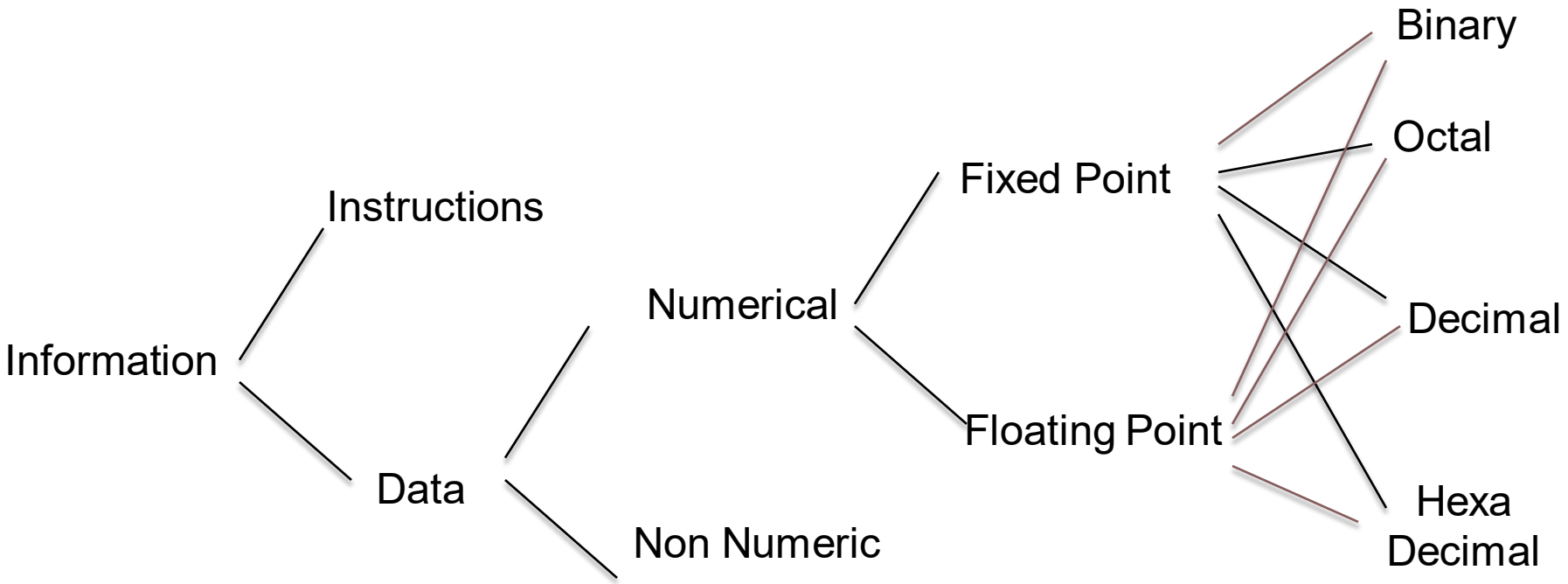
Data Representation

Data Representation



Data Representation- Integer-Fixed point

- Unsigned Number
- Signed Number
- One's Complement
- Two's Complement
- Biased one(Not Commonly used)

Cont

- Unsigned numbers: only non-negative values.
- Signed numbers: include all values (positive and negative).
- There are three common ways of representing signed numbers (positive and negative numbers) for binary numbers:
  - Sign-and-Magnitude
  - 1s Complement
  - 2s Complement

Sign-and-Magnitude

- Negative numbers are usually written by writing a minus sign in front.
  - Example:

    $-(12)_{10}$, $-(1100)_2$

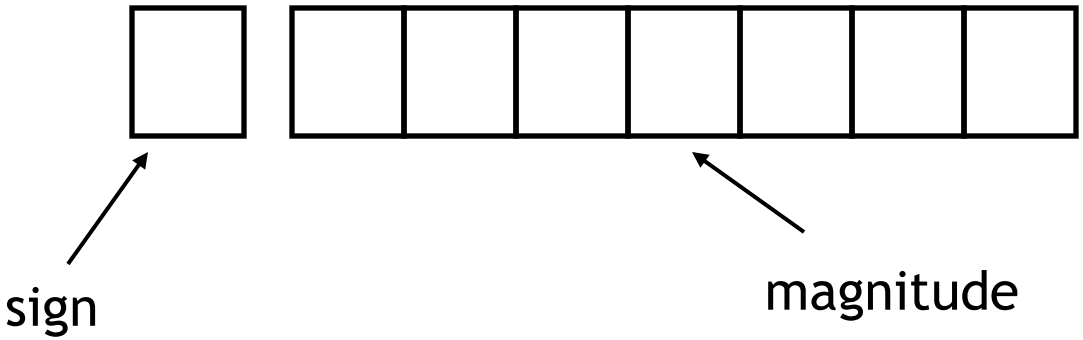
- In sign-and-magnitude representation, this sign is usually represented by a bit:

  0 for +

  1 for −

Negative Numbers: Sign-and-Magnitude (Cont')

- Example: an 8-bit number can have 1-bit sign and 7-bit magnitude.



Signed magnitude representation (cont')

.

- Examples:

$1101_2 = 13_{10}$ (a 4-bit unsigned number)
0 1101 $= +13_{10}$ (a positive number in 5-bit signed magnitude)
1 1101 $= -13_{10}$ (a negative number in 5-bit signed magnitude)

$0100_2 = 4_{10}$ (a 4-bit unsigned number)
0 0100 $= +4_{10}$ (a positive number in 5-bit signed magnitude)
1 0100 $= -4_{10}$ (a negative number in 5-bit signed magnitude)

Negative Numbers: Sign-and-Magnitude (Cont')

- To negate a number, just invert the sign bit.
- Examples:
  - $(0\ 0100001)_{sm} = (1\ 0100001)_{sm}$
  - $(1\ 0000101)_{sm} = (0\ 0000101)_{sm}$

1s and 2s Complement

- Two other ways of representing signed numbers for binary numbers are:
  - 1s-complement
  - 2s-complement
- They are preferred over the simple sign-and-magnitude representation.

One's complement representation

- A different approach, one's complement, negates numbers by complementing each bit of the number.
- We keep the sign bits: 0 for positive numbers, and 1 for negative. The sign bit is complemented along with the rest of the bits.
- Examples:

$1101_2 = 13_{10}$ (a 4-bit unsigned number)
0 1101 $= +13_{10}$ (a positive number in 5-bit one's complement)
1 0010 $= -13_{10}$ (a negative number in 5-bit one's complement)

$0100_2 = 4_{10}$ (a 4-bit unsigned number)
0 0100 $= +4_{10}$ (a positive number in 5-bit one's complement)
1 1011 $= -4_{10}$ (a negative number in 5-bit one's complement)

Why is it called “one’s complement?”

- Complementing a single bit is equivalent to subtracting it from 1.

  $0' = 1$, and $1 - 0 = 1$ $1' = 0$, and $1 - 1 = 0$

- Similarly, complementing each bit of an n-bit number is equivalent to subtracting that number from 2^n-1.
- For example, we can negate the 5-bit number 01101.
  - Here n=5, and $2^n-1 = 31_{10} = 11111_2$.
  - Subtracting 01101 from 11111 yields 10010:

```
  1 1 1 1 1
- 0 1 1 0 1
-----------
  1 0 0 1 0
```

1s Complement Addition/Subtraction

- Algorithm for addition, A + B:

1. Perform binary addition on the two numbers.
2. If there is a carry out of the MSB, add 1 to the result.
3. Check for overflow: Overflow occurs if result is opposite sign of A and B.

- Algorithm for subtraction, A – B:

  A – B = A + (–B)

1. Take 1s complement of B by inverting all the bits.
2. Add the 1s complement of B to A.

One's complement addition

- To add one's complement numbers:
  - First do unsigned addition on the numbers, *including* the sign bits.
  - Then take the carry out and add it to the sum.
- Two examples:

```
   0111     (+7)
+  1011   + (-4)
  -------  -----
 1 0010

   0010
+     1
  -------
   0011     (+3)
```

```
    0011     (+3)
+   0010   + (+2)
  ---------  -----
  0 0101

    0101
+     0
  -------
    0101     (+5)
```

- This is simpler and more uniform than signed magnitude addition.

Two's complement

- Our final idea is two's complement. To negate a number, complement each bit (just as for ones' complement) and then add 1.
- Examples:

$1101_2 = 13_{10}$ (a 4-bit unsigned number)

0 1101 $= +13_{10}$ (a positive number in 5-bit two's complement)

1 0010 $= -13_{10}$ (a negative number in 5-bit *ones'* complement)

1 0011 $= -13_{10}$ (a negative number in 5-bit two's complement)

$0100_2 = 4_{10}$ (a 4-bit unsigned number)

0 0100 $= +4_{10}$ (a positive number in 5-bit two's complement)

1 1011 $= -4_{10}$ (a negative number in 5-bit *ones'* complement)

1 1100 $= -4_{10}$ (a negative number in 5-bit two's complement)

2s Complement Addition/Subtraction

- Algorithm for addition, A + B:

1. Perform binary addition on the two numbers.
2. Ignore the carry out of the MSB (most significant bit).
3. Check for overflow: Overflow occurs if the 'carry in' and 'carry out' of the MSB are different, or if result is opposite sign of A and B.

- Algorithm for subtraction, A – B:

A – B = A + (-B)

1. Take 2s complement of B by inverting all the bits and adding 1.
2. Add the 2s complement of B to A.

2s Complement Addition/Subtraction

- Examples: 4-bit binary system

```
  +3      0011
+ +4    + 0100
----    -------
  +7      0111
----    -------
```

```
  -2      1110
+ -6    + 1010
----    -------
  -8     11000
----    -------
```

```
  +6      0110
+ -3    + 1101
----    -------
  +3     10011
----    -------
```

```
  +4      0100
+ -7    + 1001
----    -------
  -3      1101
----    -------
```

- Which of the above is/are overflow(s)?

Overflow

- If the numbers are unsigned, overflow occurs when there is a carry out of the most significant bit
- For signed numbers, overflow detected by comparing the sign of the result against the sign of the numbers
- If one number is positive and another is negative, overflow cannot occur
- If both are positive or both are negative, compare the carry into the sign bit and carry out of the sign bit
- If these two carries are not equal, then there is an overflow

Signed overflow

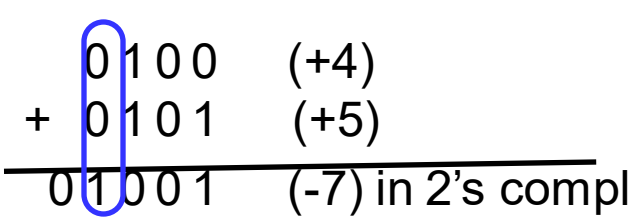
- With two's complement and a 4-bit adder, for example, the largest represent able decimal number is +7, and the smallest is -8.
- What if you try to compute 4 + 5, or (-4) + (-5)?

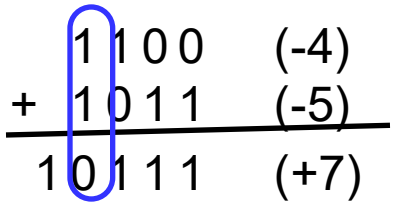
```
  0100   (+4)          1100  (-4)
+ 0101   (+5)        +  1011  (-5)
-------              ---------
 01001   (-7)          1 0111       (+7)
```

- We cannot just include the carry out to produce a five-digit result, as for unsigned addition. If we did, (-4) + (-5) would result in +23!
- Also, unlike the case with unsigned numbers, the carry out *cannot* be used to detect overflow.
  - In the example on the left, the carry out is 0 but there *is* overflow.
  - Conversely, there are situations where the carry out is 1 but there is *no* overflow.

Detecting signed overflow

- The easiest way to detect signed overflow is to look at all the sign bits.





- Overflow occurs only in the two situations above:
  - If you add two *positive* numbers and get a *negative* result.
  - If you add two *negative* numbers and get a *positive* result.
- Overflow cannot occur if you add a positive number to a negative number. Do you see why?

Signed Overflow

```
  1101 (-3)
+1011 (-5)
----------
  1000 (-8)
```

carry generated, but no overflow

```
    1
  1101 (-3)
+1010 (-6)
----------
1  0111 (+7)
```

carry and overflow

```
  1101 (-3)
+0010 (+2)
----------
0 1111 (-1)
```

no carry and no overflow

```
  0111 (+7)
+0011 (+3)
----------
0  1010 (-6)
```

no carry and overflow

Comparing the signed number systems

- Here are all the 4-bit numbers in the different systems.
- *Positive numbers are the same in all three representations.*
- Signed magnitude and one's complement have *two* ways of representing 0. This makes things more complicated.
- Two's complement has asymmetric ranges; there is one more negative number than positive number. Here, you can represent -8 but not +8.
- However, two's complement is preferred because it has only one 0, and its addition algorithm is the simplest.

Decimal	S.M.	1's comp.	2's comp.
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	1000	1111	—
-1	1001	1110	1111
-2	1010	1101	1110
-3	1011	1100	1101
-4	1100	1011	1100
-5	1101	1010	1011
-6	1110	1001	1010
-7	1111	1000	1001
-8	—	—	1000

Ranges of the signed number systems

- How many negative and positive numbers can be represented in each of the different systems on the previous page?

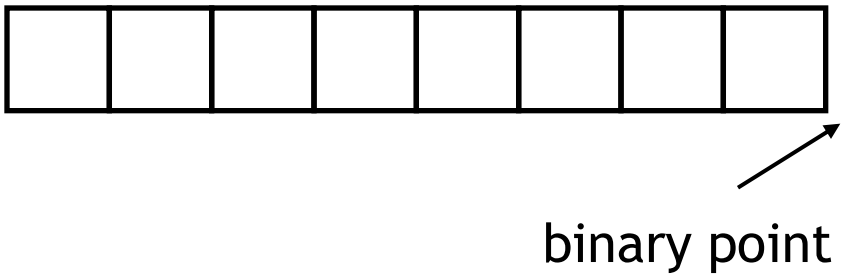
	Unsigned	Signed Magnitude	One's complement	Two's complement
Smallest	0000 (0)	1111 (-7)	1000 (-7)	1000 (-8)
Largest	1111 (15)	0111 (+7)	0111 (+7)	0111 (+7)

- In general, with n-bit numbers including the sign, the ranges are:

	Unsigned	Signed Magnitude	One's complement	Two's complement
Smallest	0	$-(2^{n-1}-1)$	$-(2^{n-1}-1)$	-2^{n-1}
Largest	2^n-1	$+(2^{n-1}-1)$	$+(2^{n-1}-1)$	$+(2^{n-1}-1)$

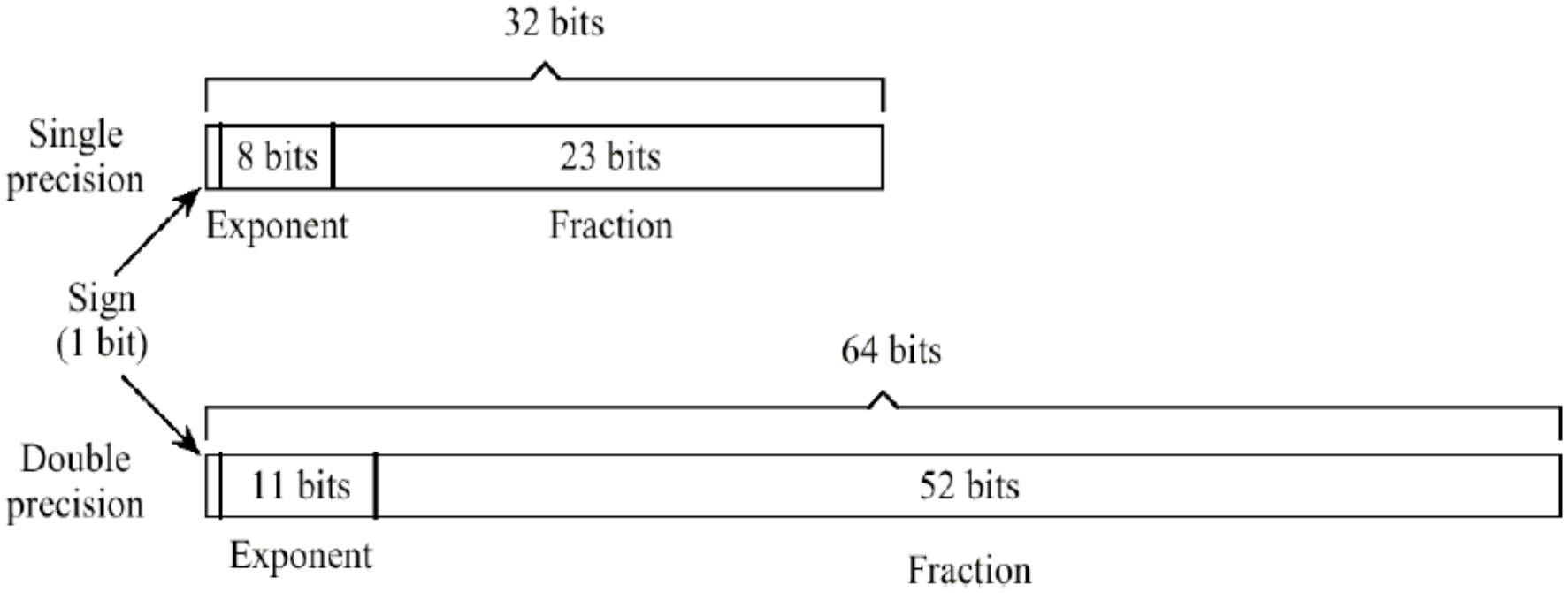
Fixed Point Numbers

- The signed and unsigned numbers representation given are fixed point numbers.
- The binary point is assumed to be at a fixed location, say, at the end of the number:

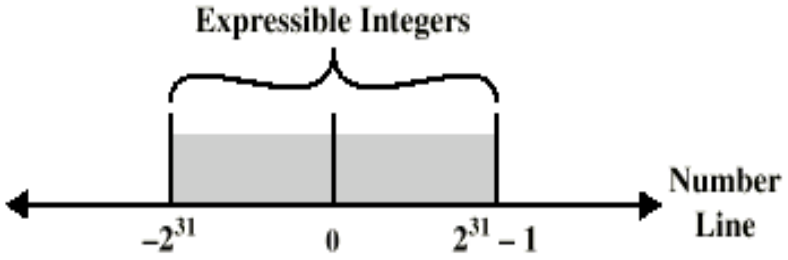


- Can represent all integers between –128 to 127 (for 8 bits).

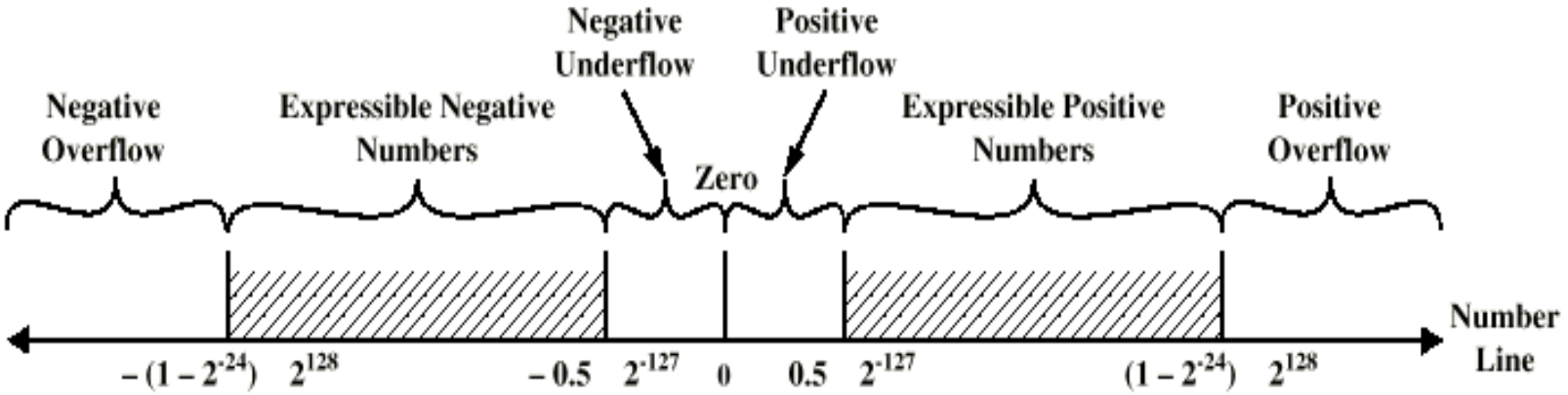
IEEE-754 Floating Point Formats



Expressible Numbers



(a) Twos Complement Integers



(b) Floating-Point Numbers

IEEE-754 Conversion Example

Represent -12.62510 in single precision IEEE-754 format.

- Step #1: Convert to target base. $-12.62510 = -1100.101_2$

- 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 significant is hidden, so final bit pattern is:

1 1000 0010 . 1001 0100 0000 0000 0000 000

Character Representation ASCII

ASCII (American Standard Code for Information Interchange) Code

MSB (3 bits)

LSB (4 bits)	0	1	2	3	4	5	6	7
0	NUL	DLE	SP	0	@	P	‘	P
1	SOH	DC1	!	1	A	Q	a	q
2	STX	DC2	“	2	B	R	b	r
3	ETX	DC3	#	3	C	S	c	s
4	EOT	DC4	$	4	D	T	d	t
5	ENQ	NAK	%	5	E	U	e	u
6	ACK	SYN	&	6	F	V	f	v
7	BEL	ETB	‘	7	G	W	g	w
8	BS	CAN	(	8	H	X	h	x
9	HT	EM	)	9	I	Y	I	y
A	LF	SUB	*	:	J	Z	j	z
B	VT	ESC	+	;	K	[	k	{
C	FF	FS	,	<	L	\	l	|
D	CR	GS	-	=	M	]	m	}
E	SO	RS	.	>	N	m	n	~
F	SI	US	/	?	O	n	o	DEL

Control Character Representation (ASCII)

NUL	Null	DC1	Device Control 1
SOH	Start of Heading (CC)	DC2	Device Control 2
STX	Start of Text (CC)	DC3	Device Control 3
ETX	End of Text (CC)	DC4	Device Control 4
EOT	End of Transmission (CC)	NAK	Negative Acknowledge (CC)
ENQ	Enquiry (CC)	SYN	Synchronous Idle (CC)
ACK	Acknowledge (CC)	ETB	End of Transmission Block (CC)
BEL	Bell	CAN	Cancel
BS	Backspace (FE)	EM	End of Medium
HT	Horizontal Tab. (FE)	SUB	Substitute
LF	Line Feed (FE)	ESC	Escape
VT	Vertical Tab. (FE)	FS	File Separator (IS)
FF	Form Feed (FE)	GS	Group Separator (IS)
CR	Carriage Return (FE)	RS	Record Separator (IS)
SO	Shift Out	US	Unit Separator (IS)
SI	Shift In	DEL	Delete
DLE	Data Link Escape (CC)		

(CC) Communication Control
(FE) Format Effector
(IS) Information Separator

The EBCDIC character code, shown with hexadecimal indices

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	

The EBCDIC control character representation

STX	Start of text	RS	Reader Stop	DC1	Device Control 1	BEL	Bell
DLE	Data Link Escape	PF	Punch Off	DC2	Device Control 2	SP	Space
BS	Backspace	DS	Digit Select	DC4	Device Control 4	IL	Idle
ACK	Acknowledge	PN	Punch On	CU1	Customer Use 1	NUL	Null
SOH	Start of Heading	SM	Set Mode	CU2	Customer Use 2		
ENQ	Enquiry	LC	Lower Case	CU3	Customer Use 3		
ESC	Escape	CC	Cursor Control	SYN	Synchronous Idle		
BYP	Bypass	CR	Carriage Return	IFS	Interchange File Separator		
CAN	Cancel	EM	End of Medium	EOT	End of Transmission		
RES	Restore	FF	Form Feed	ETB	End of Transmission Block		
SI	Shift In	TM	Tape Mark	NAK	Negative Acknowledge		
SO	Shift Out	UC	Upper Case	SMM	Start of Manual Message		
DEL	Delete	FS	Field Separator	SOS	Start of Significance		
SUB	Substitute	HT	Horizontal Tab	IGS	Interchange Group Separator		
NL	New Line	VT	Vertical Tab	IRS	Interchange Record Separator		
LF	Line Feed	UC	Upper Case	IUS	Interchange Unit Separator		

Pros and cons of integer representation

- Signed magnitude representation:
  - 2 representations for 0
  - Simple
  - 255 different numbers can be represented.
  - Need to consider both sign and magnitude in arithmetic
  - Different logic for addition and subtraction
- 1's complement representation:
  - 2 representations for 0
  - Complexity in performing addition and subtraction
  - 255 different numbers can be represented.
- 2's complement representation:
  - Only one representation for 0
  - 256 different numbers can be represented.
  - Arithmetic works easily
  - Negating is fairly easy

Reference

- Morris Mano, "Computer System Architecture", Pearson Education, 3rd edition (Chapter 3)

- William Stallings "Computer Organization and architecture", Prentice Hall, 7th edition,2006.(chapter 9)

- http://acad.intranet.vit.ac.in/CoursePage/COURSEPAGE-DUMP/EEE116%20Digital%20Logic%20System%20Design/Prof.%20Balamurugan%20%20S/Unit-I-Number%20system%20and%20Codes.ppt