Computer Architecture I: Digital Design

School of Computer Science
University of Windsor

Binary Codes

Binary-Coded Decimal (BCD)

- A number with k decimal digits will require $4k$ bits in BCD.

  For example,

  Decimal 396 is represented in BCD with 12 bits as 0011 1001 0110, with each group of 4 bits representing one decimal digit.
- A decimal number in BCD is the same as its equivalent binary number only when the number is between 0 and 9.

Binary Codes

Binary-Coded Decimal (BCD)

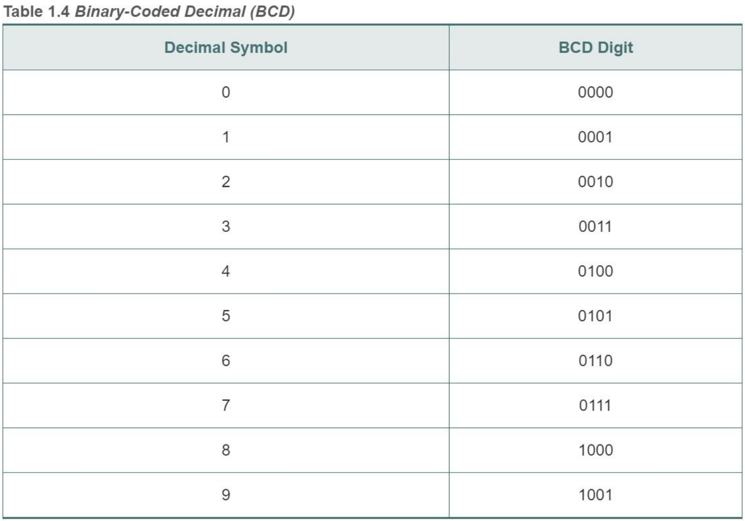


Table 1.4 ***Binary-Coded Decimal (BCD)***

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

Binary-Coded Decimal (BCD)

- A BCD number greater than 10 looks different from its equivalent binary number, even though both contain 1's and 0's.
- Moreover, the binary combinations 1010 through 1111 are not used and have no meaning in BCD.

  Consider decimal 185 and its corresponding value in BCD and binary:

  $(185)_{10} = (000110000101)_{BCD} = (10111001)_2$

Binary Codes

Binary-Coded Decimal (BCD): Practice Exercise

1. Find the BCD representation of 84_{10}.

Binary Codes

BCD Addition

$$
\begin{array}{rrrrrr}
4 & 0100 & 4 & 0100 & 8 & 1000 \\
+\frac{5}{9} & \frac{+0101}{1001} & \frac{+8}{12} & \frac{+1000}{1100} & \frac{+9}{17} & \frac{1001}{10001} \\
 & & & \frac{+0110}{10010} & & \frac{+0110}{10111}
\end{array}
$$

Binary Codes

BCD Addition: Practice Exercise

1. Find the BCD sum of 184 and 576.

Binary Codes

Other Decimal Codes

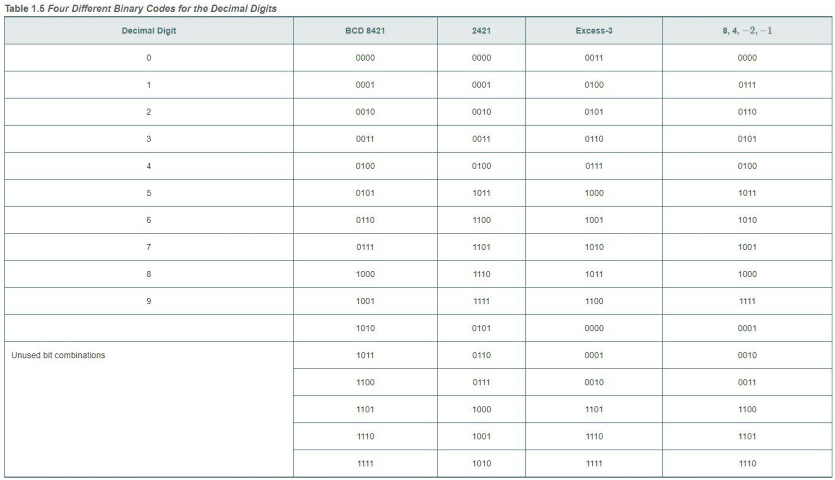


Table 1.5 *Four Different Binary Codes for the Decimal Digits*

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
	1010	0101	0000	0001
Unused bit combinations	1011	0110	0001	0010
	1100	0111	0010	0011
	1101	1000	1101	1100
	1110	1001	1110	1101
	1111	1010	1111	1110

Binary Codes

Other Decimal Codes

- The 2421 and the excess-3 codes are examples of self-complementing codes.
- Such codes have the property that the 9's complement of a decimal number is obtained directly by changing 1's to 0's and 0's to 1's (i.e., by complementing each bit in the pattern).

  For example,

  decimal 395 is represented in the excess-3 code as 0110 1100 1000. The 9's complement of 604 is represented as 1001 0011 0111, which is obtained simply by complementing each bit of the code (as with the 1's complement of binary numbers).

Binary Codes

Other Decimal Codes

- The excess-3 code has been used in some older computers because of its self-complementing property.
- Excess-3 is an unweighted code in which each coded combination is obtained from the corresponding binary value plus 3.
- Note that the BCD code is not self-complementing.

Binary Codes

Other Decimal Codes

- The 8, 4, -2, -1 code is an example of assigning both positive and negative weights to a decimal code.
- In this case, the bit combination 0110 is interpreted as decimal 2 and is calculated from $8 \times 0 + 4 \times 1 + (-2) \times 1 + (-1) \times 0 = 2$.

Binary Codes

Gray Code

- It is sometimes convenient to use the Gray code shown in Table 1.6 to represent digital data that have been converted from analog data.
- The **advantage** of the Gray code over the straight binary number sequence is that only one bit in the code group changes in going from one number to the next.

  For example,

  in going from 7 to 8, the Gray code changes from 0100 to 1100. Only the first bit changes, from 0 to 1; the other three bits remain the same. By contrast, with binary numbers the change from 7 to 8 will be from 0111 to 1000, which causes all four bits to change values.

Binary Codes

Gray Code

- The Gray code is used in **applications** in which the normal sequence of binary numbers generated by the hardware may produce an error or ambiguity during the transition from one number to the next.
- If binary numbers are used, a change,

  For example,

  from 0111 to 1000 may produce an intermediate erroneous number 1001 if the value of the rightmost bit takes longer to change than do the values of the other three bits. This could have serious consequences for the machine using the information. The Gray code eliminates this problem, since only one bit changes its value during any transition between two numbers.

Binary Codes

Gray Code

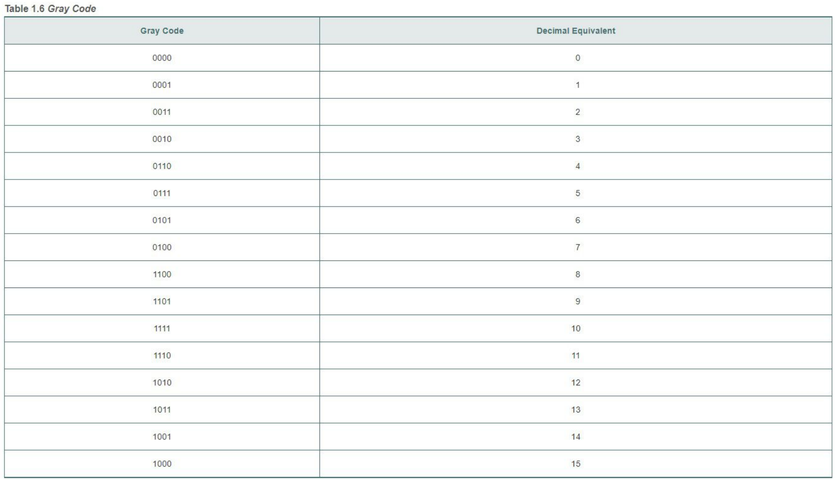


Table 1.6 *Gray Code*

Gray Code	Decimal Equivalent
0000	0
0001	1
0011	2
0010	3
0110	4
0111	5
0101	6
0100	7
1100	8
1101	9
1111	10
1110	11
1010	12
1011	13
1001	14
1000	15

Binary Codes

ASCII Character Code

- The standard binary code for the alphanumeric characters is the **American Standard Code for Information Interchange (ASCII)**, which uses seven bits to code 128 characters, as shown in Table 1.7.
- The seven bits of the code are designated by b_1 through b_7, with b_7 the most significant bit.
- The letter A,

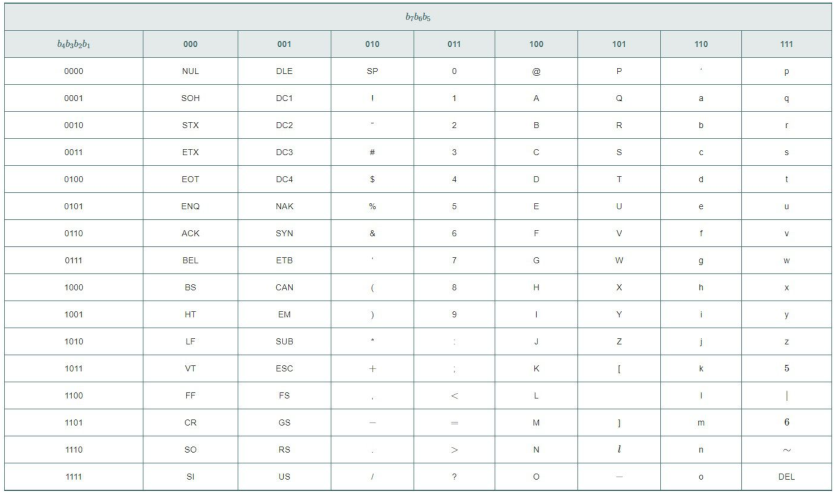
  for example,

  is represented in ASCII as 1000001 (column 100, row 0001).

Binary Codes

ASCII Character Code

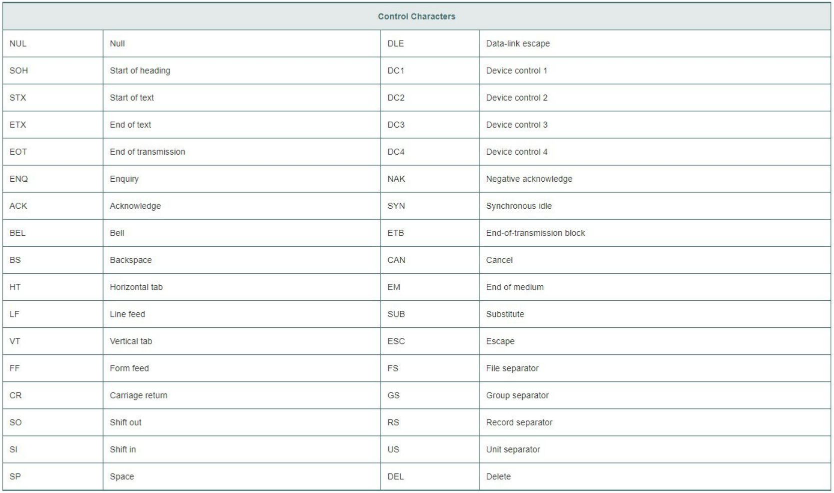
Table 1.7 *American Standard Code for Information Interchange (ASCII)*



	$b_7b_6b_5$							
$b_4b_3b_2b_1$	000	001	010	011	100	101	110	111
0000	NUL	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	5
1100	FF	FS	,	<	L		l	|
1101	CR	GS	−	=	M	]	m	6
1110	SO	RS	.	>	N	*l*	n	~
1111	SI	US	/	?	O	—	o	DEL

Binary Codes

ASCII Character Code

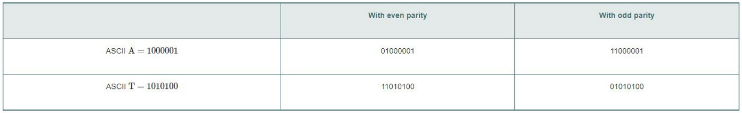


Control Characters			
NUL	Null	DLE	Data-link escape
SOH	Start of heading	DC1	Device control 1
STX	Start of text	DC2	Device control 2
ETX	End of text	DC3	Device control 3
EOT	End of transmission	DC4	Device control 4
ENQ	Enquiry	NAK	Negative acknowledge
ACK	Acknowledge	SYN	Synchronous idle
BEL	Bell	ETB	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

Binary Codes

Error-Detecting Code

- To detect errors in data communication and processing, an eighth bit is sometimes added to the ASCII character to indicate its parity.
- A **parity bit** is an extra bit included with a message to make the total number of 1's either even or odd.
- Consider the following two characters and their even and odd parity:



	With even parity	With odd parity
ASCII $A = 1000001$	01000001	11000001
ASCII $T = 1010100$	11010100	01010100

Binary Codes

Error-Detecting Code

- In each case, we insert an extra bit in the leftmost position of the code to produce an even number of 1's in the character for even parity or an odd number of 1's in the character for odd parity.
- In general, one or the other parity is adopted, with even parity being more common.

Binary Logic

Three Basic Logical Operations

- Binary logic deals with variables that take on two discrete values and with operations that assume logical meaning.
- The two values the variables assume may be called by different names (true and false, yes and no, etc.), but for our purpose, it is convenient to think in terms of bits and assign the values 1 and 0.
- The binary logic introduced in this section is equivalent to an algebra called Boolean algebra. The formal presentation of Boolean algebra is covered in more detail in Chapter 2.

Binary Logic

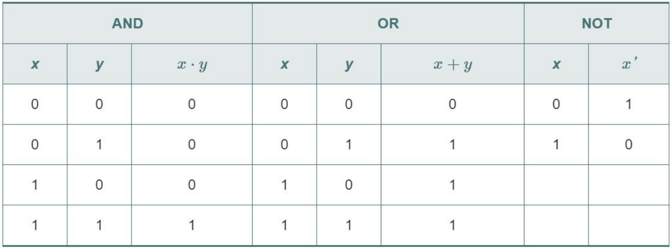
Three Basic Logical Operations

- Binary logic consists of binary variables and a set of logical operations.
- The variables are designated by letters of the alphabet, such as A, B, C, x, y, z, etc., with each variable having two and only two distinct possible values: 1 and 0.
- There are three basic logical operations: AND, OR, and NOT. Each operation produces a binary result, denoted by z.

Binary Logic

Three Basic Logical Operations

Table 1.8 ***Truth Tables of Logical Operations***



AND			OR			NOT	
x	***y***	$x \cdot y$	***x***	***y***	$x + y$	***x***	x'
0	0	0	0	0	0	0	1
0	1	0	0	1	1	1	0
1	0	0	1	0	1		
1	1	1	1	1	1		

Binary Logic

Three Basic Logical Operations

- AND and OR are the same as those used for multiplication and addition.
- However, binary logic should not be confused with binary arithmetic.
- One should realize that an arithmetic variable designates a number that may consist of many digits. A logic variable is always either 1 or 0.

  For example,

  in binary arithmetic, we have $1 + 1 = 10$ (read “one plus one is equal to 2"), whereas in binary logic, we have $1 + 1 = 1$ (read “one OR one is equal to one").

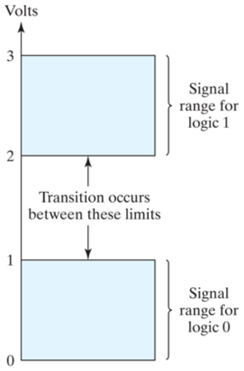
Binary Logic

Logic Gates

- Logic gates are electronic circuits that operate on one or more input signals to produce an output signal.
- Electrical signals such as voltages or currents exist as analog signals having values over a given continuous range, say, 0 to 3 V, but in a digital system these voltages are interpreted to be either of two recognizable values, 0 or 1.

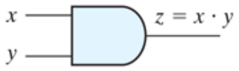
Binary Logic

Logic Gates

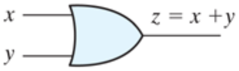


Binary Logic

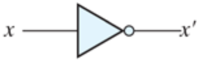
Logic Gates



(a) Two-input AND gate



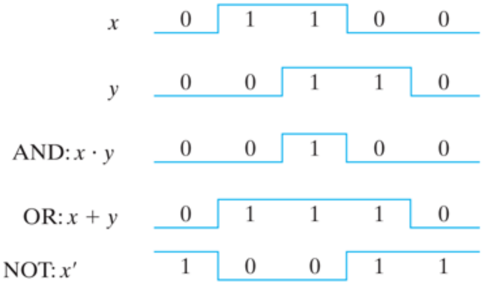
(b) Two-input OR gate



(c) NOT gate or inverter

Binary Logic

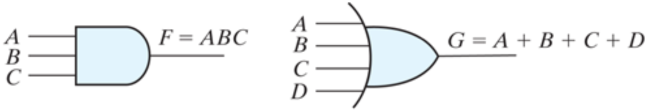
Logic Gates



Binary Logic

Logic Gates

- AND and OR gates may have more than two inputs. An AND gate with three inputs and an OR gate with four inputs are shown in Fig below.



(a) Three-input AND gate (b) Four-input OR gate

Binary Logic

Logic Gates: Practice Exercise

1. Write down the truth table of an AND gate for two inputs.