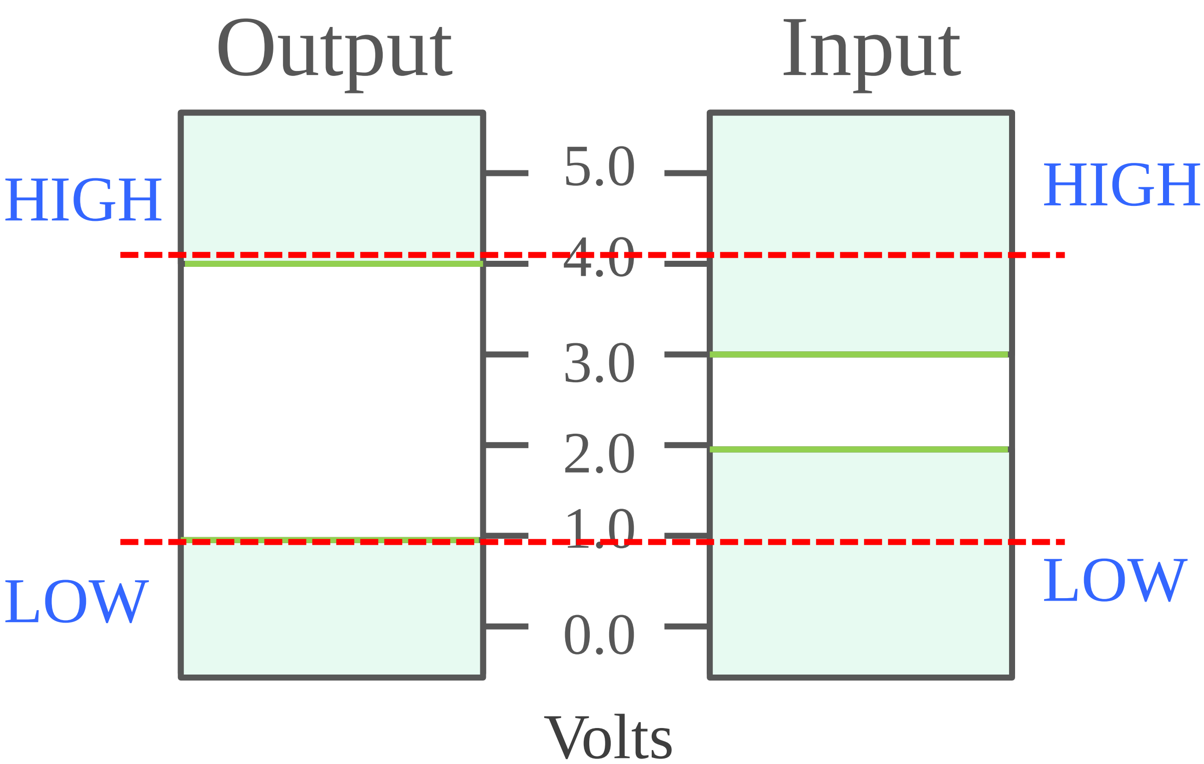
- Signal:
  - physical quantity used to represent discrete elements
  - E.g.:

CPU	Voltage
Disk	Magnetic field direction
Dynamic RAM	Electrical charge

Binary Signal



- Binary signal:
  - represents two discrete elements
  - E.g.: voltage ranges for binary signals



Design Trend

- Important trend in digital design:
  - Use of Hardware Description Language (HDL):
    - HDL:

      resembles a programming language &

      is suitable for describing digital circuits in textural form.
    - Usage:

      simulate a digital system to verify its operation before hardware is built in &

      conjunction with logic synthesis tools to automate the design

1-2 Binary Numbers

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

$$\begin{matrix} r^{n-1} & r^{n-2} & \dots & r^{1} & r^{0} & & r^{-1} & r^{-2} & \dots & r^{-m+1} & r^{-m} \\ A_{n-1} & A_{n-2} & \dots & A_{1} & A_{0} & . & A_{-1} & A_{-2} & \dots & A_{-m+1} & A_{-m} \end{matrix}$$

$0 \le A_i < r$ & . 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)

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

- Examples:

  a decimal number 7392:

  $7 \times 10^3 + 3 \times 10^2 + 9 \times 10^1 + 2 \times 10^0$

  a base-5 number 4021.2:

  $4 \times 5^3 + 0 \times 5^2 + 2 \times 5^1 + 1 \times 5^0 + 2 \times 5^{-1} = (511.4)_{10}$

  a binary number 11010.11:

  $1 \times 2^4 + 1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 0 \times 2^0 + 1 \times 2^{-1} + 1 \times 2^{-2}$

  $= (26.75)_{10}$

  an octal (base-8) number 127.4:

  $1 \times 8^2 + 2 \times 8^1 + 7 \times 8^0 + 4 \times 8^{-1} = (87.5)_{10}$

  a hexadecimal (base-16) number B65F:

  $11 \times 16^3 + 6 \times 16^2 + 5 \times 16^1 + 15 \times 16^0 = (46687)_{10}$

Arithmetic Operations

- Arithmetic ops w/ numbers in base r :
  - follow the same rules as for decimal numbers.
  - Notice: When a base other than base 10 is used
    - use only r allowable digits ($0 \sim r - 1$)
    - perform all computations w/ base-r digits

Binary Addition

- E.g.: 101101 + 100111 = 1010100

Carries	1 0 1 1 1 1
Augend:	1 0 1 1 0 1
Addend:	+ 1 0 0 1 1 1
Sum	1 0 1 0 1 0 0

Binary Subtraction

- E.g.: 101101 – 100111 = 000110

Borrows:	0 0 1 1 0
Minuend:	1 0 1 1 0 1
Subtrahend:	– 1 0 0 1 1 1
Difference:	0 0 0 1 1 0

- E.g.: 10011 – 11110 = – 01011

Borrows:		0 0 1 1 0
Minuend:	10011	1 1 1 1 0
Subtrahend:	–11110	– 1 0 0 1 1
Difference:	–01011	0 1 0 1 1

Binary Multiplication

- E.g.: $1011 \times 101 = 110111$

$$
\begin{array}{lr}
\text{Multiplicand:} & 1011 \\
\text{Multiplier:} & \times \quad 101 \\
\hline
 & 1011 \\
 & 0000 \\
 & 1011 \\
\hline
\text{Product:} & 110111
\end{array}
$$

Base-*r* Arithmetic Operations

- Base-*r* addition:
  - Convert each pair of digits in a column to decimal,

    add the digits in decimal, and then

    convert the result to the corresponding sum and carry in the base-*r* system.

1-3 & 1-4 Number-Base Conversion

- Base $r \rightarrow$ Decimal:
  - expand the number into a power series w/ a base of *r* and add all the terms

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

  - E.g.s: (p.1-10)

$(4021.2)_5 = (?)_{10}$

$4 \times 5^3 + 0 \times 5^2 + 2 \times 5^1 + 1 \times 5^0 + 2 \times 5^{-1} = (511.4)_{10}$

$(11010.11)_2 = (?)_{10}$

$1 \times 2^4 + 1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 0 \times 2^0 + 1 \times 2^{-1} + 1 \times 2^{-2} = (26.75)_{10}$

$(127.4)_8 = (?)_{10}$:

$1 \times 8^2 + 2 \times 8^1 + 7 \times 8^0 + 4 \times 8^{-1} = (87.5)_{10}$

$(B65F)_{16} = (?)_{10}$:

$11 \times 16^3 + 6 \times 16^2 + 5 \times 16^1 + 15 \times 16^0 = (46687)_{10}$

Converting Decimal to Base r

- Decimal $\rightarrow$ Base r : Integer part + Fraction part
  - Integer part:
    - divide the number and all successive quotients by r and accumulate the remainders.
  - Fraction part:
    - multiply the number and all successive fractions by r and accumulate the integers.

$$\begin{array}{cccccccccc} r^{n-1} & r^{n-2} & \dots & r^{1} & r^{0} & r^{-1} & r^{-2} & \dots & r^{-m+1} & r^{-m} \\ A_{n-1} & A_{n-2} & \cdots & A_1 & A_0 \,. & A_{-1} & A_{-2} & \cdots & A_{-m+1} & A_{-m} \end{array}$$

$$\begin{array}{cccccccccc} r^{n-1} & r^{n-2} & \ldots & r^{1} & r^{0} & r^{-1} & r^{-2} & \ldots & r^{-m+1} & r^{-m} \\ A_{n-1} & A_{n-2} & \ldots & A_1 & A_0 & A_{-1} & A_{-2} & \ldots & A_{-m+1} & A_{-m} \end{array}$$

- Example: $(41.6875)_{10} = (?)_2$

Integer part

$41 \div 2 = 20 \ldots 1$

$20 \div 2 = 10 \ldots 0$

$10 \div 2 = 5 \ldots 0$

$5 \div 2 = 2 \ldots 1$

$2 \div 2 = 1 \ldots 0$

$1 \div 2 = 0 \ldots 1$

Left

Fraction part

$0.6875 \times 2 = 1.375$

$0.375 \times 2 = 0.75$

$0.75 \times 2 = 1.5$

$0.5 \times 2 = 1.0$

Right

$$(41.6875)_{10} = (101001.1011)_2$$

- Example: $(153.513)_{10} = (?)_8$

Integer part

$$153 \div 8 = 19 \ldots \mathbf{1}$$
$$19 \div 8 = 2 \ldots \mathbf{3}$$
$$2 \div 8 = 0 \ldots \mathbf{2}$$

Fraction part

$$0.513 \times 8 = \mathbf{4}.104$$
$$0.104 \times 8 = \mathbf{0}.832$$
$$0.832 \times 8 = \mathbf{6}.656$$
$$0.656 \times 8 = \mathbf{5}.248$$
$$0.248 \times 8 = \mathbf{1}.984$$
$$0.984 \times 8 = \mathbf{7}.872$$
$$\vdots \qquad \vdots$$

$$(153.513)_{10} = (231.406517\ldots)_8$$

Conversion b/t Binary and Octal/Hexadecimal

- Binary → Octal/Hexadecimal:
  - Partition the binary number into groups of 3/4 bits each, starting from the binary point and proceeding to the left and to the right.

    The corresponding octal/hexadecimal digits is then assigned to each group.
  - E.g.s:

$$(010\ 110\ 001\ 101\ 011\ .\ 111\ 100\ 000\ 110)_2$$
$$= (2\quad 6\quad 1\quad 5\quad 3\ .\ 7\quad 4\quad 0\quad 6)_8$$

$$(0010\ 1100\ 0110\ 1011\ .\ 1111\ 0000\ 0110)_2$$
$$= (2\quad C\quad 6\quad B\ .\ F\quad 0\quad 6)_{16}$$

Octal/Hexadecimal → Binary:

- Each octal/hexadecimal digit is converted to a 3/4-bit binary equivalent and extra 0's are deleted.
- E.g.: $(673.124)_8 = (?)_2$

  $$\begin{array}{ccccccc} 6 & 7 & 3 & . & 1 & 2 & 4 \end{array}$$

  $$(110\ 111\ 011\ .\ 001\ 010\ 1\cancel{0}\cancel{0})_2$$

- E.g.: $(306.D)_{16} = (?)_2$

  $$\begin{array}{ccccc} 3 & 0 & 6 & . & D \end{array}$$

  $$(\cancel{0}\cancel{0}11\ 0000\ 0110\ .\ 1101)_2$$

1-5 Complements of Numbers

- Two types of complements for each base-r system:
  - radix complement: r's complement
    - e.g.s: 2's complement for binary numbers
      10's complement for decimal numbers
  - diminished radix complement: $(r – 1)$'s complement
    - e.g.s: 1's complement for binary numbers
      9's complement for decimal numbers
- The complement of the complement restores the number to its original value.

$(r-1)$'s complement

A. Diminished Radix Complement

- For a number N in **base r** having n digits:

$\Rightarrow$ $(r-1)$'s complement of N = $\underbrace{\mathbf{(r^n-1)}}_{(r-1)(r-1)\ \ldots\ (r-1)}$ $-$ N

$\Rightarrow$ n (r-1)'s

$\equiv$ subtracting each digit from (r – 1)

– E.g.s: $r = 10$

$$546700 \xrightarrow{\downarrow \text{ 9's comp}} \begin{array}{r} 999999 \\ -\ 546700 \\ \hline 453299 \end{array}$$

$$012398 \xrightarrow{\downarrow \text{ 9's comp}} \begin{array}{r} 999999 \\ -\ 012398 \\ \hline 987601 \end{array}$$

- For an n-bit **binary** number N:

$\Rightarrow$ 1's complement of N = $\underbrace{(2^n - 1)}_{\substack{11\ldots1 \\ \Rightarrow\ n\ 1\text{'s}}} - N$

$\equiv$ subtracting each digit from 1

$\equiv$ changing all 1's to 0's and all 0's to 1's

(applying the NOT op to each of the bits)

– E.g.s:

$$\begin{array}{r} 1011000 \\ \downarrow \text{1's comp} \\ 1111111 \\ -\ 1011000 \\ \hline 0100111 \end{array} \qquad \begin{array}{r} 0101101 \\ \downarrow \text{1's comp} \\ 1111111 \\ -\ 0101101 \\ \hline 1010010 \end{array}$$

r's complement

B. Radix Complement

- For a number N in **base *r*** having n digits:

$\Rightarrow$ r's complement of $N = r^n - N$ for $N \neq 0$ &

0 for $N = 0$

$\equiv$ adding 1 to the $(r-1)$'s complement of N $\rightarrow$ $\boxed{(r^n - 1) - N}$

$\equiv$ leaving all least significant 0's unchanged, subtracting the 1st nonzero LSD from r, and subtracting all higher significant digits from $r-1$

– E.g.s: $r = 10$

	012398		246700
	↓ 9's comp		↓ 9's comp
10's comp	987601	10's comp	753299
	↓ +1		↓ +1
	987602		753300

- For an n-bit **binary** number N:

$$\Rightarrow \text{2's complement of } N = \begin{cases} 2^n - N & \text{for } N \neq 0 \ \& \\ 0 & \text{for } N = 0 \end{cases}$$

$\equiv$ adding 1 to the 1's complement of N → $\boxed{(2^n - 1) - N}$

$\equiv$ leaving all least significant 0's and the 1st 1 unchanged and then replacing 1's w/ 0's and 0's w/ 1's in all other higher significant bits

- E.g.s:

1101100	0110111
↓ 1's comp	↓ 1's comp
2's comp 0010011	2's comp 1001000
↓ +1	↓ +1
0010100	1001001

C. **Unsigned** Binary Subtraction by (r–1)'s Complement Addition

- Subtraction of 2 n-digit unsigned numbers by **(r–1)'s** complement addition: $M - N = M + (-N)$

1. Add the (r–1)'s complement of the subtrahend N to the minuend M: $\mathbf{M + (r^n - 1 - N) = M - N + r^n - 1}$

   $$\mathbf{= r^n + (M - N - 1)}$$

2. If $M > N$, the sum produces an end carry, r^n. Discard the end carry and add one to the sum for the correct result of $M - N$. (end-around carry)
3. If $M \le N$, the sum does not produce an end carry. Perform a correction, taking the (r–1)'s complement of the sum and placing a minus sign in front to obtain the result. $-(\underbrace{r^n - 1 - (\underbrace{M - N + r^n - 1}_{\text{sum}})}_{\text{(r–1)'s-comp of sum}}) = -(N - M)$

   minus sign

Example

- Given two decimal numbers $M = 72532$ and $N = 3250$, perform the subtraction $M - N$ and $N - M$ using 9's complement addition.

<Ans.>

72532 – 03250

M – N

$$\begin{array}{rr} M = & 72532 \\ \text{9's complement of } N = & \underline{+\ 96749} \\ \text{Sum} = & 1\ \ 69281 \\ \text{End-around carry} & \underline{+\ 1} \\ \text{Answer: } M - N = & 69282 \end{array}$$

03250 – 72532

N – M

$$\begin{array}{rr} N = & 03250 \\ \text{9's complement of } M = & \underline{+\ 27467} \\ \text{Sum} = & 30717 \\ & \text{(No end carry)} \\ \text{Answer: } N - M = & -\text{(9's comp of sum)} \\ = & -\ 69282 \end{array}$$

Example 1.8

- Given the two binary numbers

$$X = 1010100 \text{ and } Y = 1000011,$$

84_{10} 67_{10}

perform the subtraction $X - Y$ and $Y - X$ using 1's complement ops.

<Ans.>

X – Y

X =	1010100
1's complement of Y =	+ 0111100
Sum =	1 0010000
End-around carry	+ 1
Answer: $X - Y$ =	0010001

17_{10}

Y – X

Y =	1000011
1's complement of X =	+ 0101011
Sum =	1101110
	(No end carry)
Answer: $Y - X$ =	– (1's comp of sum)
=	– 0010001

D. **Unsigned** Binary Subtraction by *r*'s Complement Addition

- Subtraction of 2 *n*-digit unsigned numbers by ***r*'s** complement addition: $M - N = M + (-N)$
  1. Add the minuend M to the r's complement of the subtrahend N: $M + (r^n - N) = M - N + r^n = r^n + (M - N)$
  2. If $M \geq N$, the sum produces an end carry, r^n. Discard the end carry, leaving result $M - N$.
  3. If $M < N$, the sum does not produce an end carry since it is equal to $r^n - (N - M)$. Perform a correction, taking the *r*'s complement of the sum and placing a minus sign in front to obtain the result $-(N - M)$.

$$\underset{\text{minus sign}}{-}\{\underbrace{r^n - [\underbrace{r^n - (N - M)}_{\text{sum}}]}_{r\text{'s-comp of sum}}\} = -(\mathrm{N} - \mathrm{M})$$

Example 1.5 & 1.6

- Given two decimal numbers $M = 72532$ and $N = 3250$, perform the subtraction $M - N$ and $N - M$ using 10's complement addition.

<Ans.>

72532 – 03250

M – N

M =	72532
10's complement of N =	+ 96750
Sum =	1 69282
Discard end carry 10^5 =	–1 00000
Answer: $M - N$ =	69282

03250 – 72532

N – M

N =	03250
10's complement of M =	+ 27468
Sum =	30718
	(No end carry)
Answer: $N - M$ =	– (10's comp of sum)
=	– 69282

Example 1.7

- Given two binary numbers

  $X = 1010100$ and $Y = 1000011$,

  84_{10} $\qquad$ 67_{10}

  perform the subtraction $X - Y$ and $Y - X$ using 2's complement addition.

<Ans.>

X – Y

X =	1010100
2's complement of Y =	+ 0111101
Sum =	1 0010001
Discard end carry 2^7 =	–1 0000000
Answer: $X - Y$ =	0010001

Y – X

Y =	1000011
2's complement of X =	+ 0101100
Sum =	1101111
	(No end carry)
Answer: $Y - X$ =	– (2's comp of sum)
=	– 0010001

1-6 **Signed** Binary Numbers

- Signed binary numbers:
  - Represent the sign w/ a bit placed in the most significant position of an *n*-bit number:
    - Convention: Sign bit = 0 for positive numbers
      = 1 for negative numbers

  * The ***user*** determines whether a string of bit is a number or not & whether the number is signed or unsigned.

- Representations of signed numbers:
  i. Signed-magnitude
  ii. Signed-complement:
     signed-1's complement & signed-2's complement

Representations of Signed Numbers

- Signed-magnitude representation:
  – The number consists of

    a magnitude and

    a symbol (+/–) or a bit (0/1) indicating the sign.
  – Negate a number: change its sign.

- Signed-complement representation:
  – A negative number is represented by its complement.
  – Negate a number: take its complement
  – can use either 1's or 2's complement (for a binary number)

Example

- E.g.: Represent +9 and –9 in binary w/ 8 bits

	Signed-magnitude	Signed-1's complement	Signed-2's complement
+9	00001001	00001001	00001001
–9	10001001	11110110	11110111

* **Which one of the representations will you choose for signed binary numbers?**

Comparison of Different Representations

$-(2^{n-1}-1) \sim +(2^{n-1}-1)$

$-2^{n-1} \sim +(2^{n-1}-1)$

- E.g.: 4-bit signed binary numbers

* The positive numbers in all 3 representations are identical and have 0 in the leftmost position &

all negative numbers have a 1 in the leftmost bit position.

* (a) (b): 7 positive numbers

2 zeros

7 negative numbers

* (c): 7 positive numbers

1 zero

8 negative numbers

Decimal	(a) Signed Magnitude	(b) Signed 1's Complement	(c) Signed 2's Complement
+ 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	1000	1111	—
− 1	1001	1110	1111
− 2	1010	1101	1110
− 3	1011	1100	1101
− 4	1100	1011	1100
− 5	1101	1010	1011
− 6	1110	1001	1010
− 7	1111	1000	1001
− 8	—	—	1000

A. Arithmetic Addition for Signed Numbers

- Addition for Signed-magnitude system: $M + N$
  - Basic idea:
    - The single sign bit in the leftmost position and the $n - 1$ magnitude bits are processed separately.
    - The magnitude bits are processed as unsigned binary numbers. $\Rightarrow$ Subtraction involves the correction step.
  - Follow the rules of ordinary addition arithmetic:
    - If the sign are the same, add the 2 magnitudes and give the sum the sign of M.
    - If the sign are different, subtract the magnitude of N from the magnitude of M. The absence or presence of an end borrow then determines the sign of the result based on the sign of M, and determines whether or not a 2's complement correction is performed.
  - E.g.: (0 0011001) + (1 0100101)

– E.g.: (0 0011001) + (1 0100101)

Two signed bits are different.

⇒ 0011001 – 0100101 = 1110100 & an end borrow of 1 occurs

⇒ The sign of the result = 1 (is opposite to that of *M*) & take the 2's complement of the magnitude of the result

1110100 → 0001100

⇒ The result = 1 0001100

- Addition for signed-2's complement system:

  (Negative numbers are represented in signed-2's complement form.)

  - Add the 2 numbers, including their sign bits. A carry out of the sign bit position is discarded.
    - Negative results are automatically in 2's complement form.
  - E.g.s: for 8-bit signed-2's complement binary numbers

+ 6	00000110
+ 13	00001101
+ 19	00010011

– 6	11111010
+ 13	00001101
+ 7	$\cancel{1}$00000111

+ 6	00000110
+ (–13)	11110011
– 7	11111001

– 6	11111010
+ (–13)	11110011
– 19	$\cancel{1}$11101101

*** Detection of "*overflow*" !**

B. Arithmetic Subtraction

- Subtraction for signed-2's complement system:

  (Negative numbers are represented in signed-2's complement form.)

  – 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.

$$\boxed{\begin{aligned}(\pm A)-(+B)&=(\pm A)+(-B)\\(\pm A)-(-B)&=(\pm A)+(+B)\end{aligned}}$$

  – Example 5-5:

$$\begin{array}{rrr} -6 & 11111010 & 11111010 \\ -(-13) & -11110011 & +00001101 \\ \hline +7 & & \cancel{1}00000111 \end{array}$$

$$\begin{array}{rrr} +6 & 00000110 & 00000110 \\ -(-13) & -11110011 & +00001101 \\ \hline +19 & & 00010011 \end{array}$$

  – ***Overflow***

Summary (1/2)

- Signed-magnitude system:
  - is used in ordinary arithmetic
  - is awkward when employed in computer arithmetic

    $\Leftarrow$ separate handling of the sign &

    the correction step required for subtraction (p.1-40)
- Signed-1's complement system:
  - is useful as a logical op
  - is seldom used for arithmetic ops

    $\Leftarrow$ 2 representations of 0 &

    end-around carry
- Signed-2's complement system (✓)
  - is used in computer arithmetic

Summary (2/2)

- In the ***signed-complement*** system, binary numbers are added and subtracted by the same basic addition and subtraction rules as are ***unsigned numbers***.
  - $\Rightarrow$ Computers need only one common HW ckt to handle both types of arithmetic.
  - $\Rightarrow$ The user or programmer must interpret the results of such addition or subtraction differently, depending on whether it is assumed that the numbers are signed or unsigned.

1-7 Binary Codes

- *Recall*-- Binary vs. Decimal number system
  - Binary: the most natural system for a computer
  - Decimal: people are accustomed to it
- *n*-bit binary code:
  - a group of *n* bits that assume up to 2^n distinct combinations of 1's and 0's
  - each combination represents one element of the set being coded
  - will have some unassigned bit combinations if the # of elements in the set is not a power of 2.
- Decimal codes:
  - represent the decimal digits (0 ~ 9) by a code that contains 1's and 0's

A. Binary-Coded Decimal (BCD) Code

- Binary-coded decimal (BCD):
  - 1010 ~ 1111 are not used and have no meaning.
  - A number w/ *n* decimal digits requires 4*n* bits in BCD.
    - E.g.: $(185)_{10} = (0001\ 1000\ 0101)_{BCD}$
      $= (10111001)_2$
  - Note: BCD numbers are decimal numbers and not binary numbers.
  - Adv.: Computer input and output data are handled by people who use the decimal system.

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

B. BCD Addition

- BCD addition:
  - In each position, use binary arithmetic to add the digits.
    If the binary sum is greater than 1001, add 0110 to obtain the correct BCD digits sum and a carry.
  - E.g.: 448 + 489 = 937 = $(1001\ 0011\ 0111)_{BCD}$

BCD carry	1	1	
	0100	0100	1000
	+ 0100	+ 1000	+ 1001
Binary sum	1001	1101	1 0001
Add 6		+ 0110	+ 0110
BCD sum		1 0011	1 0111

C. Other Decimal Codes

- Four different binary codes for the decimal digits:
  - BCD (8 4 2 1)
  - 2 4 2 1
  - Excess-3
  - 8 4 −2 −1

- **Weighted codes:**
  - each bit position is assigned a weighting factor
  - E.g.s: BCD (8421) code, 2421 code, 84-2-1 code
    - Some digits can be coded in two possible ways in the 2421 code.
- **Self-complementing codes:**
  - the 9's complement of a decimal number is obtained directly by changing 1's to 0's and 0's to 1's (1's comp)
  - E.g.s: 2421 & excess-3 codes
    - excess-3 code: each coded combination is obtained from the corresponding binary value plus 3

$(395)_{10}$ = 0110 1100 1000

↓ 9's comp ↓ 1's comp

$(604)_{10}$ = 1001 0011 0111

D. Gray Codes

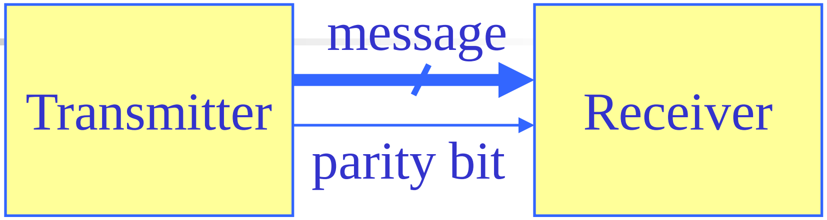
- Gray code:
  - Only one bit in the code group changes in going from one number to the next
  - is used in applications in which the normal sequence of binary numbers may produce an error or ambiguity during the transition from one number to the next.

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

E. Alphanumeric Codes

- ASCII character code: 7 bits, Table 1.7

F. Error-Detecting Code



- Parity bit:
  - is an extra bit included with a message to make the total number of 1's either even or odd.
  - is helpful in detecting errors during the transmission of information from one location to another.
- Even parity:
  - A parity bit is included to make the total # of 1s in the resulting code word even.
- Odd parity:
  - A parity bit is included to make the total # of 1s in the resulting code word odd.

- Example: ASCII A = 1000001, ASCII T = 1010100

		With **Even** Parity	With **Odd** Parity
A	1000001	01000001	11000001
T	1010100	11010100	01010100

1-8 Binary Storage and Registers

- Binary cell:
  - a device that possesses 2 stable states and is capable of storing one bit of information
- Register:
  - a group of binary cells
  - E.g.: a 16-bit register with content

    1100 0011 1100 1001

1-9 Binary Logic

- Binary logic:
  - deals with binary variables and with logic operations
    - binary variable: variable that take on two discrete values
    - basic logical operations: AND, OR, NOT
  - is used to describe the manipulation and processing of binary information.
  - resembles binary arithmetic, but should no be confused w/ each other.

Basic Logic Operations

- Basic logical ops:

AND

x y	$x \cdot y$
0 0	0
0 1	0
1 0	0
1 1	1

OR

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

NOT

x	x'
0	1
1	0

- **AND**: $\bullet$, $\wedge$
  - identical to binary multiplication
- **OR**: $+$, $\vee$
  - resemble binary addition

    In binary logic, 1 + 1 = 1

    In binary arithmetic: 1 + 1 = 10
- **NOT**: complement; $\bar{}$, $'$

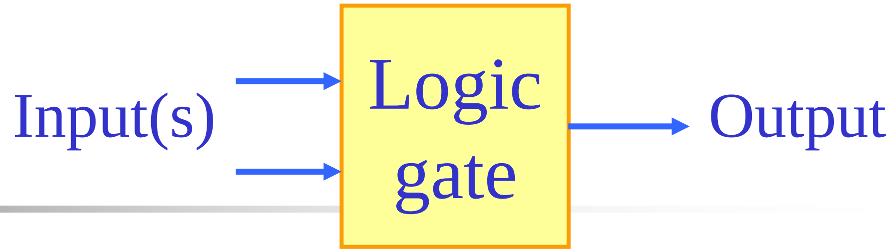
Truth Table

- Truth table:
  - a table of combinations of the binary variables showing the relationship b/t the values that the variables take on and the values of the result of the op.
  - n variables $\rightarrow 2^n$ rows
  - E.g.: truth table for AND op

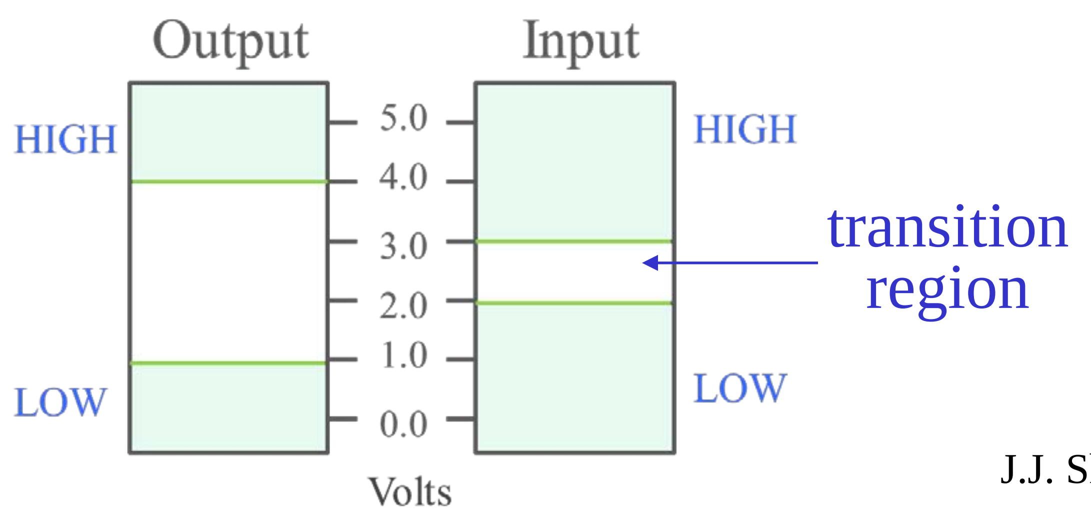
AND

x	y	$x \cdot y$
0	0	0
0	1	0
1	0	0
1	1	1

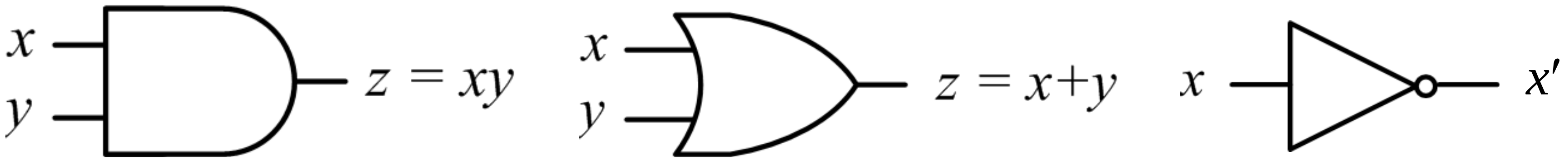
Logic Gates



- Logic gate:
  - is an electronic ckt that operate on one or more input signals to produce an output signal.
  - Electrical signals (voltages or currents) exist throughout a digital system in either of two recognizable values.
    - The input terminals of logic gates accept binary signals within the allowable range and respond at the output terminals w/ binary signals that fall within a specified range.
    - E.g.:



- Graphic symbols of 3 basic logic gates:



Two-input AND gate　　Two-input OR gate　　NOT gate (inverter)

- Multiple-input logic gates:
  - AND and OR gates may have ≥ 2 inputs.

Chapter Summary

- Digital Systems
- Number Systems
  - Binary Numbers
  - Number Base Conversions
  - Octal and Hexadecimal Numbers
  - Complements
  - Signed Binary Numbers
- Binary Codes
- Binary Storage and Registers
- Binary Logic

Problems & Homework (6th ed.)

Sections	Exercises	Homework
§1-2	1.2, 1.11, 1.12	1.1, 1.12*
§1-3	1.1, 1.3~1.6, 1.13	1.3, 1.5*
§1-4	1.7~1.10	1.7*
§1-5	1.14~1.18	1.14, 1.18
§1-6	1.19~1.21	1.20
§1-7	1.22~1.34	1.23, 1.33*