Binary System

Information representation

- Digital form
  - A set restricted to a finite number or sequence of elements/digits; thus the information is discrete
  - E.g. a digital watch, which expresses time in a numerical form using digits
  - Limits the precision of the information to the number of digits
- Analog form
  - A continuum is used to denote the information
  - E.g. a conventional watch using hands and the angle between the hands to show the time, voltages, current, etc.
- Digital systems deal with digitized information
  - cheaper, reliable and greater versatility

Why binary?

- Digital computers/systems deal with discrete elements of information, which are themselves represented physically as signals.
    - Signals e.g., voltage and current are themselves analog or continuous quantities
    - An analog to digital (A-to-D) conversion is hence required
- Hence digital computers only (in fact can only) manipulate numbers!
- So what does a digital computer do?
    - Receives numbers called data
    - Performs operations on these numbers
    - Forms new numbers
        - The desired operations to be performed are also given to the computer in the form of numbers called instructions

Why binary?

- Since numbers are stored and manipulated – a number system, which is easy to represent electronically is necessary
- Binary number system or a coded binary system is used
  - Highly reliable electronic devices with 2 stable states are easily fabricated
  - Signals have 2 discrete values (hence the term binary)
  - A binary digit – bit has 2 values 0 and 1

Binary system

- The traditional decimal number system
  - Base (or radix) 10 uses ten digits (0,1,…9), each multiplied by a power of 10 depending on its position
  - E.g. $7392 = 7 \times 10^3 + 3 \times 10^2 + 9 \times 10^1 + 2 \times 10^0$

- The binary number system
  - Base 2 uses two digits (0 and 1), each multiplied by a power of 2 depending on its position
  - $1011_2 = 1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0 = 11$

- The radix point distinguishes positive powers of 10 (or 2) from negative powers of 10 (or 2)
  - $11010.11_2 = 1 \times 2^4 + 1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 0 \times 2^0 + 1 \times 2^{-1} + 1 \times 2^{-2} = 26.75$

Binary system

- Some commonly used terms in computers: binary prefix

  <u>Term</u> <u>Binary</u> <u>Decimal</u>

  - K(Kilo) $= 2^{10} = 1024$ $\cong 10^3$ thousand
  - M(Mega) $= 2^{20} = 1048576$ $\cong 10^6$ million
  - G(Giga) $= 2^{30} = 1073741824$ $\cong 10^9$ billion
  - T(Tera) $= 2^{40} = 1.099\text{x}10^{12}$ $\cong 10^{12}$ trillion

k	kilo	2^{10}
M	mega	2^{20}
G	giga	2^{30}
T	tera	2^{40}
P	peta	2^{50}
E	exa	2^{60}
Z	zetta	2^{70}
Y	yotta	2^{80}

Thus 4K $= 2^2\text{x}2^{10} = 2^{12} = 4096$

- Computer capacity is measured in *bytes*, which is equal to 8 *bits* of information (e.g. 11111111 or 10101010 or 11110000, …)
- Thus 4Kb $= 2^2\text{x}2^{10} = 2^{12} = 4096$ bits
- While 4KB $= 2^2\text{x}2^{10}\text{x}2^3 = 2^{15} = 32768$ bits

Binary system

- Some 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

Base-*r* system

- In general, a number expressed in base-*r* system
  - Has coefficients multiplied by power of *r*

$$a_n \cdot r^n + a_{n-1} \cdot r^{n-1} + \cdots + a_2 \cdot r^2 + a_1 \cdot r^1 + a_0 \cdot r^0 + a_{-1} \cdot r^{-1} + a_{-2} \cdot r^{-2} + \cdots + a_{-m} \cdot r^{-m}$$

  - Coefficients a_j can range from 0 to *r*-1; we enclose the coefficients in parentheses and write a subscript equal to the base ; $(a_n a_{n-1} \cdots a_1 a_0 . a_{-1} \cdots a_{-m})_r$
  - Some examples are

$$(4021.2)_5 = 4 \times 5^3 + 0 \times 5^2 + 2 \times 5^1 + 1 \times 5^0 + 2 \times 5^{-1} = (511.4)_{10}$$

$$(127.4)_8 = 1 \times 8^2 + 2 \times 8^1 + 7 \times 8^0 + 4 \times 8^{-1} = (87.5)_{10}$$

  - When the base is greater than 10, the letters of the alphabet are used to supplement the 10 decimal digits

$$(B65F)_{16} = 11 \times 16^3 + 6 \times 16^2 + 5 \times 16^1 + 15 \times 16^0 = (46687)_{10}$$

Base-*r* system

- Some number systems

Base	Number system	Digit symbols
2	Binary	0,1
3	Ternary	0,1,2
4	Quaternary	0,1,2,3
5	Quinary	0,1,2,3,4
8	Octal	0,1,2,3,4,5…7
10	Decimal	0,1,2,3,4…9
12	Duodecimal	0,1,2…9,A,B
16	Hexadecimal	0,1,2…9,A,B,C,D,E,F

Converting decimal to base-*r*

- Converting a number from base *r* to decimal - Expand the number in a power series and add all the terms
  - $(1011)_2 = 1x2^3+0x2^2+1x2^1+1x2^0 = (11)_{10}$
- Converting from decimal to base *r*
  - $(41)_{10} \rightarrow (x)_2$

	Integer Quotient		Remainder	Coefficient
41/2 =	20	+	$\frac{1}{2}$	$a_0 = 1$
20/2 =	10	+	0	$a_1 = 0$
10/2 =	5	+	0	$a_2 = 0$
5/2 =	2	+	$\frac{1}{2}$	$a_3 = 1$
2/2 =	1	+	0	$a_4 = 0$
1/2 =	0	+	$\frac{1}{2}$	$a_5 = 1$

Integer	Remainder	
41		
20	1	
10	0	
5	0	
2	1	
1	0	
0	1	101001 = answer

$$(41)_{10} = (a_5a_4a_3a_2a_1a_0)_2 = (101001)_2$$

Converting decimal to base-r

- $(153)_{10} \rightarrow (x)_8$

153	
19	1
2	3
0	$2 = (231)_8$

- What if the number has a fraction?
  - continue multiplying till the fraction becomes 0 or you have sufficient accuracy
  - Convert (0.6875)

	Integer		Fraction	Coefficient
$0.6875 \times 2 =$	1	+	0.3750	$a_{-1} = 1$
$0.3750 \times 2 =$	0	+	0.7500	$a_{-2} = 0$
$0.7500 \times 2 =$	1	+	0.5000	$a_{-3} = 1$
$0.5000 \times 2 =$	1	+	0.0000	$a_{-4} = 1$

$$(0.6875)_{10} = (0.a_{-1}a_{-2}a_{-3}a_{-4})_2 = (0.1011)_2$$

Converting decimal to base-*r*

- $(0.513)_{10} \rightarrow (x)_8$

$$0.513 \times 8 = 4.104$$
$$0.104 \times 8 = 0.832$$
$$0.832 \times 8 = 6.656$$
$$0.656 \times 8 = 5.248$$
$$0.248 \times 8 = 1.984$$
$$0.984 \times 8 = 7.872$$

$(0.153)_{10} = (0.406517 \ldots)_8$

- The conversion of decimal numbers with both integer and fraction part is done by converting the integer and fraction separately and then combining the two answers.

Converting decimal to base-r

- Exercises
  - Convert $(103.732)_{10} \rightarrow (x)_2, (x)_3, (x)_8$ and $(x)_{16}$
  - Convert $(41.6875)_{10} \rightarrow (x)_2$
  - Convert $(153.513)_{10} \rightarrow (x)_8$

$$41.6875 = 41 + 0.6875$$

2	41	
2	20	1
2	10	0
2	5	0
2	2	1
2	1	0
	0	1

$0.6875 \times 2 = 1.375$

$0.375 \times 2 = 0.75$

$0.75 \times 2 = 1.5$

$0.5 \times 2 = 1.0$

$$101001.1011_2$$

Converting Binary to Octal

- When one base is an integer power of the other e.g., from base-2 (binary) to base-8 (octal) $2^3=8$, and base-16 (hexadecimal) $2^4=16$
- Converting from binary to octal
  - Starting from the binary point
  - Working both left and right
    - Group bits into threes
    - Add leading or trailing zeros if necessary
    - Convert each group of threes into their octal equivalent
- Example:
  - Convert $(11111101.0011)_2 \rightarrow (x)_8$
  - Group as: $\underline{011}\ \ \underline{111}\ \ \underline{101}\ \ .\ \ \underline{001}\ \ \underline{100}$

    $3\ \ \ 7\ \ \ 5\ \ .\ \ 1\ \ \ 4$

    Answer = $(375.14)_8$
  - The reverse procedure gives the octal to binary conversion

Converting binary to hexadecimal

- Converting from binary to hexadecimal is similar except that we now group into fours
  - Example: $(10110001101011.11110010)_2 \rightarrow (x)_{16}$

    <u>*00*10</u> <u>1100</u> <u>0110</u> <u>1011</u> . <u>1111</u> <u>0010</u>

    2 C 6 B F 2

  Answer: $(2C6B.F2)_{16}$
  - The reverse procedure gives the hexadecimal to binary conversion
- Why octal & hexadecimal?
  - Binary use a lot of bits to represent a number; E.g. the decimal 4095 requires only 4 digits but in binary – 111111111111, 12 bits/digits are needed
  - The octal & hexadecimal reduces the number of digits; E.g. 4095$\rightarrow(7777)_8$ (4 digits), $(FFF)_{16}$ (3 digits) while retain the binary system; Simple and efficient

Binary arithmetic

- Binary arithmetic
  0+0 = 1, 1+0 = 0+1 = 1, 1+1 = 10,
  0-0 = 0, 0-1 = 1, 1-0 = 1, 1-1 = 0
  0x0 = 0, 1x0 = 0x1 = 0, 1x1 = 1
- Example Addition
  1 1111 → carries
  101101
  +100111
  1010100
- Example Subtract, using borrow
  101101
  - 100111
  000110
- Example Multiplication
  1011
  x 11
  1011
  1011
  100001

Complements

- complements are used to simplify the subtraction operation and for logical manipulation
- Diminished radix complement, **(*r*-1)’s complement**
  - Given a number N in base r having n digits, the (r-1)’s complement of N is defined as $(r^n - 1) - N$
  - For decimal number, the 9’s complement is obtained by subtracting each digit from 9

    The 9’s complement of 546700 is 999999 – 546700 = 453299

    The 9’s complement of 012398 is 999999 – 012398 = 987601
  - For binary numbers, the 1’s complement is obtained by subtracting each digit from 1 (i.e. changing 1’s to 0’s and 0’s to 1’s)

    The 1’s complement of 1011000 is 0100111

    The 1’s complement of 0101101 is 1010010

Complements

- Radix complement, ***r*'s complement**
  - the *r*'s complement of an *n*-digit number *N* is defined as r^n - *N* for $N \neq 0$ and as 0 for *N* = 0
  - The *r*'s complement is obtained by adding 1 to the (*r*-1)'s complement, since $r^n - N = [(r^n - 1) - N] + 1$
  - 10's complement can be formed by leaving all least significant 0's unchanged, subtracting the first nonzero least significant digit from 10, and subtracting all higher significant digits from 9

    The 10's complement of 012398 is 987602

    The 10's complement of 246700 is 753300
  - 2's complement can be formed by leaving all least significant 0's and the first 1 unchanged and replacing 1's with 0's and 0's with 1's in all other higher significant digits

    The 2's complement of 1101100 is 0010100

    The 2's complement of 0110111 is 1001001

Complements

- If the number *N* contains a radix point, the point should be removed temporarily in order to form the *r*'s or (*r*-1)'s complement, then the radix point is restored to the complemented number in the same relative position
- The complement of the complement number restores the number to its original value

Subtraction using complements

- $M - N$ in base r
  - Add M to the r's complement of N ; $M + (r^n - N) = M - N + r^n$
  - If $M \geq N$, there is an end carry, discard it result $M - N$
  - If $M < N$, no end carry, result is r's complement of $N - M$; take the r's complement and place a ‘minus’ sign in front

Subtraction using complements

- Using 10's comp, subtract 72532 – 3250 ($M \geq N$)

$$
\begin{aligned}
M &= 72532 \\
\text{10's complement of } N &= +\underline{96750} \\
\text{Sum} &= 169282 \\
\text{Discard end carry } 10^5 &= -\underline{100000} \\
\textit{Answer} &= 69282
\end{aligned}
$$

- Using 10's comp, subtract 3250 - 72532 ($M < N$)

$$
\begin{aligned}
M &= 03250 \\
\text{10's complement of } N &= +\underline{27468} \\
\text{Sum} &= 30718
\end{aligned}
$$

  - No end carry; the answer is
    –(10's comp of 30718) = -69282

Subtraction using complements

- Given the two binary numbers X = 1010100 and Y = 1000011, perform the subtraction (a) X – Y and (b) Y – X by using 2's complement
  - (a)

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

  - (b)

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

No end carry, the answer is Y-X = -(2's comp of 1101111)

= -0010001

Subtraction using complements

- Repeat the previous example using 1's complement
  - (a)

$$\begin{array}{rr} X = & 1010100 \\ \text{1's complement of } Y = & +\ \underline{0111100} \\ \text{Sum} = & 10010000 \\ \text{End-around carry} = & +\ \underline{\qquad\quad 1} \\ \textit{Answer: } X - Y = & 0010001 \end{array}$$

  - (b)

$$\begin{array}{rr} Y = & 1000011 \\ \text{1's complement of } X = & +\ \underline{0101011} \\ \text{Sum} = & 1101110 \end{array}$$

No end carry, the answer is Y-X = -(1's comp of 1101111)

= -0010001

Signed number

- How do we deal with Positive and Negative numbers?

- Signed magnitude convention
  - Represent the sign with the leftmost bit
  - A '0' to indicate a +ve number and a '1' for a –ve number (remember computers represent *everything* using 0 and 1 bits)
  - Thus 11001 → 25 if the binary number is unsigned, else 11001 → – 9 (01001 → +9), if we assume the number is a signed number
  - Need to know the representation in advance
    - Above is known as **Signed-magnitude representation**
    - Complement the sign bit and we get the same magnitude but with the opposite sign

Signed number

- The *signed-complement* system is more convenient and used for –ve numbers
  - -ve numbers are indicated by their complement
  - Negate a number by taking the complement
  - +ve numbers start with a 0 the complement will always start with 1 (-ve)
    - Can use 1's or 2's comp
  - Represent 9 in binary with 8 bits
    - +9 – 00001001 → only 1 way to represent
    - -9 – 10001001 → signed-magnitude
      - 11110110 → 1's comp
      - 11110111 → 2's comp
    - 3 ways to represent a –ve number
  - How do you get the complements in each case?

Signed number

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	—	—

Signed number

- Adding 8 bit 2's complement numbers
  - -ve numbers in signed-complement form (2's comp. Most commonly used)
  - Simple requires only addition no sign comparison etc.
  - All carries are discarded
  - If the answer is +ve DONE
  - If the answer is –ve, it is in 2's comp form
  - To place in a more familiar form
    - Take 2's comp of the –ve answer and place a '—' in front of it, e.g. –7 above is in the 2's comp form 11111001. Take the 2's comp of this and you get 00000111, put '—' in front

+6	00000110	-6	11111010
+13	00001101	+13	00001101
+19	00010011	+ 7	00000111
+6	00000110	-6	11111010
-13	11110011	-13	11110011
-7	11111001	-19	11101101

Signed number

- Subtraction M – N
    1. Take 2’s comp of N
    2. Add to M
    3. Discard any carry out of the sign bit
    4. If the answer is –ve, it is in 2’s comp form
    5. Then follow same steps as before

Signed number

Example: (-6) – (-13)
+6 → 00000110 and -6 → 11111010
+13 → 00001101; -13 →11110011 and 2's comp → 00001101
=> -6+(+13) 11111010
<u>00001101</u>
~~1~~00000111 → +7
Example: –13-(-6)
⇒-13+(+6) 11110011
<u>00000110</u>
11111001 (-ve so to put in familiar form have to take 2's comp of result and place a '-' sign in front of it.
→-(00000111) → -7

- Since now we *always* end up adding, the *same* hardware can be used to do both addition and subtraction arithmetic ; The user/program must interpret the result correctly

Binary codes

- Binary codes
  - An n-bit binary code is a group of n bits that can assume a max of 2^n distinct combinations of 0 and 1
    - What if we had 3 combinations say 0, 1& 1/2, how many distinct combinations could we then have? What if we had 4 possible combinations?
  - A 3-bit binary code can assume 8 distinct combinations
    - Each combination can be assigned a number determined from the binary count from 0 to $2^n - 1$, similarly for a 4-bit binary code

Binary codes

- Binary Coded Decimal code (BCD)
  - Assign a binary code of 4 bits to each decimal symbol 0 to 9
  - The remaining 6 are unused
  - A number with k decimal digits requires 4k bits if BCD is used

    $(185)_{10}$

    $= (0001\ 1000\ 0101)_{BCD}$

    $= (10111001)_2$

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

Binary codes

- BCD addition
  - Similar to binary addition as long as BCD digit sum is less than or equal to 1001
  - When sum is greater than 1001, add 6=0110 to correct it
  - e.g. 4+5 = 0100 + 0101 = 1001 (+9) OK as answer is <= 1001
  - e.g. 4+8 = 0100+1000 = 1100 (>1001), so add 0110 to this; 1100+0110 = 10010 =
  - 0001 0010 = 12
  - Exercise: perform the following BCD additions 188+675, 9099+2345, 23+89

Binary codes

- Other Decimal codes
  - 2421, Excess-3 and 84-2-1; BCD is 8421
  - These are weighted codes (including BCD), each bit position is assigned a weighting factor
  - Some digits can be coded in 2 possible ways in 2421 e.g. 4 – 0100 or 1010
  - 2421 and Excess-3 are self complementing
    - 9's complement of a decimal number can be obtained directly by changing 1 to 0 and 0 to 1
  - Gray code:
    - Only 1 bit in the code changes when going from one number to next
    - Reduces electronic error in counting, which can happen when too many bits need to be changed at the same time e.g. 7 to 8 in binary →0111 to 1000, **all** bits change! In gray code 0100 to 1100, **only** a 1 bit change

Self-complementing

weighted code

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

Binary codes

- ASCII code
  - American Standard Code for Information Interchange
  - Alphanumeric – numbers, characters and symbols total of 128 codes
  - Require at least 7 bits, as $2^7 = 128$, most computers use 8 bits (a byte) to store an ASCII character
    - 7 bit ASCII characters are stored in an 8 bit byte
  - Byte is the most common unit of memory that is manipulated by the computer

American Standard Code for Information Interchange (ASCII)

	$B_7B_6B_5$							
$B_4B_3B_2B_1$	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

Error detecting codes

- Error Detecting Codes
  - Errors can occur while reading and writing data or any sort of information, especially when we send information over a communication medium e.g. copper wires/ telephone lines, wireless communications
  - To detect errors – add redundancy (i.e. additional information) along with the data/message being sent
  - E.g. Add an extra Parity bit to the ASCII character to indicate its parity
    - Even parity – add extra bit such that the total number of bits is even
    - Odd parity – add extra bit such that the total number of bits is odd

	Even parity	Odd parity
ASCII A = 1000001	***0***1000001	***1***1000001

Error detecting codes

- Both the sender (transmitter Tx) and receiver (Rx) agree upon using a certain type of parity
    1. Generate parity for each character at Tx
    2. Rx checks parity of each character
    3. If parity does not match then
        - ERROR – At least one bit has changed
        - Tx is informed and asked to resend the message

    - This method detects 1, 3, 5, or any odd number of errors
    - What happens if there are 2, 4, or even number of errors?
    - Remember a 7 bit ASCII character is stored in an 8 bit byte – the extra bit is usually used for parity
        - To check if each ASCII character is read/written/transferred/stored correctly