

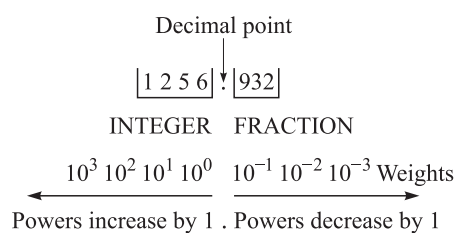
Appendix A

Number Systems and Codes

Number systems and computer codes are reviewed in this appendix. The books listed as references provide further details on these topics.

A number system consists of a set of symbols called *digits* and a set of relations defining the addition (+), subtraction (−), multiplication (×), and division (÷) of the digits. The total number of digits is the *radix* (or *base*) of the number system. In the “positional notation” of a number, the *radix point* separates the integer portion of the number from its fraction portion and if there is no fraction portion, the radix point is also omitted from the representation. Further, in this notation, each position has a weight equivalent to a power of the base. The power starts with 0 and increases by 1 as we move each position to the left of the radix point, and decreases by 1 as we move to the right. A typical number in decimal system is shown in Example A.1.

Example A.1



This number also can be represented as the polynomial

$$1 \times 10^3 + 2 \times 10^2 + 5 \times 10^1 + 6 \times 10^0 + 9 \times 10^{-1} + 3 \times 10^{-2} + 2 \times 10^{-3}.$$

A general positional representation of a number N is

$$N = (a_n \cdots a_3 a_2 a_1 a_0 \cdot a_{-1} a_{-2} a_{-3} \cdots a_{-m})_r. \quad (\text{A.1})$$

Where r is the radix of the number system, a_{-1} , a_0 , a_1 , a_2 , etc. are digits such that $0 \leq a_i \leq (r - 1)$ for all i ; a_n is the most significant digit (MSD) and a_{-m} is the least significant digit (LSD). The polynomial representation of the above number is

$$N = \sum_{i=-m}^n a_i r^i. \quad (\text{A.2})$$

The largest integer that can be represented with n digits is $(r^n - 1)$. Table A.1 lists the first few numbers in various systems. We will discuss binary, octal, and hexadecimal number systems. Digital computers employ the binary system, but octal and hexadecimal systems are useful in representing binary information in a shorthand form.

A.1 BINARY SYSTEM

In this system, radix (r) is 2 and the allowed *binary digits* (bits) are 0 and 1. A typical number is shown below in the positional notation.

Example A.2

$N = (11010$	$\cdot$	$1101)_2$	
$2^4 \ 2^3 \ 2^2 \ 2^1 \ 2^0$	$\cdot$	$2^{-1} 2^{-2} 2^{-3} 2^{-4}$	Weights
16 8 4 2 1	$\cdot$	$\frac{1}{2} \ \frac{1}{4} \ \frac{1}{8} \ \frac{1}{16}$	Weights in decimal
$\longleftarrow$		$\longrightarrow$	
Weights double for each move to left from binary point.		Weights are halved for each move to right from binary point.	

In polynomial form, the above number is

$$\begin{aligned}
 N &= 1 \times 2^4 + 1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 0 \times 2^0 \\
 &\quad + 1 \times 2^{-1} + 1 \times 2^{-2} + 0 \times 2^{-3} + 1 \times 2^{-4} \\
 &= 16 + 8 + 0 + 2 + 0 + \frac{1}{2} + \frac{1}{4} + 0 + \frac{1}{16} \text{ Decimal} \\
 &= 26 + \frac{1}{2} + \frac{1}{4} + \frac{1}{16} \text{ Decimal} \\
 &= (26\frac{13}{16})_{10}
 \end{aligned}$$

Thus, to convert a number from binary to decimal, we can simply accumulate the weights corresponding to each nonzero bit of the binary number.

Table A.1 Number Systems

Decimal $r = 10$	Binary $r = 2$	Ternary $r = 3$	Quaternary $r = 4$	Octal $r = 8$	Hexadecimal $r = 16$
0	0	0	0	0	0
1	1	1	1	1	1
2	10	2	2	2	2
3	11	10	3	3	3
4	100	11	10	4	4
5	101	12	11	5	5
6	110	20	12	6	6
7	111	21	13	7	7
8	1000	22	20	10	8
9	1001	100	21	11	9
10	1010	101	22	12	A
11	1011	102	23	13	B
12	1100	110	30	14	C
13	1101	111	31	15	D
14	1110	112	32	16	E
15	1111	120	33	17	F
16	10000	121	100	20	10
17	10001	122	101	21	11
18	10010	200	102	22	12
19	10011	201	103	23	13
20	10100	202	110	24	14

Each bit can take the value of a 0 or a 1. With 2 bits, we can derive four (2^2) distinct combinations: 00, 01, 10, and 11. The decimal values of these patterns range from 0 to 3 (i.e., $2^2 - 1$). In general, with n bits it is possible to have 2^n combinations of 0s and 1s and these combinations, when viewed as binary numbers, take values from 0 to $(2^n - 1)_{10}$. Table A.2 shows some binary numbers. A procedure for writing all the combinations of 0s and 1s for any n follows.

With n bits, 2^n combinations are possible. The first combination contains n 0s and the last contains n 1s. To generate the other combinations, note that in Table A.2, in the bit position i ($0 \leq i \leq n - 1$), the 0s and 1s alternate every 2^i rows as we move from row to row. The least significant bit (LSB) corresponds to $i = 0$ and the most significant bit (MSB) corresponds to $i = n - 1$. That is, starting with a 0 in the first row, in the LSB position ($i = 0$), the values of bits alternate every row ($2^i = 2^0 = 1$); the values in bit position 1 ($i = 1$), alternate every two rows, etc. This procedure is repeated through the MSB ($i = n - 1$) position, to generate all the 2^n combinations.

Table A.2 Binary Numbers

$n = 2$	$n = 3$	$n = 4$
1 0 00 01 10 11	2 1 0 000 001 010 011 100 101 110 111	3 2 1 0 ← Bit position 0000 0001 0010 0011 0100 0101 0110 0111 1000 1001 1010 1011 1100 1101 1110 1111

A.2 OCTAL SYSTEM

In this system, $r = 8$ and the allowed digits are 0, 1, 2, 3, 4, 5, 6, and 7. A typical number is shown below in positional notation.

Example A.3

$$\begin{aligned}
 N &= (4\ 5\ 2\ 6\cdot\ 2\ 3)_8 \\
 &\quad 8^3 8^2 8^1 8^0 \cdot 8^{-1} 8^{-2} && \text{Weights} \\
 &= 4 \times 8^3 + 5 \times 8^2 + 2 \times 8^1 + 6 \times 8^0 + 2 \times 8^{-1} + 3 \times 8^{-2} \leftarrow \\
 & && \text{Polynomial form} \\
 &= 2048 + 320 + 16 + 6 + \frac{2}{8} + \frac{3}{64} && \text{Decimal} \\
 &= (2390\frac{19}{64})_{10}
 \end{aligned}$$

A.3 HEXADECIMAL SYSTEM

In this system, $r = 16$, and the allowed digits are 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, and F. Digits A through F correspond to decimal values 10 through 15. A typical number is shown in Example A.4.

Example A.4

$$\begin{aligned}
 N &= (A \ 1 \ F \ . \ 1 \ C)_{16} \\
 &\quad 16^2 16^1 16^0 \cdot 16^{-1} 16^{-2} && \text{Weights} \\
 &= A \times 16^2 + 1 \times 16^1 + F \times 16^0 + 1 \times 16^{-1} + C \times 16^{-2} \leftarrow \begin{array}{l} \text{Polynomial form} \end{array} \\
 &= 10 \times 16^2 + 1 \times 16^1 + 15 \times 16^0 + 1 \times 16^{-1} + 12 \times 16^{-2} \\
 &= (2591 \frac{28}{256})_{10} && \text{Decimal}
 \end{aligned}$$

A.4 CONVERSION

To convert numbers from any system to decimal, we simply expand the given number as a polynomial and evaluate the polynomial using decimal arithmetic, as we have seen in Examples A.1–A.4.

To convert an integer number in decimal to any other system, we use the *radix divide* technique. This technique requires the following steps:

1. Divide the given number successively by the required base, noting the remainders at each step; the quotient at each step becomes the new dividend for subsequent division. Stop division when the quotient reaches 0.
2. Collect the remainders from each step (last to first) and place them left to right to form the required number.

This procedure is illustrated by the following examples.

Example A.5

$$(245)_{10} = (?)_2 \text{ (i.e., convert } (245)_{10} \text{ to binary).}$$

Required	→ 2	245		
base	2	122	1	↑
	2	61	0	
	2	30	1	
	2	15	0	
	2	7	1	
	2	3	1	
	2	1	1	
		0	1	
				$= (1111\ 0101)_2$

Example A.6

$$(245)_{10} = (?)_8.$$

8	245		
8	30	5	↑
8	3	6	
	0	3	
			$= (365)_8$

Example A.7

$$(245)_{10} = (?)_{16}.$$

16	245		
16	15	5 = 5	↑
	0	15 = F	
			$= (F5)_{16}$

To convert a fraction in decimal to any other system, the *radix multiply* technique is used. The steps in this method are as follows:

1. Successively multiply the given fraction by the required base, noting the integer portion of the product at each step. Use the fractional part of the product as the multiplicand for subsequent steps. Stop when the fraction either reaches 0 or recurs.
2. Collect the integer digits at each step from first to last and arrange them left to right.

It is not possible to represent a given decimal fraction in binary exactly if the radix multiplication process does not converge to 0. The accuracy, then, depends on the number of bits used to represent the fraction. Some examples follow.

Example A.8

$$(0.250)_{10} = (?)_2.$$

$$\begin{array}{r}
 0.25 \\
 \times 2 \\
 \hline
 0.50 \\
 \times 2 \\
 \hline
 1.00
 \end{array}
 \downarrow
 \begin{array}{l}
 \\
 \\
 \end{array}
 = (0.01)_2$$

Example A.9

$$(0.345)_{10} = (?)_2.$$

$$\begin{array}{r}
 0.345 \\
 \times 2 \\
 \hline
 0.690 \\
 \times 2 \\
 \hline
 1.380 \\
 \times 2 \\
 \hline
 0.760 \\
 \times 2 \\
 \hline
 1.520 \\
 \times 2 \\
 \hline
 1.040 \\
 \times 2 \\
 \hline
 0.080
 \end{array}
 \begin{array}{l}
 \\
 \\
 \\
 \\
 \\
 \\
 \\
 \end{array}
 \begin{array}{l}
 \\
 \\
 \\
 \text{Multiply fractions only.} \\
 \\
 \\
 \end{array}
 \downarrow
 = (0.010110)_2$$

The fraction may never reach zero; stop when the required number of fraction digits are obtained, the fraction will not be exact.

Example A.10

$$(0.345)_{10} = (?)_8.$$

$$\begin{array}{r}
 0.345 \\
 \times 8 \\
 \hline
 2.760 \\
 \times 8 \\
 \hline
 6.080 \\
 \times 8 \\
 \hline
 0.640 \\
 \times 8 \\
 \hline
 5.120 = (0.2605)_8
 \end{array}$$

Example A.11

$$(242.45)_{10} = (?)_2.$$

2	242		0.45
2	121	0	$\times 2$
			$\hline 0.90$
2	60	1	$\times 2$
			$\hline 1.80$
2	30	0	$\times 2$
			$\hline 0.60$
2	15	0	$\times 2$
			$\hline 1.20$
2	7	1	$\times 2$
			$\hline 0.40$
2	3	1	$\times 2$
			$\hline 0.80$
2	1	1	$\times 2$
			$\hline 1.60$
	0	1	

$= (1111\ 0010 . 01\ 1100)_2$

*Repeats

The radix divide and multiply algorithms are applicable to the conversion of numbers from any base to any other base. In converting a number from base p to base q , the number in base p is divided (or multiplied) by q in

base p arithmetic. These methods are convenient when p is 10 because of our familiarity with decimal arithmetic. In general, it is easier to convert a base p number to base q ($p \neq 10, q \neq 10$) by first converting the number to decimal from base p and then converting this decimal number into base q ; that is, $(N)_p \rightarrow (?)_{10} \rightarrow (?)_q$, as shown by Example A.12.

Example A.12

$$(25.34)_8 = (?)_5.$$

Convert to base 10

$$\begin{aligned} (25.34)_8 &= 2 \times 8^1 + 5 \times 8^0 + 3 \times 8^{-1} + 4 \times 8^{-2} \text{ Decimal} \\ &= 16 + 5 + \frac{3}{8} + \frac{4}{64} \text{ Decimal} \\ &= (21\frac{28}{64})_{10} \\ &= (21.4375)_{10} \end{aligned}$$

Convert to base 5

5	21		0.4375	
	4	1	<u> </u> × 5	
	0	4	2.1875	
			<u> </u> × 5	
			0.9375	
			<u> </u> × 5	
			4.6875	
			<u> </u> × 5	
			3.4375	
			<u> </u> × 5	
			2.1875	*Repeats

$= (41.\overline{2043})_5$

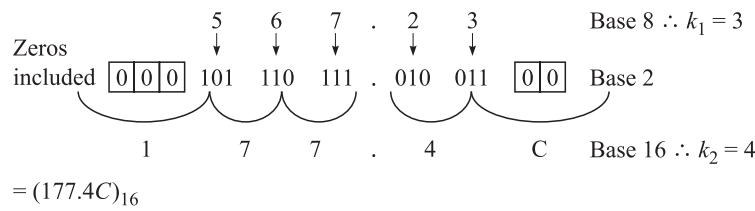
A.4.1 Base 2^k Conversion

When converting a number from base p to base q , if p and q are both powers of 2 an easier conversion procedure may be used. Here, the number in base p is first converted into binary, which number is then converted into base q . This is called *base 2^k conversion*.

Let $p = 2^{k_1}$. In base 2^k conversion, each digit of the number in base p is first expanded into a k_1 -bit binary number. The bits of the number thus

Example A.15

$$(567.23)_8 = (?)_{16}$$



A.5 ARITHMETIC

This section provides a review of binary, octal, and hexadecimal arithmetics. Arithmetic in any number system is similar to that in the decimal system. Binary arithmetic is easier to perform (and understand) than decimal arithmetic since only two digits are involved. Arithmetic in octal and hexadecimal systems is more complex and may require practice if we are unfamiliar with those systems.

A.5.1 Binary Arithmetic

Table A.3 shows the rules for binary addition, subtraction, and multiplication.

Table A.3 Binary Arithmetic

A			A			A		
A + B	0	1	A - B	0	1	A × B	0	1
B 0	0	1	0	0	1	0	0	0
B 1	1	10	1	11	0	1	0	1
CARRY SUM			BORROW DIFFERENCE					
(a) Addition			(b) Subtraction			(c) Multiplication		

Example A.20

1011	Multiplicand	
<u>× 1100</u>	Multiplier	
		Multiplier bits
0000		(1011) × 0
0000		(1011) × 0
1011		(1011) × 1
<u>1011</u>		(1011) × 1
10000100	PRODUCT	

In general, the **PRODUCT** of two n -bit numbers is $(2n)$ bits long. To multiply two n -bit numbers **A** and **B**, where $\mathbf{B} = (b_{n-1}b_{n-2} \cdots b_1b_0)$, the following procedure can be used:

1. Start with a $2n$ -bit **PRODUCT** with a zero value (all 0s).
2. For each $b_i (0 \leq i \leq n-1) \neq 0$: shift **A**, i positions to the left and add to the **PRODUCT**.

Thus, binary multiplication can be performed by repeated shifting and addition of the multiplicand to the partial product, to obtain the **PRODUCT**.

Division

Division is the repeated subtraction of the divisor from the dividend. The longhand procedure of decimal division can also be used in binary.

Example A.21

110101 ÷ 111.		
0111	Quotient	
111 $\overline{) 110,101}$		110 < 111 ∴ $q_1 = 0$
<u>−000</u>		
1101		1101 > 111 ∴ $q_2 = 1$
<u>−111</u>		
1100		1100 > 111 ∴ $q_3 = 1$
<u>−111</u>		
1011		1011 > 111 ∴ $q_4 = 1$
<u>−111</u>		
100	Remainder	

Note that the division is repeated subtraction of the divisor from the dividend.

References listed at the end of this appendix provide the details of several other binary multiplication and division algorithms. Hardware implementation of binary addition and subtraction is discussed in Chapter 1 of this book and multiplication and division algorithms are given in Chapter 10.

Shifting

In general, shifting a base r number left by one position (and inserting a 0 into the vacant LSD position) is equivalent to multiplying the number by r ; and shifting the number right by one position (inserting a 0 into the vacant MSD position) is equivalent to dividing the number by r .

In the binary system, each left shift multiplies the number by 2 and each right shift divides the number by 2, as shown by Example A.22.

Example A.22

	Binary	Decimal
N	01011.11	$11\frac{3}{4}$
$2 \cdot N$	10111.1 0 ← INSERT	$23\frac{1}{2}$
$N \div 2$	INSERT 0 0101.11	$5\frac{3}{4}$ (Inaccurate, since only 2-bit accuracy is retained.)

If the MSB of an n -bit number is not 0, shifting left would result in a number larger than the magnitude that can be accommodated in n bits and the 1 shifted out of MSB cannot be discarded; if nonzero bits shifted out of LSB bit during a right shift are discarded, the accuracy is lost.

A.5.2 Octal Arithmetic

Table A.4 shows the addition and multiplication tables for octal arithmetic. The following examples illustrate the four arithmetic operations and their similarity to decimal arithmetic.

Table A.4 Octal Arithmetic

A + B	A							
	0	1	2	3	4	5	6	7
0	0	1	2	3	4	5	6	7
1	1	2	3	4	5	6	7	10
2	2	3	4	5	6	7	10	11
B 3	3	4	5	6	7	10	11	12
4	4	5	6	7	10	11	12	13
5	5	6	7	10	11	12	13	14
6	6	7	10	11	12	13	14	15
7	7	10	11	12	13	14	15	16

(a) Addition

A × B	A							
	0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0	0
1	0	1	2	3	4	5	6	7
2	0	2	4	6	10	12	14	16
B 3	0	3	6	11	14	17	22	25
4	0	4	10	14	20	24	30	34
5	0	5	12	17	24	31	36	43
6	0	6	14	22	30	36	44	52
7	0	7	16	25	34	43	52	61

(b) Multiplication

Example A.23 *Addition.*

$ \begin{array}{r} 1\ 1\ 1 \\ 1\ 4\ 7\ 6 \\ +\ 3\ 5\ 5\ 4 \\ \hline 5\ 2\ 5\ 2 \end{array} $	← Carries SUM	Scratch pad $ \begin{array}{l} 6 \\ +4 \\ \hline (10)_{10} = (12)_8 \\ 1 \\ +7 \\ \hline +5 \\ (13)_{10} = (15)_8 \\ 1 \\ +4 \\ \hline +5 \\ (10)_{10} = (12)_8 \\ 1 \\ +1 \\ \hline +3 \\ (5)_{10} = (5)_8 \end{array} $
--	-----------------------------	--

Example A.24 *Subtraction.*

$ \begin{array}{r} 4\ 14 \\ \cancel{5}\ \cancel{4}\ 7\ 5 \\ -\cancel{3}\ \cancel{7}\ 6\ 4 \\ \hline 1\ 5\ 1\ 1 \end{array} $	Digit position 2 Borrow from position 3. ∴	Octal $ \begin{array}{r} 14_8 \\ -7_8 \\ \hline 5_8 \end{array} $	Decimal $ \begin{array}{r} 12 \\ -7 \\ \hline 5_{10} = 5_8 \end{array} $
--	---	---	--

Example A.25

$$\begin{array}{r}
 3\ 7\ 7 \\
 5\ \cancel{4}\ \emptyset\ \emptyset\ 4\ 5 \\
 -\cancel{3}\ \cancel{2}\ \cancel{5}\ \cancel{6}\ \cancel{5}\ \cancel{4} \\
 \hline
 2\ 1\ 2\ 1\ 7\ 1
 \end{array}$$

The intermediate 0s become $(r - 1)$ or 7 when borrowed.

Example A.26 *Multiplication.*

$ \begin{array}{r} 543 \\ \times 27 \\ \hline 4665 \\ 1306 \\ \hline 17745 \end{array} $		Scratch pad $3 \times 7 = (21)_{10} = (25)_8$ ← These can be $4 \times 7 = (28)_{10} = (34)_8$ obtained directly $5 \times 7 = (35)_{10} = (43)_8$ from Table A.4. $\begin{array}{r} 25 \\ 34 \\ 43 \\ \hline 4665 \end{array}$ $\begin{array}{r} 3 \times 2 = (6)_{10} = (6)_8 \\ 4 \times 2 = (8)_{10} = (10)_8 \\ 5 \times 2 = (10)_{10} = (12)_8 \end{array}$ $\begin{array}{r} 6 \\ 10 \\ 12 \\ \hline 1306 \end{array}$
--	--	---

Product

Example A.27 *Division.* $543 \div 7$.

$$\begin{array}{r}
 7 \overline{) 543} \\
 \underline{0} \\
 54 \\
 \underline{52} \\
 23 \\
 \underline{16} \\
 5
 \end{array}$$

Use the multiplication table of Table A.4 to derive the quotient digit (by trial and error).

A.5.3 Hexadecimal Arithmetic

Table A.5 provides the addition and multiplication tables for hexadecimal arithmetic. The following examples illustrate the arithmetic.

Table A.5 Hexadecimal Arithmetic

+	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
1	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	10
2	2	3	4	5	6	7	8	9	A	B	C	D	E	F	10	11
3	3	4	5	6	7	8	9	A	B	C	D	E	F	10	11	12
4	4	5	6	7	8	9	A	B	C	D	E	F	10	11	12	13
5	5	6	7	8	9	A	B	C	D	E	F	10	11	12	13	14
6	6	7	8	9	A	B	C	D	E	F	10	11	12	13	14	15
7	7	8	9	A	B	C	D	E	F	10	11	12	13	14	15	16
8	8	9	A	B	C	D	E	F	10	11	12	13	14	15	16	17
9	9	A	B	C	D	E	F	10	11	12	13	14	15	16	17	18
A	A	B	C	D	E	F	10	11	12	13	14	15	16	17	18	19
B	B	C	D	E	F	10	11	12	13	14	15	16	17	18	19	1A
C	C	D	E	F	10	11	12	13	14	15	16	17	18	19	1A	1B
D	D	E	F	10	11	12	13	14	15	16	17	18	19	1A	1B	1C
E	E	F	10	11	12	13	14	15	16	17	18	19	1A	1B	1C	1D
F	F	10	11	12	13	14	15	16	17	18	19	1A	1B	1C	1D	1E

(a) Addition

×	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
2	0	2	4	6	8	A	C	E	10	12	14	16	18	1A	1C	1E
3	0	3	6	9	C	F	12	15	18	1B	1E	21	24	27	2A	2D
4	0	4	8	C	10	14	18	1C	20	24	28	2C	30	34	38	3C
5	0	5	A	F	14	19	1E	23	28	2D	32	37	3C	41	46	4B
6	0	6	C	12	18	1E	24	2A	30	36	3C	42	48	4E	54	5A
7	0	7	E	15	1C	23	2A	31	38	3F	46	4D	54	5B	62	69
8	0	8	10	18	20	28	30	38	40	48	50	58	60	68	70	78
9	0	9	12	1B	24	2D	36	3F	48	51	5A	63	6C	75	7E	87
A	0	A	14	1E	28	32	3C	46	50	5A	64	6E	78	82	8C	96
B	0	B	16	21	2C	37	42	4D	58	63	6E	79	84	8F	9A	A5
C	0	C	18	24	30	3C	48	54	60	6C	78	84	90	9C	A8	B4
D	0	D	1A	27	34	41	4E	5B	68	75	82	8F	9C	A9	B6	C3
E	0	E	1C	2A	38	46	54	62	70	7E	8C	9A	A8	B6	C4	D2
F	0	F	1E	2D	3C	4B	5A	69	78	87	96	A5	B4	C3	D2	E1

(b) Multiplication

Scratch pad

Decimal

$$\begin{array}{r} 1 \quad 1 \\ 1 \quad 5 \quad \text{F} \quad \text{C} \\ + 2 \quad 4 \quad 5 \quad \text{D} \\ \hline 3 \quad \text{A} \quad 5 \quad 9 \end{array}$$

$\text{C} = 12$
 $\text{D} = 13$

$$\begin{array}{r} 16 \overline{) 25} = (19)_{16} \\ 16 \overline{) 1} \quad 9 \uparrow \\ \quad 0 \quad 1 \end{array}$$

← These can be obtained directly from Table A.5.

Decimal

$$\begin{array}{l} 1 = 1 \\ \text{F} = 15 \\ 5 = 5 \\ \underline{21} = (15)_{16} \end{array}$$

Scratch pad	
1 13 15	Decimal
3	
2 4 5 D Minuend	$(15)_{16} = 21$
<u>- 1 5 F C</u> Subtrahend	$-(F)_{16} = \underline{-15}$
0 E 6 1 Difference	$6 = (6)_{16}$
	$(13)_{16} = 19$
	$-(5)_{16} = \underline{-5}$
	$14 = (E)_{16}$

Example A.30 *Multiplication.*

Scratch pad

	Decimal	Hexadecimal
$A \times 2 = 20 =$	1 4	
$4 \times 2 = 8 =$	0 8	
$E \times 2 = 28 =$	1 C	
$1 \times 2 = 2 =$	0 2	
		0 3 C 9 4 = P_1
$A \times A = 100 =$	6 4	
$4 \times A = 40 =$	2 8	
$E \times A = 140 =$	8 C	
$1 \times A = 10 =$	A	
		1 2 E E 4 = P_2
$A \times F = 150 =$	9 6	
$4 \times F = 60 =$	3 C	
$E \times F = 210 =$	D 2	
$1 \times F = 15 =$	D	
		1 C 6 5 6 = P_3

Example A.31 *Division.* 1 A F 3 $\div$ E.

	0 1 E C	Quotient
E	1 A F 3	
	0	
	1 A	
	E	
	C F	
	C 4	
	B 3	
	A 8	
	B	Remainder

A.6 SIGN-MAGNITUDE REPRESENTATION

In this representation, the MSD of the number represents the sign. It is 0 for positive numbers and $(r - 1)$ for negative numbers, where r is the base of the number system. Some representation examples follow.

Example A.32

$$(+25)_{10} = 0.0025$$

$$(-25)_{10} = 9.0025$$

$$(+10)_2 = 0.0010$$

$$(-10)_2 = 1.0010$$

$$(+56)_8 = 0.0056$$

$$(-56)_8 = 7.0056$$

$$(+1F)_{16} = 0.001F$$

$$(-1F)_{16} = F.001F$$

↑

Sign, magnitude

All numbers are shown as five-digit numbers. In arithmetic with sign-magnitude numbers, the magnitude and sign are treated separately.

A.7 COMPLEMENT NUMBER SYSTEMS

Consider the subtraction of a number A from a number B. This is equivalent to adding $(-A)$ to B. The complement number system provides a convenient way of representing negative numbers (that is, complements of positive numbers), thus converting the subtraction to an addition. Since multiplication and division correspond to repeated addition and subtraction, respectively, it is possible to perform the four basic arithmetic operations using only the hardware for addition when negative numbers are represented in complement form. There are two popular complement number systems, (1) *radix complement* and (2) *diminished radix complement*.

A.7.1 Radix Complement System

This system is called either 2s complement or 10s complement system, depending on whether the base of the number is 2 or 10. We will discuss

the 2s complement system. The 10s complement system also displays the same characteristics as the 2s complement system.

The radix complement of a number $(N)_r$ is defined as

$$[N]_r = r^n - (N)_r \quad (\text{A.3})$$

where $[N]_r$ is the radix complement of $(N)_r$ and n is the number of digits in $(N)_r$.

Example A.33 The 2s complement of $(01010)_2$ is $2^5 - (01010)_2$; $n = 5$, $r = 2$.

$$\begin{aligned} &= 100000 - 01010 \\ &= 10110 \\ \therefore [01010]_2 &= (10110)_2. \end{aligned}$$

There are two other methods for obtaining the 2s complement of a number.

Method 1

$[01010]_2 = ?$

10101 $\leftarrow$ Complement each bit (i.e., change each 0 to 1 and 1 to 0).

$\begin{array}{r} + 1 \\ 10110 \end{array}$ $\leftarrow$ Add 1 to the LSB to get the 2s complement.

Method 2

$[01010]_2 = ?$

$\begin{array}{r|l} 010 & 10 \\ \hline & 10 \\ & 101 \\ \hline 101 & 10 \end{array}$ $\leftarrow$ Copy the bits from LSB until and including the first nonzero bit.
 $\leftarrow$ Complement the remaining bits through MSB to get the 2s complement.

Twos Complement Arithmetic

In the 2s complement system, positive numbers remain in their normal form while the negative numbers are represented in the 2s complement form. Some examples using a 5-bit (4-magnitude, 1-sign) representation are presented.

Example A.34

+5	0 0101	
−5	1 1011	2s complement of +5
+4	0 0100	
−4	1 1100	

Examples of arithmetic using these 5-bit 2s complement numbers are shown in the examples below:

Example A.35

	Sign-magnitude	2s complement	
5	0, 0101	0 0101	
−4	1, 0100	1 1100	(2s complement of +4)


 CARRY from sign position 10 0001 SUM
 = +(0001)₂

The CARRY from the MSB (sign bit in this case) is ignored. The result is positive because the MSB is 0.

Example A.36

	Sign-magnitude	2s complement	
4	0, 0100	0 0100	
−5	1, 0101	1 1011	
		1 1111 SUM	
		= −(0001) ₂	

There is no CARRY from the MSB. Since MSB is 1, the result is negative and must be 2s-complemented to find its value, that is, −(0001)₂ or −1.

To summarize, in 2s complement addition if the MSB generates a CARRY, it is ignored and the result is treated as positive; if there is no CARRY, the result is negative and is in the 2s complement format. Note that in this arithmetic the sign bit is treated as though it is one of the magnitude bits.

Shifting

Shifting a 2s complement number left or right by one bit is equivalent to multiplying it by 2 or dividing it by 2, respectively. The sign bit is copied into the vacant position during a right shift and 0 is inserted into LSB during a left shift. An example follows.

Example A.37

1 0 0 1 0 1 0 0 0 0									
1	1	0	0	1	0	1	0	0	0
1	1	1	0	0	1	0	1	0	0
1	1	0	0	1	0	1	0	0	0
1	0	0	1	0	1	0	0	0	0
0	0	1	0	1	0	0	0	0	0

Right shift (copy sign bit.)

Right shift (copy sign bit.)

Left shift (Insert 0.)

Left shift (Insert 0.)

Left shift (Insert 0) , OVERFLOW.

Change in the value of the sign bit during a left shift indicates that there is an overflow (i.e., the result is too large), as shown by the last case above in this example.

A.7.2 Diminished Radix Complement

The diminished radix complement $[N]_{r-1}$ of a number $(N)_r$ is defined as

$$[N]_{r-1} = r^n - (N)_r - 1 \quad (\text{A.4})$$

where n is the number of digits in $(N)_r$. Note that

$$[N]_r = [N]_{r-1} + 1. \quad (\text{A.5})$$

The diminished radix complement is called 1s complement or 9s complement, depending on whether the base of the system is 2 or 10, respectively.

Since $(2^n - 1)$ is an n -bit binary number with all of its bits equal to 1, the 1s complement of a number is obtained by subtracting each of its bits from 1. This is the same as complementing each bit. As an example, the 1s complement of $(10110)_2$ is $(01001)_2$. The 9s complement of a decimal number is obtained by subtracting each of its digits from 9.

Addition

In the 1s complement number system, the positive numbers remain in their true binary form while the negative numbers are represented in the 1s complement form. The following examples illustrate the arithmetic in this system using a 5-bit (4-magnitude, 1-sign) representation.

Example A.38

	Sign-magnitude	1s complement	
5	0,0101	0 0101	
-4	1,0100	<u>1 1011</u>	
		1 0 0000	SUM
Add the CARRY end-round →			
		<u>1</u>	
		0 0001	

The CARRY generated from MSB during the addition is added (end-around) to the LSB to obtain the correct SUM; the result is positive, because MSB=0.

Example A.39

	Sign-magnitude	1s complement	
4	0,0100	0 0100	
-5	1,0101	<u>1 1010</u>	
		1 1110	SUM
		↑ Negative.	
		∴ Complement and the result is -(0001).	

Shifting

While shifting 1s complement numbers right, the sign bit is copied into the vacant positions on the left (as in the 2s complement system); during a left shift, sign bit is copied into the vacant positions on the right.

Table A.6 summarizes the operations in both complement systems. Some digital systems use a 2s complement system while others employ 1s complement.

Table A.6 Complement Number Systems

Operation	Result	1s complement	2s complement
ADD	MSB is 1	Result is in 1s complement	Result is in 2s complement
	CARRY from MSB	Add 1 to LSB of the result	Neglect the CARRY
Shift left		Copy sign bit on the right	Copy 0s on the right
Shift right		Copy sign bit on the left	Copy sign bit on the left

A.8 CODES

A digital system requires that all its information be in binary form. But the external world uses the alphabetic characters, decimal digits, and special characters (e.g., periods, commas, plus and minus signs) to represent information. A unique pattern of 0s and 1s is used to represent each required character. This pattern is the “code” corresponding to that character. We will list several codes that are commonly used in digital systems.

A.8.1 Binary Coded Decimal (BCD)

BCD is one of the popular codes used to represent numeric information. Each decimal digit is coded into 4 bits:

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

Only ten of the possible sixteen (2^4) patterns of bits are used. As an example, the number $(432)_{10}$ would be represented as

$$\begin{array}{ccc} (0100 & 0011 & 0010)_{\text{BCD}} \\ 4 & 3 & 2 \end{array}$$

When the data are represented in BCD, the arithmetic can also be done on each digit, as in decimal arithmetic.

A.8.2 Excess-3

Excess-3 is also a 4-bit code. The code for each decimal digit is obtained by adding 3 to the corresponding BCD code:

0	0011	5	1000
1	0100	6	1001
2	0101	7	1010
3	0110	8	1011
4	0111	9	1100

Excess-3 code enables simpler BCD arithmetic hardware. For example, the 9s complement of any digit can be obtained simply by taking the 1s complement of the corresponding code.

Consider the addition of two decimal digits. In BCD, when the SUM exceeds 9, a 6 is added to bring the resulting digit into the range 0–9. A CARRY is generated in this process that is used in the addition of digits in the next stage.

In Excess-3 it is not necessary to check to see if the SUM exceeds 9 or not since the correction can be done just by checking to see if the addition has resulted in a CARRY or not. The correction is to either add 3 or subtract 3 from the result, as shown by Example A.40.

Example A.40

Decimal	BCD	Excess-3
3	0011	0110
<u>+ 2</u>	<u>0010</u>	<u>0101</u>
5	0101	0 1011
	SUM < 9: No correction needed.	CARRY = 0 ∴ Subtract <u>0011</u> 3 1000
5	0101	1000
<u>+ 6</u>	<u>0010</u>	<u>1001</u>
11	1011	10011
	SUM > 9: ∴ Add 6: <u>0110</u> 1,0001	CARRY = 1 ∴ Add 3: <u>0011</u> 1,0100

A.8.3 Two-out-of-five Code

This code uses 5 bits to represent each decimal digit. As the name implies, 2 bits out of the 5 bits are 1s in each code word:

0	11000	5	01010
1	00011	6	01100
2	00101	7	10001
3	00110	8	10010
4	01001	9	10100

With 5 bits, 32 code words are possible. Only 10 of them are used, but the code provides an automatic error detection capability since the number of 1s in the code word must always be two.

This code uses a simple “parity” scheme. Since the number of 1s is always even, it is called “even parity.” In general, one or more parity bits are included into the code word to facilitate error detection and correction. Even and odd parity schemes are the simplest ones. Several other schemes are used to generate parity bits where a higher degree of error detection and correction is needed.

A.8.4 Alphanumeric Codes

When alphabetic characters, numeric digits, and special characters are used in representing the information to be processed by the digital system, an alphanumeric code is used. Two popular alphanumeric codes are Extended BCD Interchange Code (EBCDIC) and American Standard Code for Information Interchange (ASCII). These codes are shown in Table A.7.

Table A.7 Alphanumeric Codes

Character	EBCDIC code	ASCII code
blank	0100 0000	0010 0000
.	0100 1011	0010 1110
(	0100 1101	0010 1000
+	0100 1110	0010 1011
\$	0101 1011	0010 0100
*	0101 1100	0010 1010
)	0101 1101	0010 1001
—	0110 0000	0010 1101

Table A.7 Alphanumeric Codes (Continued)

Character	EBCDIC code	ASCII code
/	0110 0001	0010 1111
,	0110 1011	0010 1100
'	0111 1101	0010 0111
=	0111 1110	0011 1101
0	1111 0000	0011 0000
1	1111 0001	0011 0001
2	1111 0010	0011 0010
3	1111 0011	0011 0011
4	1111 0100	0011 0100
5	1111 0101	0011 0101
6	1111 0110	0011 0110
7	1111 0111	0011 0111
8	1111 1000	0011 1000
9	1111 1001	0011 1001
A	1100 0001	0100 0001
B	1100 0010	0100 0010
C	1100 0011	0100 0011
D	1100 0100	0100 0100
E	1100 0101	0100 0101
F	1100 0110	0100 0110
G	1100 0111	0100 0111
H	1100 1000	0100 1000
I	1100 1001	0100 1001
J	1101 0001	0100 1010
K	1101 0010	0100 1011
L	1101 0011	0100 1100
M	1101 0100	0100 1101
N	1101 0101	0100 1110
O	1101 0110	0100 1111
P	1101 0111	0101 0000
Q	1101 1000	0101 0001
R	1101 1001	0101 0010
S	1110 0010	0101 0011
T	1110 0011	0101 0100
U	1110 0100	0101 0101
V	1110 0101	0101 0110
W	1110 0110	0101 0111
X	1110 0111	0101 1000
Y	1110 1000	0101 1001
Z	1110 1001	0101 1010