Lecture 5 – Binary Codes

Dr. Aftab M. Hussain,

Assistant Professor, PATRIoT Lab, CVEST

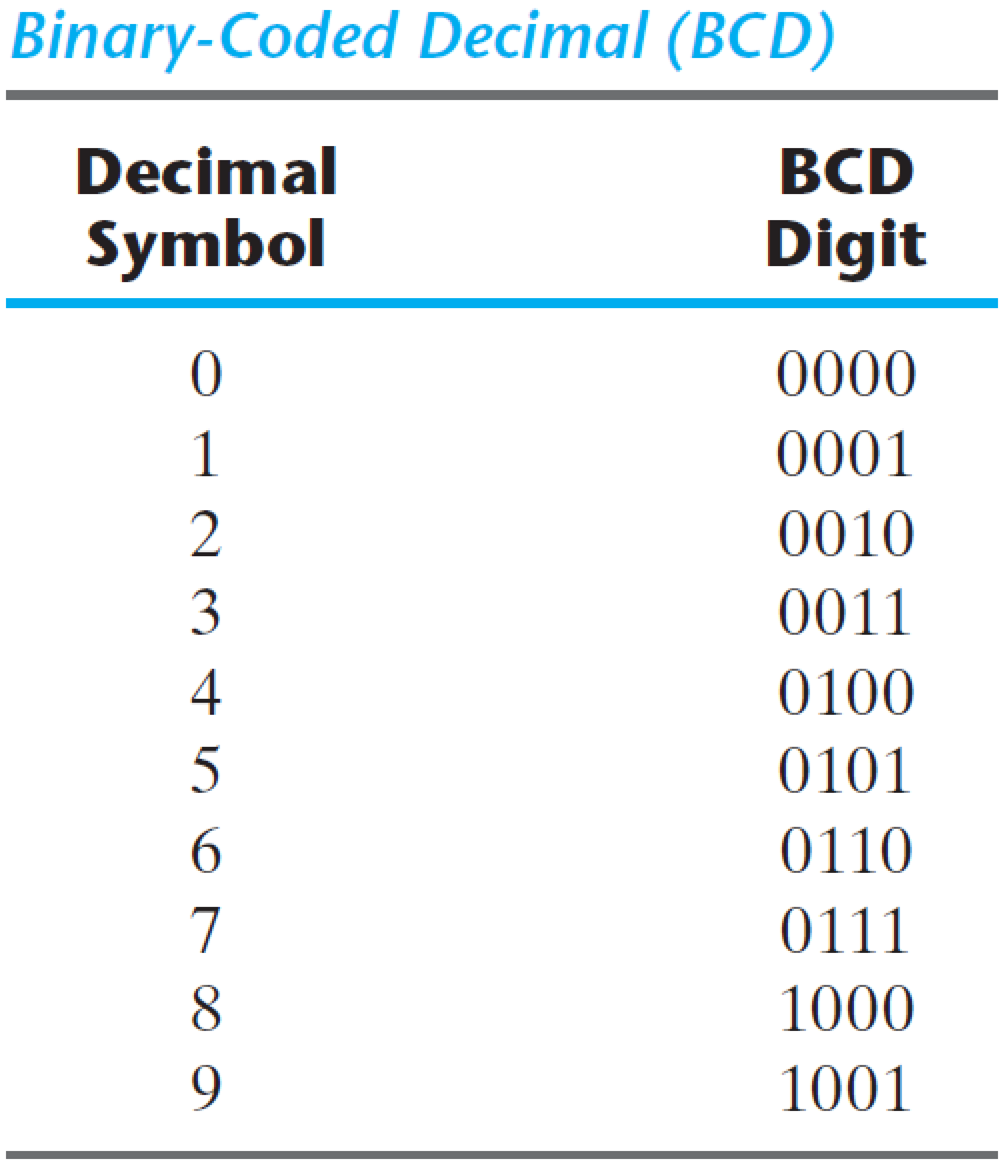
Chapter 2

Binary codes

- Any discrete element of information that is distinct among a group of quantities can be represented with a binary code (i.e., a pattern of 0’s and 1’s)
- The codes must be in binary because, in today’s technology, only circuits that represent and manipulate patterns of 0’s and 1’s can be manufactured economically for use in computers
- However, it must be realized that binary codes merely change the symbols, not the meaning of the elements of information that they represent
- If we inspect the bits of a computer at random, we will find that most of the time they represent some type of coded information rather than binary numbers
- An *n*-bit binary code can represent up to 2^n distinct elements of the set that is being coded

Binary Coded Decimal (BCD)

- The code most commonly used for the decimal digits is the straight binary assignment and is called *binary-coded decimal* (BCD)
- A binary code will have some unassigned bit combinations if the number of elements in the set is not a multiple power of 2
- The 10 decimal digits form such a set
- A binary code that distinguishes among 10 elements must contain at least four bits, but 6 out of the 16 possible combinations remain unassigned

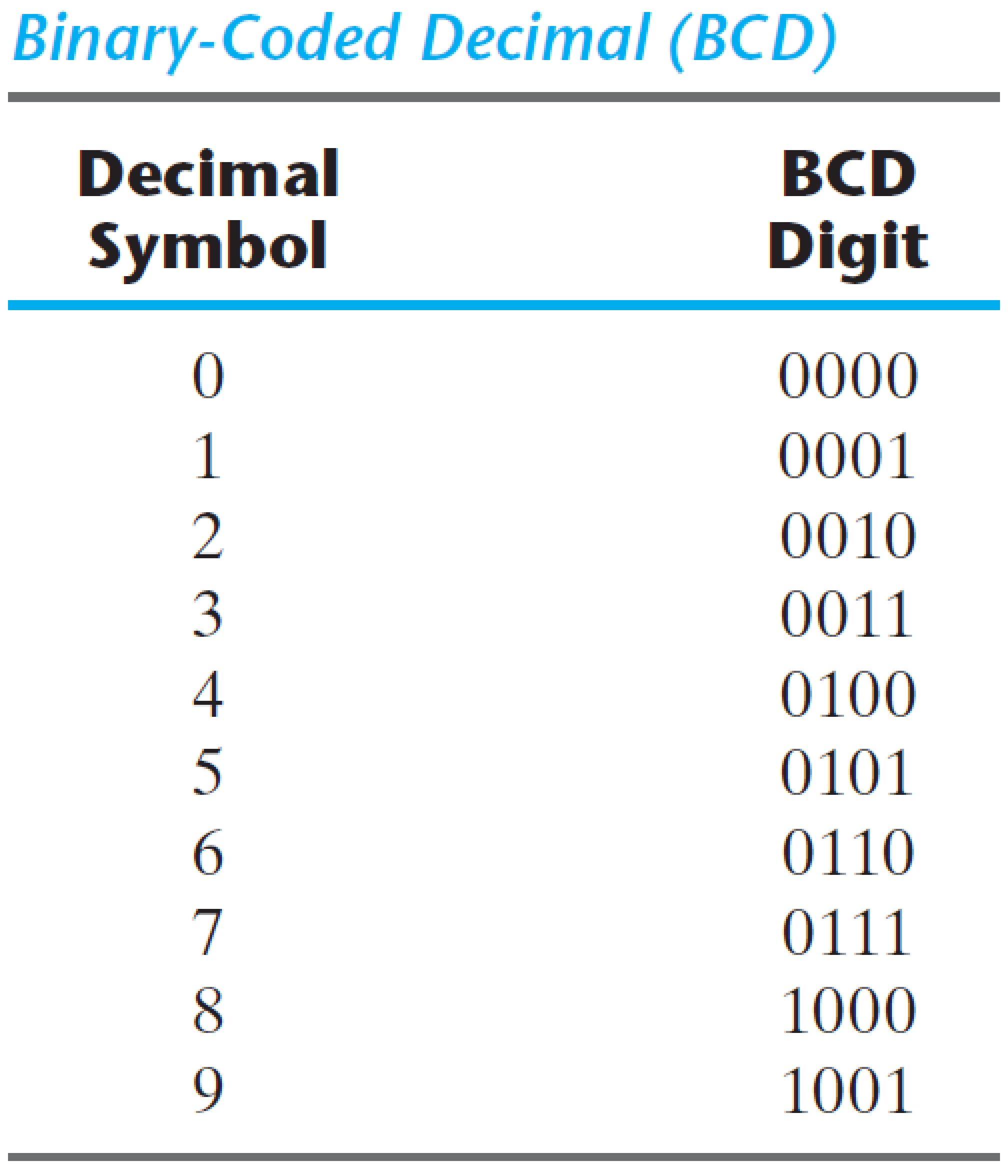


Binary-Coded Decimal (BCD)

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 Coded Decimal (BCD)

- A number with *k* decimal digits will require 4*k* bits in BCD
- Decimal 396 is represented in BCD with 12 bits as 0011 1001 0110, with **each group of 4 bits representing one decimal digit**
- 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**



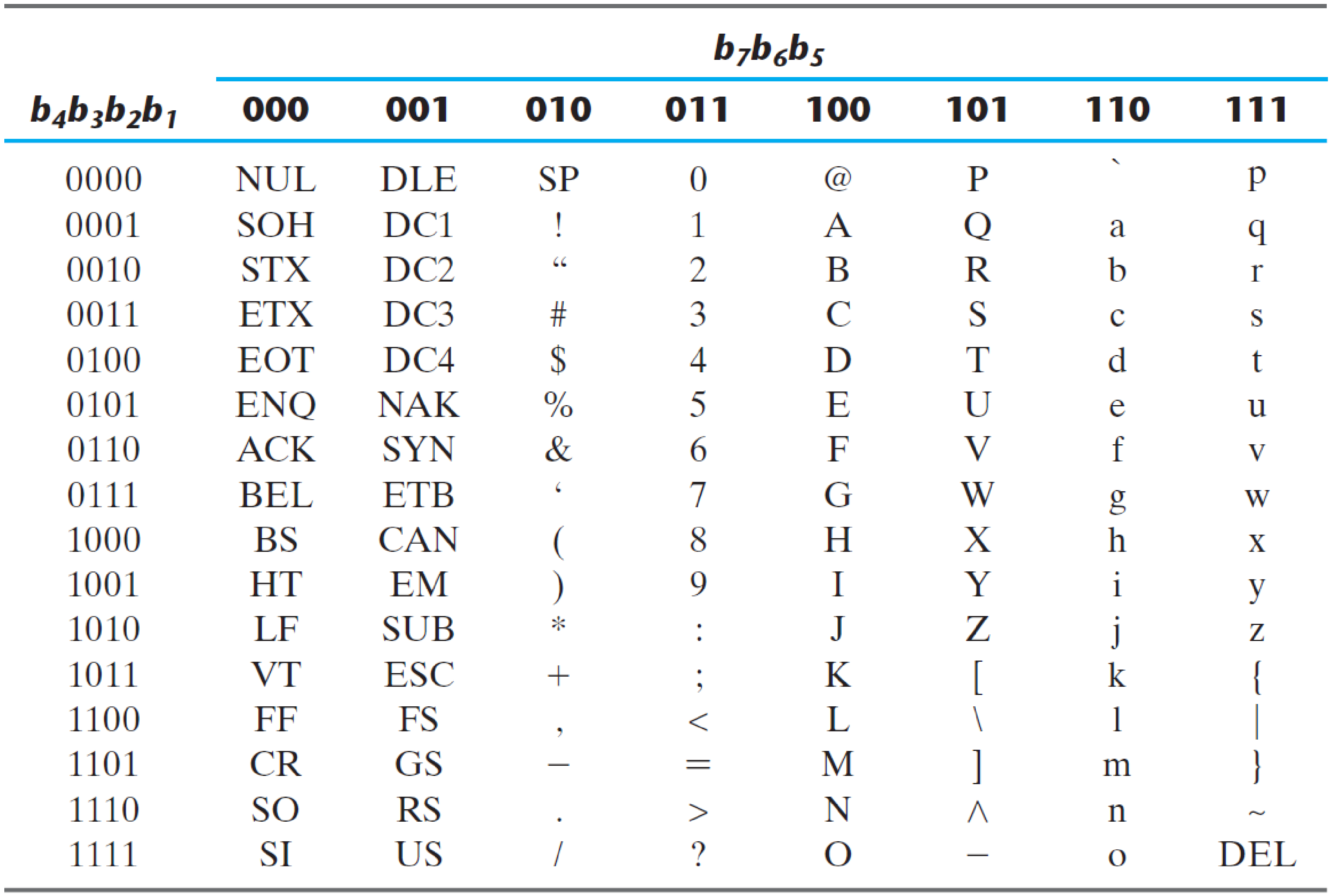
Binary-Coded Decimal (BCD)

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

ASCII code

- Many applications of digital computers require the handling not only of numbers, but also of other characters or symbols, such as the letters of the alphabet
- 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
- The seven bits of the code are designated by $b1$ through $b7$, with $b7$ the most significant bit
- The letter *A*, for example, is represented in ASCII as 1000001
- The ASCII code also contains 94 graphic characters that can be printed and 34 nonprinting characters used for various control functions
- The graphic characters consist of the 26 uppercase letters (A through Z), the 26 lowercase letters (a through z), the 10 numerals (0 through 9), and 32 special printable characters, such as %, *, and $

ASCII code



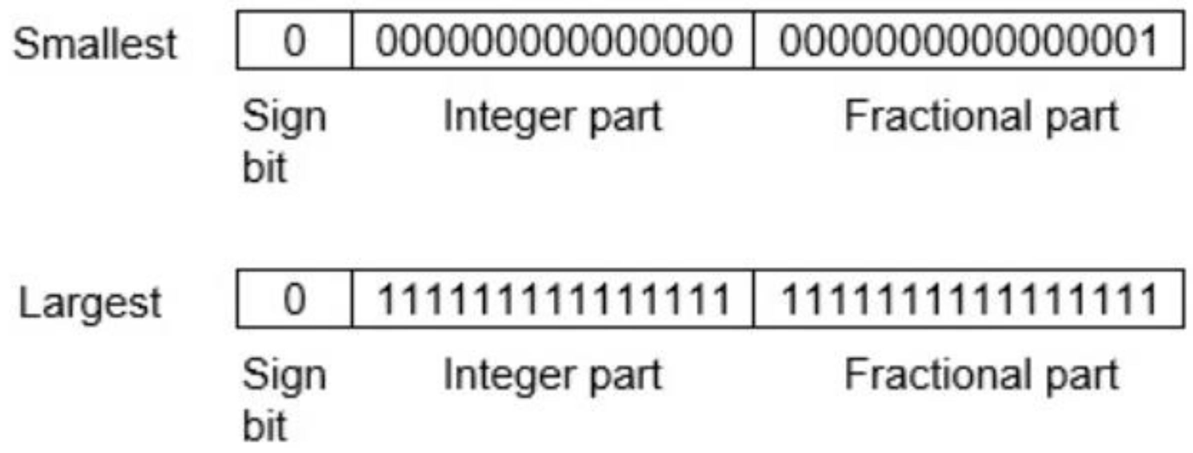
	$b_7b_6b_5$							
$b_4b_3b_2b_1$	**000**	**001**	**010**	**011**	**100**	**101**	**110**	**111**
0000	NUL	DLE	SP	0	@	P	`	p
0001	SOH	DC1	!	1	A	Q	a	q
0010	STX	DC2	“	2	B	R	b	r
0011	ETX	DC3	#	3	C	S	c	s
0100	EOT	DC4	$	4	D	T	d	t
0101	ENQ	NAK	%	5	E	U	e	u
0110	ACK	SYN	&	6	F	V	f	v
0111	BEL	ETB	‘	7	G	W	g	w
1000	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	{
1100	FF	FS	,	<	L	\	l	|
1101	CR	GS	–	=	M	]	m	}
1110	SO	RS	.	>	N	∧	n	~
1111	SI	US	/	?	O	—	o	DEL

Representing fractions (real numbers)

- We need to operate with fractions all the time
- This means we need a method to store/represent them in binary
- The simplest way is to have a “fixed” point representation where the binary point is assumed to be fixed at a certain location
- For example, for an 4-bit system, if given fixed-point representation is II.FF, then you can store minimum value is 00.01 (0001) and maximum value is 11.11 (1111)
- Remember the point is not actually stored – it is assumed to be there
- There are three parts of a fixed-point number representation: the sign field, integer field, and fractional field – which means we can also have signed magnitude fixed point numbers and signed complement fixed point numbers

Fixed point representation

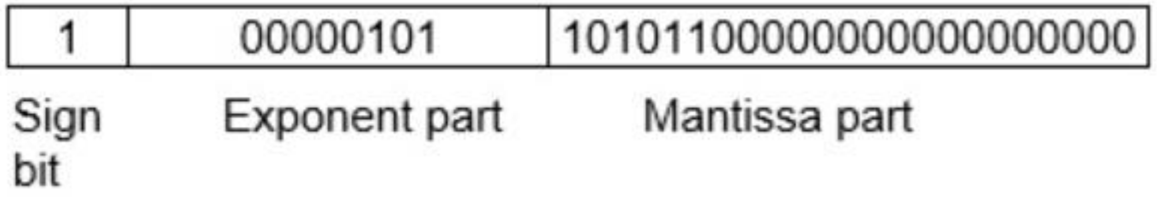
- The advantage of using a fixed-point representation is performance and ease of arithmetic
- The disadvantage is relatively limited range of values that they can represent
- So, it is usually inadequate for numerical analysis as it does not allow enough numbers and accuracy
- For instance, using 32-bit format: the 1 bit sign bit, 15 bits for integer, and 16 bits for the fractional part, the smallest positive number is $2^{-16} \approx 0.000015$, and the largest positive number is ≈ 32768



Floating point representation

(IEEE Standard 754 Floating point)

- This representation does not reserve a specific number of bits for the integer part or the fractional part
- Instead it reserves a certain number of bits for the number (called the mantissa) and a certain number of bits to say where within that number the decimal place sits (called the exponent)
- We convert the number to be stored as $N = M * r^e$ and store M and e as binary
- Clearly, a large mantissa and small exponent can give both high precision and high range
- The exponent field needs to represent both positive and negative exponents. To do this, a *bias* is added to the actual exponent in order to get the stored exponent
- For instance, using 32-bit format: the 1 bit sign bit, 8 bits for signed exponent, and 23 bits for the fractional part. Smallest is 1.18×10^{-38} and the largest is 3.40×10^{38}



$$-53.5=(-110101.1)_2=(-1.101011)\text{x}25$$

Floating Point Components

	Sign	Exponent	Fraction
Single Precision	1 [31]	8 [30–23]	23 [22–00]
Double Precision	1 [63]	11 [62–52]	52 [51–00]

Laid out as bits, floating point numbers look like this:

```
Single: SEEEEEEE EFFFFFFF FFFFFFFF FFFFFFFF
Double: SEEEEEEE EEEEFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF
```

Floating Point Range

	Denormalized	Normalized	Approximate Decimal
Single Precision	$\pm 2^{-149}$ to $(1-2^{-23})\times 2^{-126}$	$\pm 2^{-126}$ to $(2-2^{-23})\times 2^{127}$	$\pm \approx 10^{-44.85}$ to $\approx 10^{38.53}$
Double Precision	$\pm 2^{-1074}$ to $(1-2^{-52})\times 2^{-1022}$	$\pm 2^{-1022}$ to $(2-2^{-52})\times 2^{1023}$	$\pm \approx 10^{-323.3}$ to $\approx 10^{308.3}$

Read more here