CMPE 212
Principles of Digital Design

Lecture 5

Computer Codes

February 8, 2016

www.csee.umbc.edu/~younis/CMPE212/CMPE212.htm

Lecture's Overview

❑ *Previous Lecture:*

➔ Boolean Algebra
(History, postulates, theorems)

➔ Simplification of Boolean expressions
(how to put postulates and theorem to work)

➔ Boolean algebra and switching circuits
(Logic gates and algebraic method for minimization)

❑ *This Lecture:*

➔ Binary codes (BCD, Character representation)

➔ Representations for floating point numbers

➔ Error detect and correction codes

Computer Codes

- A binary code is a definition of the meaning of a sequence of bits of a certain size
- Encoding refers to defining the mapping between a set of words and the corresponding code words (bit sequence)
- Decoding refers to the opposite operation in which a sequence of bits are interpreted
- Sample categorization of binary codes:
  1. Numeric codes (fixed size integer and floating point)
  2. Character codes
  3. Cyclic codes
  4. Error detection and correction code

Excess (Biased)

- The leftmost bit is the sign (usually 1 = positive, 0 = negative). Representations of a number are obtained by adding a bias to the two's complement representation. This goes both ways, converting between positive and negative numbers.
- The effect is that numerically smaller numbers have smaller bit patterns, simplifying comparisons for floating point exponents.
- Example (excess 128 "adds" 128 to the two's complement version, ignoring any carry out of the most significant bit):

  $+12_{10} = 10001100_2$, $-12_{10} = 01110100_2$
- Only one representations for zero:

  $+0 = 10000000_2$, $-0 = 10000000_2$
- Range for an 8-bit representation is $[+127_{10}, -128_{10}]$
- Range for an N-bit representation is $[+(2^{N-1}-1)_{10}, -(2^{N-1})_{10}]$

* Slide is courtesy of M. Murdocca and V. Heuring

3-Bit Signed Integer Representations

Decimal	Unsigned	Sign–Mag.	1's Comp.	2's Comp.	Excess 4
7	111	–	–	–	–
6	110	–	–	–	–
5	101	–	–	–	–
4	100	–	–	–	–
3	011	011	011	011	111
2	010	010	010	010	110
1	001	001	001	001	101
+0	000	000	000	000	100
-0	–	100	111	000	100
-1	–	101	110	111	011
-2	–	110	101	110	010
-3	–	111	100	101	001
-4	–	–	–	100	000

* Slide is courtesy of M. Murdocca and V. Heuring

Binary Coded Decimal (BCD)

- Each binary coded decimal digit is composed of 4 bits.

(a)	0 0 0 0	0 0 1 1	0 0 0 0	0 0 0 1	$(+301)_{10}$	Nine's and ten's complement
	$(0)_{10}$	$(3)_{10}$	$(0)_{10}$	$(1)_{10}$		
(b)	1 0 0 1	0 1 1 0	1 0 0 1	1 0 0 0	$(-301)_{10}$	Nine's complement
	$(9)_{10}$	$(6)_{10}$	$(9)_{10}$	$(8)_{10}$		
(c)	1 0 0 1	0 1 1 0	1 0 0 1	1 0 0 1	$(-301)_{10}$	Ten's complement
	$(9)_{10}$	$(6)_{10}$	$(9)_{10}$	$(9)_{10}$		

- Example: Represent $+079_{10}$ in BCD: 0000 0111 1001
- Example: Represent -079_{10} in BCD: 1001 0010 0001

1. Subtract each digit of –079 from 9 to obtain the nine's complement, so 999 - 079 = 920.
2. Adding 1 produces the ten's complement: 920 + 1 = 921.
3. Converting each base 10 digit of 921 to BCD produces 1001 0010 0001

* Slide is courtesy of M. Murdocca and V. Heuring

Addition Example Using BCD

- Addition is performed digit by digit (not bit by bit), in 4-bit groups
- Example $(255 + 63 = 318)_{10}$:

	0	1	0	0 ←	Carries
	0 0 0 0	0 0 1 0	0 1 0 1	0 1 0 1	$(+255)_{10}$
	$(0)_{10}$	$(2)_{10}$	$(5)_{10}$	$(5)_{10}$	
+	0 0 0 0	0 0 0 0	0 1 1 0	0 0 1 1	$(+63)_{10}$
	$(0)_{10}$	$(0)_{10}$	$(6)_{10}$	$(3)_{10}$	
	0 0 0 0	0 0 1 1	0 0 0 1	1 0 0 0	$(+318)_{10}$
	$(0)_{10}$	$(3)_{10}$	$(1)_{10}$	$(8)_{10}$	

* Slide is courtesy of M. Murdocca and V. Heuring

Subtraction Example Using BCD

- Subtraction is carried out by adding the ten's complement negative of the subtrahend to the minuend
- Ten's complement negative of subtrahend is obtained by adding 1 to the nine's complement negative of the subtrahend
- Example: $(255 - 63 = 192)_{10}$

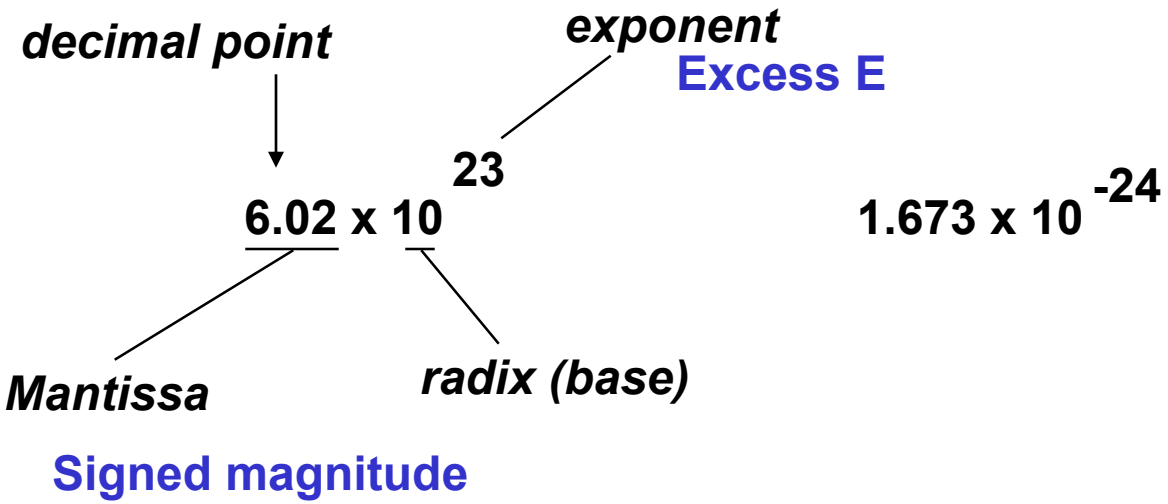
```
  9999
 -0063
 ------
  9936

  9936
 +0001
 ------
  9937
```

```
      1      1      0      1      0   <-- Carries
   0000   0010   0101   0101   (+255)10
+  1001   1001   0011   0111   (-63)10
-------------------------------------
1  0000   0001   1001   0010   (+192)10
^
Discard carry
```

* Slide is courtesy of M. Murdocca and V. Heuring

Floating Point Numbers



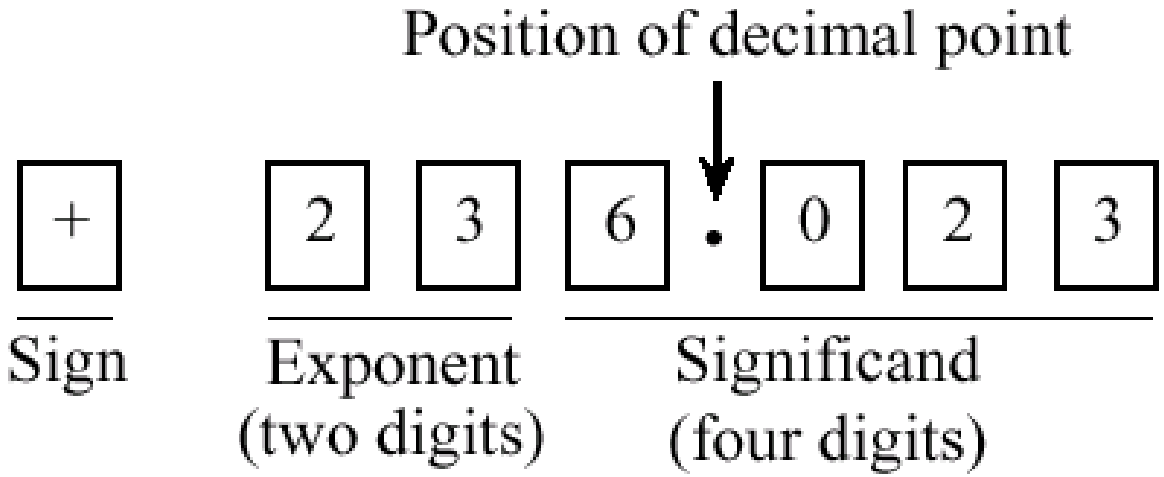
IEEE F.P. $\pm 1.M \times 2^{e - 127}$

- Issues:
  - Arithmetic (+, -, *, /)
  - Representation, Normal form (no leading zeros)
  - Range and Precision
  - Rounding
  - Exceptions (e.g., divide by zero, overflow, underflow)
  - Errors
  - Properties (negation, inversion, if $A \geq B$ then $A - B \geq 0$)

* Slide is courtesy of Dave Patterson

Base 10 Floating Point Numbers

- Floating point numbers allow very large and very small values to be represented using only few digits, at the expense of precision
- The precision is primarily determined by the number of digits in the fraction (or significand, which has integer and fractional parts), and the range is primarily determined by the number of digits in the exponent
- Example ($+6.023 \times 10^{23}$)



* Slide is courtesy of M. Murdocca and V. Heuring

Conversion Example

- Example: Convert $(9.375 \times 10^{-2})_{10}$ to base 2 scientific notation
- Start by converting from base 10 floating point to base 10 fixed point by moving the decimal point two positions to the left, which corresponds to the -2 exponent: .09375.
- Next, convert from base 10 fixed point to base 2 fixed point:

  .09375 × 2 = 0.1875
  .1875 × 2 = 0.375
  .375 × 2 = 0.75
  .75 × 2 = 1.5
  .5 × 2 = 1.0

- Thus, $(.09375)_{10} = (.00011)_2$.
- Finally, convert to normalized base 2 floating point:

  $.00011 = .00011 \times 2^0 = 1.1 \times 2^{-4}$

* Slide is courtesy of M. Murdocca and V. Heuring

Floating-Point Representation

- The size of the exponent determines the range of represented numbers
- Accuracy of the representation depends on the size of the significand
- The fixed word size requires a trade-off between accuracy and range
- Too large number cannot be represented causing an “overflow” while too small number cause an “underflow”
- Negative and positive mantissas are designated by a sign bit using a sign and magnitude representation
- Exponents are usually represented using “excess *E*” representation to facilitate comparison between floating point numbers
- Double precision uses multiple words to expand the range of both the exponent and mantissa and limits overflow and underflow conditions

Single precision

Double precision

Floating Point Example

- Represent $.254 \times 10^3$ in a normalized base 8 floating point format with a sign bit, followed by a 3-bit excess 4 exponent, followed by four base 8 digits

Step #1: Convert to the target base.

$.254 \times 10^3 = (254)_{10} = 376 \times 8^0$ (Using the remainder method)

254/8 = 31 R 6
31/8 = 3 R 7
3/8 = 0 R 3

Step #2: Normalize: $376 \times 8^0 = .376 \times 8^3$

Step #3: Fill in the bit fields, with a positive sign (sign bit = 0), an exponent of 3 + 4 = 7 (excess 4), and 4-digit fraction = .3760:

0 111 . 011 111 110 000

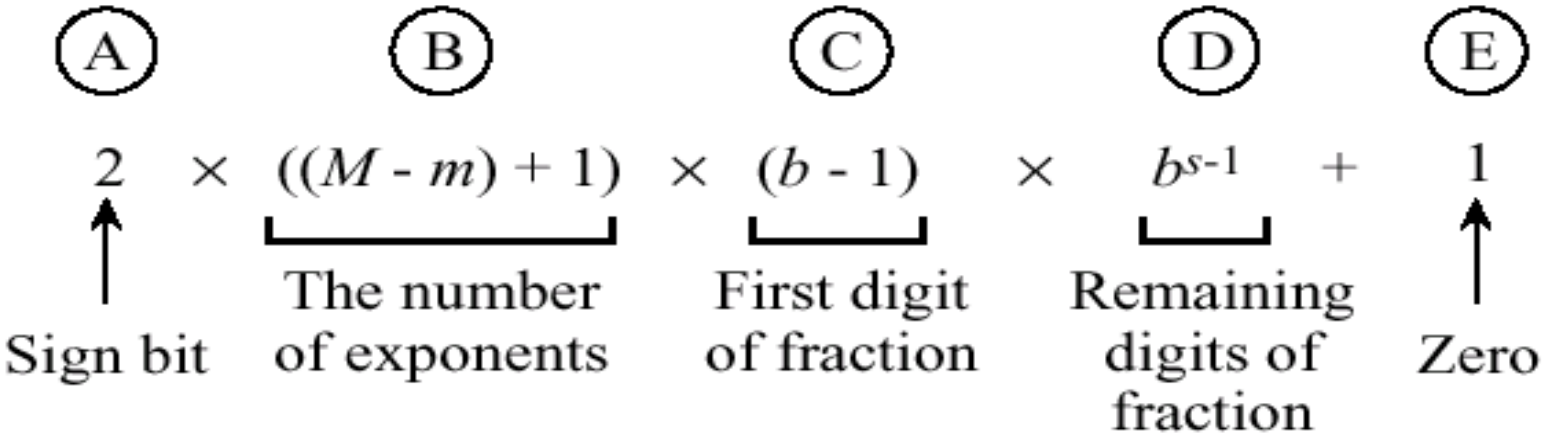
* Slide is courtesy of M. Murdocca and V. Heuring

Error, Range, and Precision

- In the previous example, we have the base *b* = *8*, the number of significant digits (not bits!) in the fraction *s* = *4*, the largest exponent value (not bit pattern) *M* = *3*, and the smallest exponent value *m* = *-4*
- In the previous example, there is no explicit representation of 0, but there needs to be a special bit pattern reserved for 0 otherwise there would be no way to represent 0 without violating the normalization rule. We will assume a bit pattern of *0 000 000 000 000 000* represents *0*
- Using *b*, *s*, *M*, and *m*, generally:
  - Largest representable number: $b^M \times (1 - b^{-s}) = 8^3 \times (1 - 8^{-4})$
  - Smallest representable number: $b^m \times b^{-1} = 8^{-4 - 1} = 8^{-5}$
  - Largest gap between two successive numbers: $b^M \times b^{-s} = 8^{3 - 4} = 8^{-1}$
  - Smallest gap between two successive numbers: $b^m \times b^{-s} = 8^{-4 - 4} = 8^{-8}$

* Slide is courtesy of M. Murdocca and V. Heuring

Counting Representable Number



For the total count of representable numbers, there are 5 components:

(A) sign bit

(B) *(M - m) + 1* bits for exponents

(C) *b – 1* values for the first digit (0 is disallowed in normalized numbers)

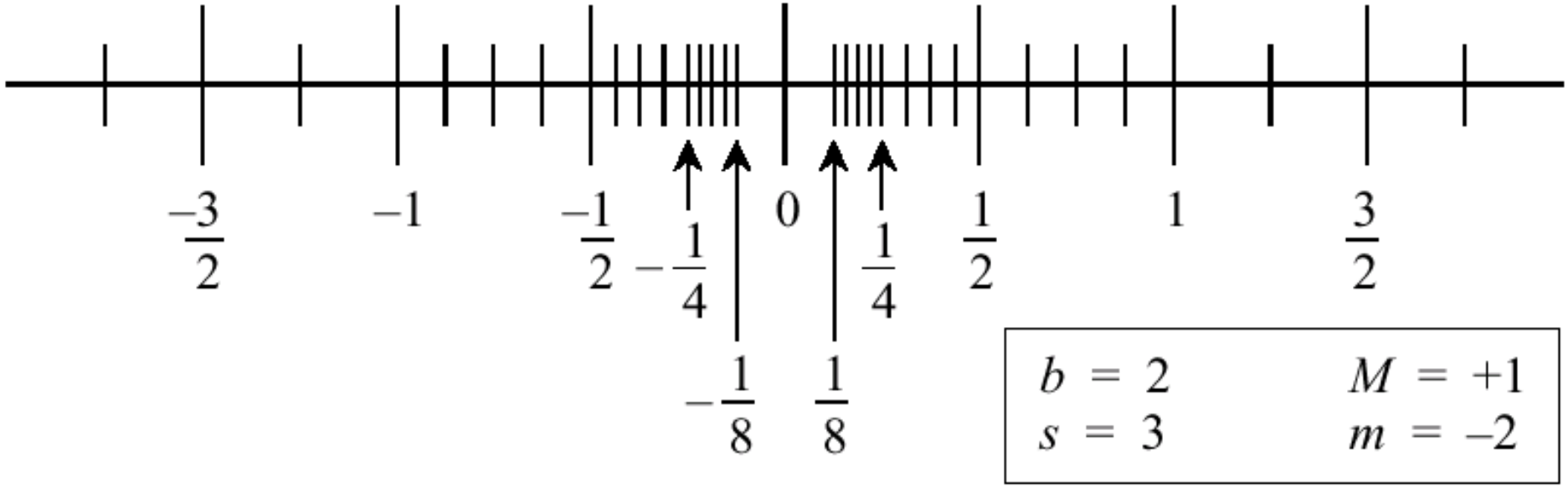
(D) b^{s-1} values for each of the *s-1* remaining digits,

(E) A special representation for 0

- For $.376 \times 8^3$, the 5 components result in:
  $2 \times ((3 - 4) + 1) \times (8 - 1) \times 8^{4-1} + 1$ numbers that can be represented
- Notice this number must be no greater than the number of possible bit patterns that can be generated, which is 2^{16}

* Slide is courtesy of M. Murdocca and V. Heuring

Example Floating Point Format



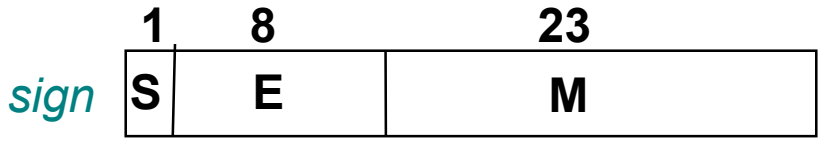
- Smallest number is 1/8
- Largest number is 7/4
- Smallest gap is 1/32
- Largest gap is 1/4
- Number of representable numbers is 33.

* Slide is courtesy of M. Murdocca and V. Heuring

IEEE 754 Standard Representation

- Virtually has been used in every computer since 1980

Single precision



exponent:
excess 127
binary integer

mantissa:
sign + magnitude, normalized
binary significand w/ hidden
integer bit: 1.M

Actual exponent is
e = E - 127

$$N = (-1)^S \, 2^{E-127} \, (1.M) \qquad 0 < E < 255$$

0 = 0 00000000 0 . . . 0

-1.5 = 1 01111111 10 . . . 0

- Magnitude of numbers that can be represented is in the range:

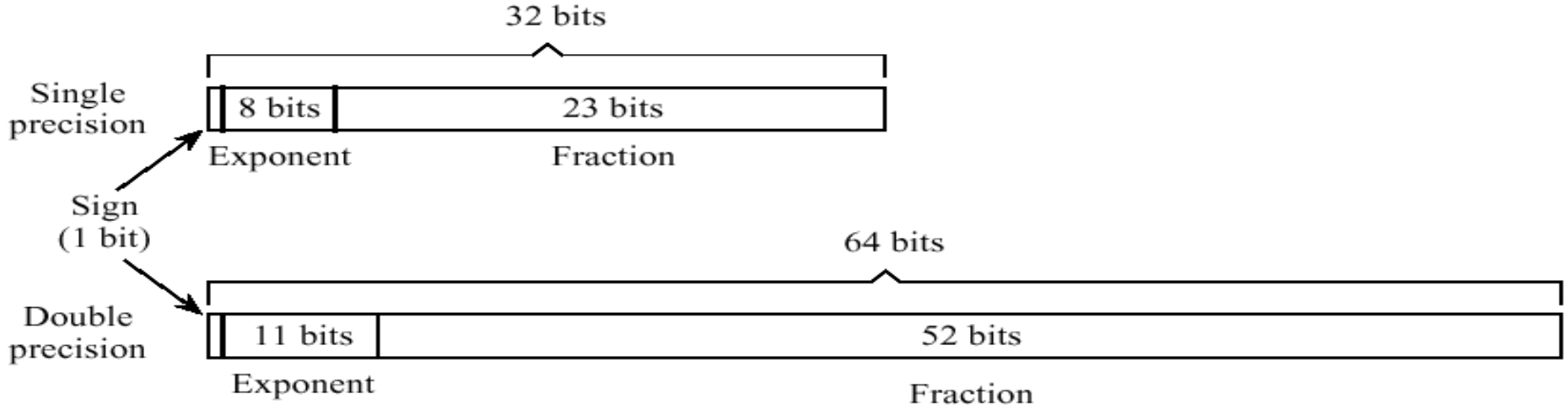
$$2^{-126}(1.0) \quad \text{to} \quad 2^{127}(2 - 2^{-23})$$

which is approximately:

$$1.8 \times 10^{-38} \quad \text{to} \quad 3.40 \times 10^{38}$$

Integer comparison is valid on IEEE Floating Point numbers of same sign

IEEE-754 Floating Point Formats



Example: show -12.625_{10} in single precision IEEE-754 format.

Step #1: Convert to target base. $-12.625_{10} = -1100.101_2$

Step #2: Normalize. $-1100.101_2 = -1.100101_2 \times 2^3$

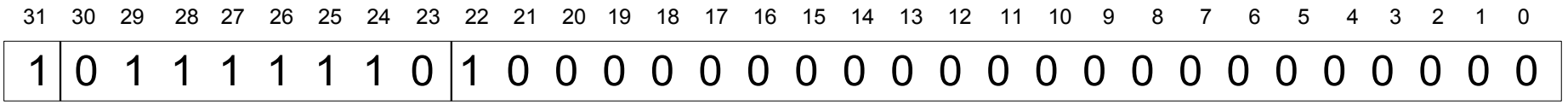
Step #3: Fill in bit fields. Sign is negative, so sign bit is 1. Exponent is in excess 127 (not excess 128!), so exponent is represented as the unsigned integer 3 + 127 = 130. Leading 1 of significand is hidden, so final bit pattern is:

1 1000 0010 . 1001 0100 0000 0000 0000 000

* Slide is courtesy of M. Murdocca and V. Heuring

An Example

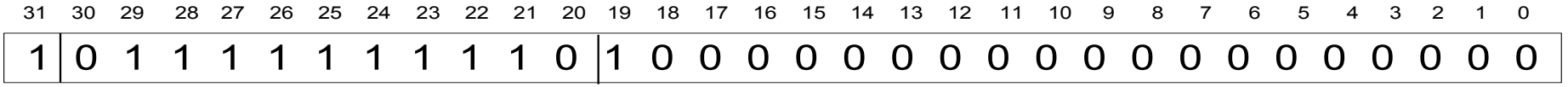
Show the IEEE 754 binary representation of -0.75 in single & double precision

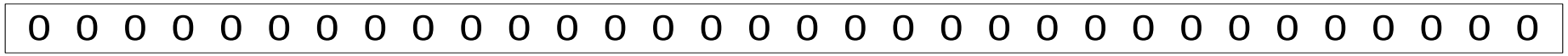


$(-0.75)_{10} = (-3/4)_{10} = (-3/2^2)_{10} = (-11 \times 2^{-2})_2 = (-0.11)_2 = (-1.1 \times 2^{-1})_2$

Single precision representation is: $(-1)^s \times (1+\text{Significand}) \times 2^{(\text{Exponent} + 127)}$

$(-0.75)_{10}$ is represented as $(-1)^1 \times (1+.1000\ 0000\ 0000\ 0000\ 0000\ 000) \times 2^{(126)}$





Double precision representation is: $(-1)^s \times (1+\text{Significand}) \times 2^{(\text{Exponent} + 1023)}$

$(-0.75)_{10}$ is represented as $(-1)^1 \times (1+.1000\ 0000\ \ldots..\ 0000\ 0000) \times 2^{(1022)}$

Symbol Representation

- Early versions (60s and 70s):
  - Six-bit binary code (Control Data Corp., CDC)
  - EBCDIC – extended binary coded decimal interchange code (IBM)
- Presently used:
  - ASCII – American standard code for information interchange – 7 bit code specified by American National Standards Institute (ANSI); an eighth MSB is often used as parity bit to construct a byte-code.
  - ISO/IEC 646 a version of ASCII that supports extension for languages other than English.
  - Unicode – 16 bit code and an extended 32 bit version

ASCII Character Code

- ASCII is a 7-bit code, commonly stored in 8-bit bytes
- "A" is at 41_{16} . To convert upper case letters to lower case letters, add 20_{16} ➔ "a" is at $41_{16} + 20_{16} = 61_{16}$
- The character "5" at position 35_{16} is different from the number 5. To convert character-numbers into number-numbers, subtract 30_{16} :

  $35_{16} - 30_{16} = 5_{16}$

00	NUL	10	DLE	20	SP	30	0	40	@	50	P	60	`	70	p
01	SOH	11	DC1	21	!	31	1	41	A	51	Q	61	a	71	q
02	STX	12	DC2	22	"	32	2	42	B	52	R	62	b	72	r
03	ETX	13	DC3	23	#	33	3	43	C	53	S	63	c	73	s
04	EOT	14	DC4	24	$	34	4	44	D	54	T	64	d	74	t
05	ENQ	15	NAK	25	%	35	5	45	E	55	U	65	e	75	u
06	ACK	16	SYN	26	&	36	6	46	F	56	V	66	f	76	v
07	BEL	17	ETB	27	'	37	7	47	G	57	W	67	g	77	w
08	BS	18	CAN	28	(	38	8	48	H	58	X	68	h	78	x
09	HT	19	EM	29	)	39	9	49	I	59	Y	69	i	79	y
0A	LF	1A	SUB	2A	*	3A	:	4A	J	5A	Z	6A	j	7A	z
0B	VT	1B	ESC	2B	+	3B	;	4B	K	5B	[	6B	k	7B	{
0C	FF	1C	FS	2C	'	3C	<	4C	L	5C	\	6C	l	7C	|
0D	CR	1D	GS	2D	-	3D	=	4D	M	5D	]	6D	m	7D	}
0E	SO	1E	RS	2E	.	3E	>	4E	N	5E	^	6E	n	7E	~
0F	SI	1F	US	2F	/	3F	?	4F	O	5F	_	6F	o	7F	DEL

NUL	Null	FF	Form feed	CAN	Cancel
SOH	Start of heading	CR	Carriage return	EM	End of medium
STX	Start of text	SO	Shift out	SUB	Substitute
ETX	End of text	SI	Shift in	ESC	Escape
EOT	End of transmission	DLE	Data link escape	FS	File separator
ENQ	Enquiry	DC1	Device control 1	GS	Group separator
ACK	Acknowledge	DC2	Device control 2	RS	Record separator
BEL	Bell	DC3	Device control 3	US	Unit separator
BS	Backspace	DC4	Device control 4	SP	Space
HT	Horizontal tab	NAK	Negative acknowledge	DEL	Delete
LF	Line feed	SYN	Synchronous idle		
VT	Vertical tab	ETB	End of transmission block		

* Slide is courtesy of M. Murdocca and V. Heuring

EBCDIC Character Code

- **EBCDIC is an 8-bit code.**

00	NUL	20	DS	40	SP	60	–	80		A0		C0	{	E0	\
01	SOH	21	SOS	41		61	/	81	a	A1	~	C1	A	E1	
02	STX	22	FS	42		62		82	b	A2	s	C2	B	E2	S
03	ETX	23		43		63		83	c	A3	t	C3	C	E3	T
04	PF	24	BYP	44		64		84	d	A4	u	C4	D	E4	U
05	HT	25	LF	45		65		85	e	A5	v	C5	E	E5	V
06	LC	26	ETB	46		66		86	f	A6	w	C6	F	E6	W
07	DEL	27	ESC	47		67		87	g	A7	x	C7	G	E7	X
08		28		48		68		88	h	A8	y	C8	H	E8	Y
09		29		49		69		89	i	A9	z	C9	I	E9	Z
0A	SMM	2A	SM	4A	¢	6A	‘	8A		AA		CA		EA	
0B	VT	2B	CU2	4B		6B	,	8B		AB		CB		EB	
0C	FF	2C		4C	<	6C	%	8C		AC		CC		EC	
0D	CR	2D	ENQ	4D	(	6D	_	8D		AD		CD		ED	
0E	SO	2E	ACK	4E	+	6E	>	8E		AE		CE		EE	
0F	SI	2F	BEL	4F	|	6F	?	8F		AF		CF		EF	
10	DLE	30		50	&	70		90		B0		D0	}	F0	0
11	DC1	31		51		71		91	j	B1		D1	J	F1	1
12	DC2	32	SYN	52		72		92	k	B2		D2	K	F2	2
13	TM	33		53		73		93	l	B3		D3	L	F3	3
14	RES	34	PN	54		74		94	m	B4		D4	M	F4	4
15	NL	35	RS	55		75		95	n	B5		D5	N	F5	5
16	BS	36	UC	56		76		96	o	B6		D6	O	F6	6
17	IL	37	EOT	57		77		97	p	B7		D7	P	F7	7
18	CAN	38		58		78		98	q	B8		D8	Q	F8	8
19	EM	39		59		79		99	r	B9		D9	R	F9	9
1A	CC	3A		5A	!	7A	:	9A		BA		DA		FA	|
1B	CU1	3B	CU3	5B	$	7B	#	9B		BB		DB		FB	
1C	IFS	3C	DC4	5C	·	7C	@	9C		BC		DC		FC	
1D	IGS	3D	NAK	5D	)	7D	'	9D		BD		DD		FD	
1E	IRS	3E		5E	;	7E	=	9E		BE		DE		FE	
1F	IUS	3F	SUB	5F	¬	7F	"	9F		BF		DF		FF	

STX	Start of text	RS	Reader Stop	DC
DLE	Data Link Escape	PF	Punch Off	DC
BS	Backspace	DS	Digit Select	DC
ACK	Acknowledge	PN	Punch On	CU
SOH	Start of Heading	SM	Set Mode	CU
ENQ	Enquiry	LC	Lower Case	CU
ESC	Escape	CC	Cursor Control	SY
BYP	Bypass	CR	Carriage Return	IF
CAN	Cancel	EM	End of Medium	EC
RES	Restore	FF	Form Feed	ET
SI	Shift In	TM	Tape Mark	NA
SO	Shift Out	UC	Upper Case	SM
DEL	Delete	FS	Field Separator	SC
SUB	Substitute	HT	Horizontal Tab	IG
NL	New Line	VT	Vertical Tab	IR
LF	Line Feed	UC	Upper Case	IU

* Slide is courtesy of M. Murdocca and V. Heuring

Unicode Character Code

Unicode is a 16-bit code

0000	NUL	0020	SP	0040	@	0060	`	0080	Ctrl	00A0	NBS	00C0	À	00E0	à
0001	SOH	0021	!	0041	A	0061	a	0081	Ctrl	00A1	¡	00C1	Á	00E1	á
0002	STX	0022	"	0042	B	0062	b	0082	Ctrl	00A2	¢	00C2	Â	00E2	â
0003	ETX	0023	#	0043	C	0063	c	0083	Ctrl	00A3	£	00C3	Ã	00E3	ã
0004	EOT	0024	$	0044	D	0064	d	0084	Ctrl	00A4	¤	00C4	Ä	00E4	ä
0005	ENQ	0025	%	0045	E	0065	e	0085	Ctrl	00A5	¥	00C5	Å	00E5	å
0006	ACK	0026	&	0046	F	0066	f	0086	Ctrl	00A6	¦	00C6	Æ	00E6	æ
0007	BEL	0027	'	0047	G	0067	g	0087	Ctrl	00A7	§	00C7	Ç	00E7	ç
0008	BS	0028	(	0048	H	0068	h	0088	Ctrl	00A8	¨	00C8	È	00E8	è
0009	HT	0029	)	0049	I	0069	i	0089	Ctrl	00A9	©	00C9	É	00E9	é
000A	LF	002A	*	004A	J	006A	j	008A	Ctrl	00AA	ª	00CA	Ê	00EA	ê
000B	VT	002B	+	004B	K	006B	k	008B	Ctrl	00AB	«	00CB	Ë	00EB	ë
000C	FF	002C	´	004C	L	006C	l	008C	Ctrl	00AC	¬	00CC	Ì	00EC	ì
000D	CR	002D	-	004D	M	006D	m	008D	Ctrl	00AD	–	00CD	Í	00ED	í
000E	SO	002E	.	004E	N	006E	n	008E	Ctrl	00AE	®	00CE	Î	00EE	î
000F	SI	002F	/	004F	O	006F	o	008F	Ctrl	00AF	¯	00CF	Ï	00EF	ï
0010	DLE	0030	0	0050	P	0070	p	0090	Ctrl	00B0	°	00D0	Ð	00F0	¶
0011	DC1	0031	1	0051	Q	0071	q	0091	Ctrl	00B1	±	00D1	Ñ	00F1	ñ
0012	DC2	0032	2	0052	R	0072	r	0092	Ctrl	00B2	²	00D2	Ò	00F2	ò
0013	DC3	0033	3	0053	S	0073	s	0093	Ctrl	00B3	³	00D3	Ó	00F3	ó
0014	DC4	0034	4	0054	T	0074	t	0094	Ctrl	00B4	´	00D4	Ô	00F4	ô
0015	NAK	0035	5	0055	U	0075	u	0095	Ctrl	00B5	µ	00D5	Õ	00F5	õ
0016	SYN	0036	6	0056	V	0076	v	0096	Ctrl	00B6	¶	00D6	Ö	00F6	ö
0017	ETB	0037	7	0057	W	0077	w	0097	Ctrl	00B7	·	00D7	×	00F7	+
0018	CAN	0038	8	0058	X	0078	x	0098	Ctrl	00B8	¸	00D8	Ø	00F8	ø
0019	EM	0039	9	0059	Y	0079	y	0099	Ctrl	00B9	¹	00D9	Ù	00F9	ù
001A	SUB	003A	:	005A	Z	007A	z	009A	Ctrl	00BA	º	00DA	Ú	00FA	ú
001B	ESC	003B	;	005B	[	007B	{	009B	Ctrl	00BB	»	00DB	Û	00FB	û
001C	FS	003C	<	005C	\	007C	|	009C	Ctrl	00BC	1/4	00DC	Ü	00FC	ü
001D	GS	003D	=	005D	]	007D	}	009D	Ctrl	00BD	1/2	00DD	Ý	00FD	Þ
001E	RS	003E	>	005E	^	007E	~	009E	Ctrl	00BE	3/4	00DE	ý	00FE	þ
001F	US	003F	?	005F	_	007F	DEL	009F	Ctrl	00BF	¿	00DF	§	00FF	ÿ

NUL	Null	SOH	Start of heading	CAN	Cancel	SP	Space
STX	Start of text	EOT	End of transmission	EM	End of medium	DEL	Delete
ETX	End of text	DC1	Device control 1	SUB	Substitute	Ctrl	Control
ENQ	Enquiry	DC2	Device control 2	ESC	Escape	FF	Form feed
ACK	Acknowledge	DC3	Device control 3	FS	File separator	CR	Carriage return
BEL	Bell	DC4	Device control 4	GS	Group separator	SO	Shift out
BS	Backspace	NAK	Negative acknowledge	RS	Record separator	SI	Shift in
HT	Horizontal tab	NBS	Non-breaking space	US	Unit separator	DLE	Data link escape
LF	Line feed	ETB	End of transmission block	SYN	Synchronous idle	VT	Vertical tab

* Slide is courtesy of M. Murdocca and V. Heuring

Gray Code

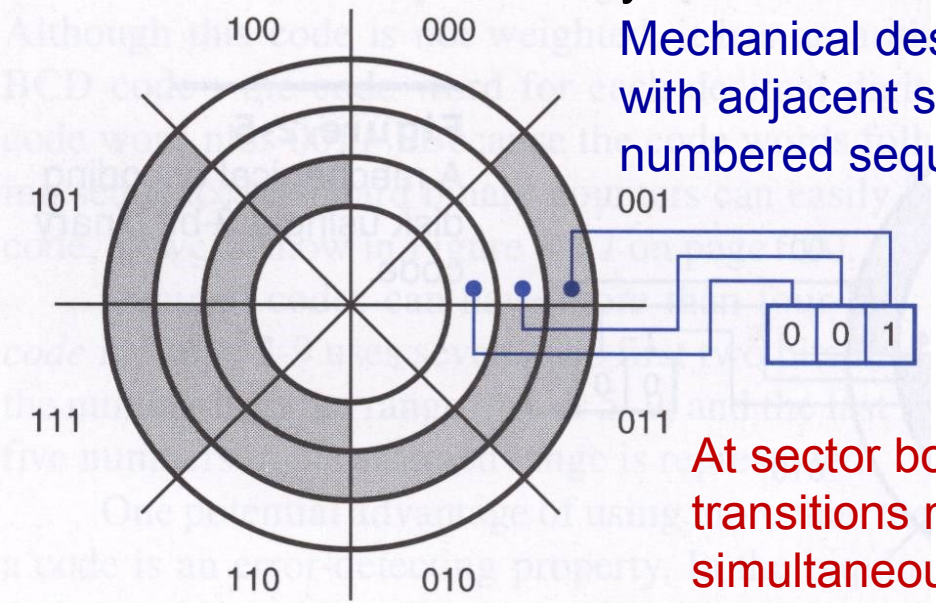
Cyclic codes

- A circular shift of any code word produces a valid code word (the new word may stay the same)

Gray code – example of a cyclic code

- Code words for consecutive numbers differ by one bit only
- Can be generated from the n-bit binary numbers as follows:
  - Bit *i* equals to 1 only if bits *i* and *i+1* in the binary number are different
  - Assume 0 in bit n+1;

Decimal	Binary	Gray
0	000	000
1	001	001
2	010	011
3	011	010
4	100	110
5	101	111
6	110	101
7	111	100



Mechanical desk encoder with adjacent sectors numbered sequentially

Gray coding allows only one bit change in transition

At sector boundaries transitions may not be simultaneous, causing to incorrect number (transient)

Error Detection Code

- Error in binary data ➔ an incorrect value in one or multiple bits
- Bits can flip due to hardware failure, noise in circuits and in transition during transmission (communication).
- Flip errors is a major concern in space and in wireless system
- Extra bits used for error detection and correction (need more)
- The number of redundant bits increases if multiple bits error is to be detected and tolerated.
- Example: a parity bit in ASCII code

7-bit ASCII code

Even parity code for A (even number of 1s) 01000001

Parity bits

Odd parity code for A (odd number of 1s) 11000001

Single-bit error in 7-bit code of “A”, e.g., 1000101, will change symbol to “E” or 1000000 to “@”. But error will be detected in the 8-bit code because the error changes the specified parity.

Slide contents is (partially) curtsey of Vishwani Agrawal

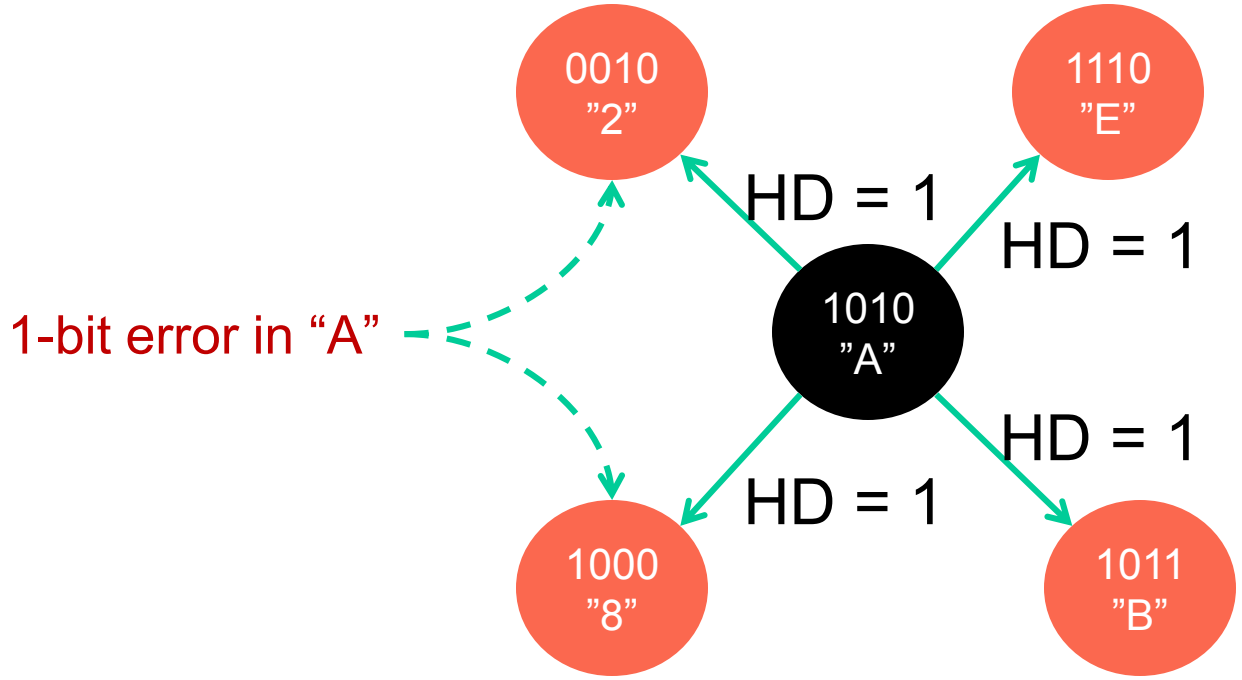
Fundamental Definitions

- Weight of a fixed size binary data is the number of 1's in the data
  - W(1011101) = 5
- The distance between two equal-sized binary numbers (or data items) is the wright of their difference (XOR)
- Usually referred to as hamming distance
    - HD(1101, 1010) = 3
- Richard Hamming is the one of the pioneers in coding who is to be credited for breakthroughs in the development of error-correcting codes (ECC)

Richard W. Hamming
1915 -1998

The Idea of Hamming Code

Code space contains 2^N possible N-bit code words:

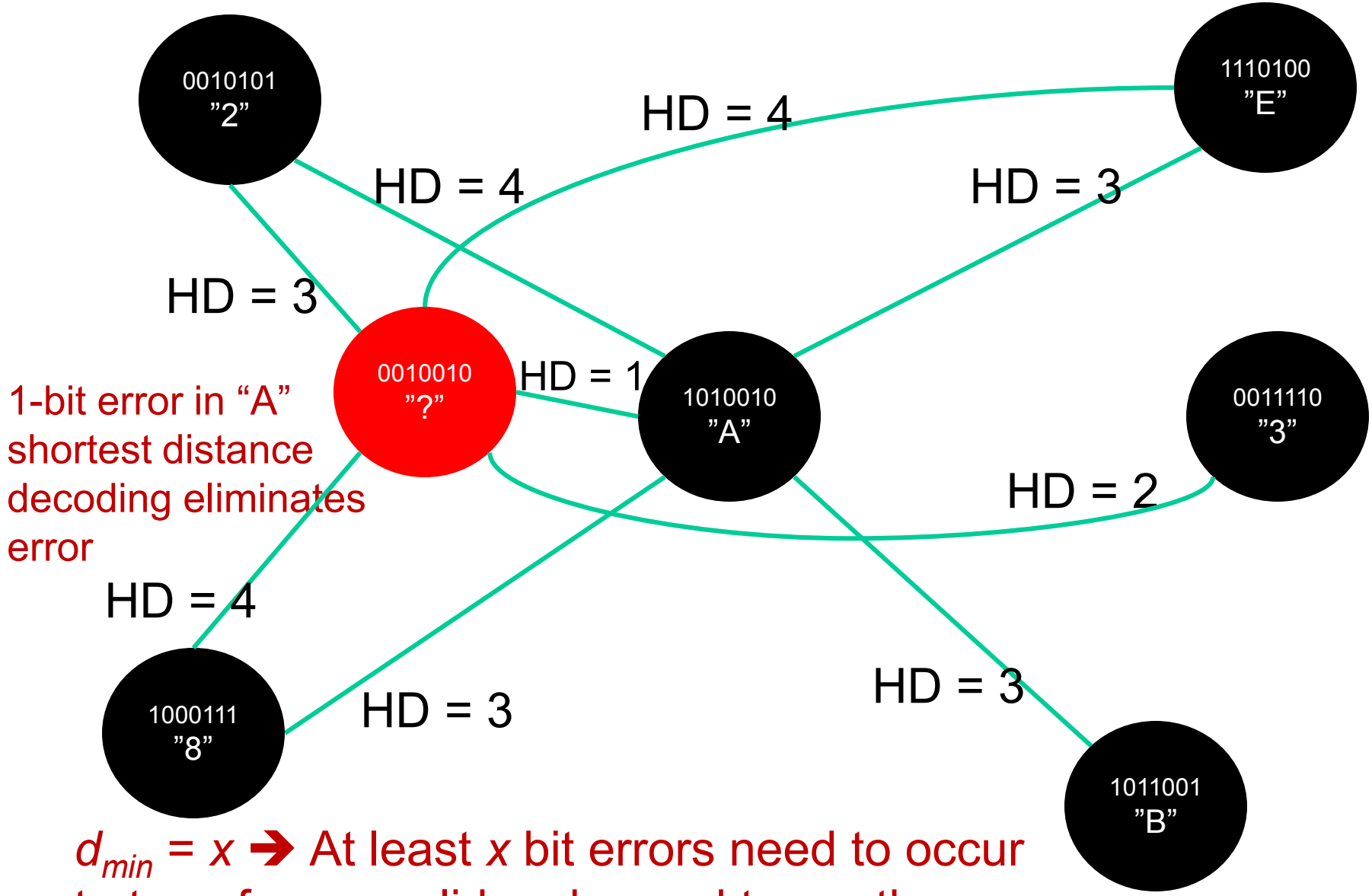


N = 4 Code	Symbol
0000	0
0001	1
0010	2
0011	3
0100	4
0101	5
0110	6
0111	7
1000	8
1001	9
1010	A
1011	B
1100	C
1101	D
1110	E
1111	F

Error not correctable. Reason: No redundancy.

Hamming's idea: Increase HD between valid code words.

Hamming's Distance ≥ 3 Code



$d_{min} = x$ ➔ At least x bit errors need to occur to transform a valid code word to another

Slide contents is curtsey of Vishwani Agrawal

Minimum Distance-3 Hamming Code

Symbol	Original code	Odd-parity code	ECC, HD ≥ 3
0	0000	10000	0000000
1	0001	00001	0001011
2	0010	00010	0010101
3	0011	10011	0011110
4	0100	00100	0100110
5	0101	10101	0101101
6	0110	10110	0110011
7	0111	00111	0111000
8	1000	01000	1000111
9	1001	11001	1001100
A	1010	11010	1010010
B	1011	01011	1011001
C	1100	11100	1100001
D	1101	01101	1101010
E	1110	01110	1110100
F	1111	11111	1111111

Original code: Symbol "0" with a single-bit error will be Interpreted as "1", "2", "4" or "8".

Reason: Hamming distance between codes is 1. A code with any bit error will map onto another valid code.

Remedy 1: Design codes with HD ≥ 2.

Example: Parity code. Single bit error Detectable but not correctable.

Remedy 2: Design codes with HD ≥ 3. For single bit error correction, decode as the valid code at HD = 1.

To detect "s" and correct "t " bit errors: $2t + s + 1 \leq d_{min}$

Slide contents is (mostly) curtsey of Vishwani Agrawal

Conclusion

❑ *Summary*

➔ Multiplication and division of binary numbers
(unsigned numbers, determining the sign of the results)

➔ Binary codes (BCD, Character representation)
(BCD, ASCII, EBCDIC, Unicode, Gray, etc.)

➔ Representations for floating point numbers
(Format, scientific notation, standard notation)

➔ Error detect and correction codes
(weight, Hamming distance, Hamming codes)

❑ *Next Lecture*

➔ Switching Functions

Reading assignment: section 1.5 in the textbook