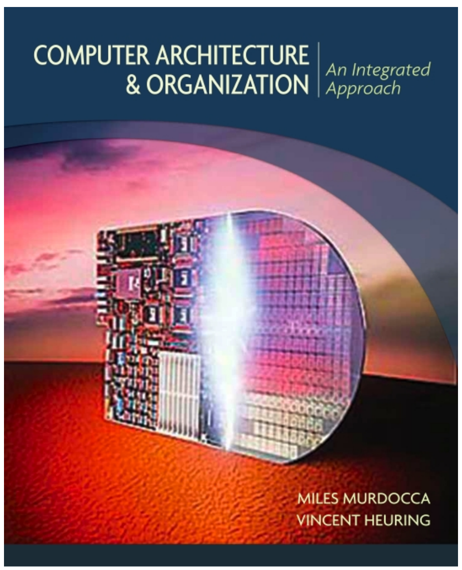
Computer Architecture and Organization

Miles Murdocca and Vincent Heuring



Chapter 2 – Data Representation

Chapter Contents

2.1 Fixed Point Numbers

2.2 Floating-Point Numbers

2.3 Case Study: Patriot Missile Defense Failure Caused by Loss of Precision

2.4 Character Codes

Fixed Point Numbers

- Using only two digits of precision for signed base 10 numbers, the *range* (interval between lowest and highest numbers) is [-99, +99] and the *precision* (distance between successive numbers) is 1.

- The maximum *error*, which is the difference between the value of a real number and the closest representable number, is 1/2 the precision. For this case, the error is $1/2 \times 1 = 0.5$.

- If we choose a = 70, b = 40, and c = -30, then a + (b + c) = 80 (which is correct) but (a + b) + c = -30 which is incorrect. The problem is that (a + b) is +110 for this example, which exceeds the range of +99, and so only the rightmost two digits (+10) are retained in the intermediate result. This is a problem that we need to keep in mind when representing real numbers in a finite representation.

Weighted Position Code

- The base, or radix of a number system defines the range of possible values that a digit may have: 0 – 9 for decimal; 0,1 for binary.

- The general form for determining the decimal value of a number is given by:

$$Value = \sum_{i=-m}^{n-1} b_i \cdot k^i$$

Example:

$541.25_{10} = 5 \times 10^2 + 4 \times 10^1 + 1 \times 10^0 + 2 \times 10^{-1} + 5 \times 10^{-2}$

$= (500)_{10} + (40)_{10} + (1)_{10} + (2/10)_{10} + (5/100)_{10}$

$= (541.25)_{10}$

Base Conversion with the Remainder Method

<u>Example:</u> Convert 23.375_{10} to base 2. Start by converting the integer portion:

		Integer	Remainder	
23/2	=	11	1	← Least significant bit
11/2	=	5	1	
5/2	=	2	1	
2/2	=	1	0	
1/2	=	0	1	← Most significant bit

$$(23)_{10} = (10111)_2$$

Base Conversion with the Multiplication Method

- Now, convert the fraction:

Most significant bit

$$.375 \times 2 = 0.75$$

$$.75 \times 2 = 1.5$$

$$.5 \times 2 = 1.0$$

Least significant bit

$$(.375)_{10} = (.011)_2$$

- Putting it all together, $23.375_{10} = 10111.011_2$

Nonterminating Base 2 Fraction

- We can't always convert a terminating base 10 fraction into an equivalent terminating base 2 fraction:

$$.2 \times 2 = 0.4$$
$$.4 \times 2 = 0.8$$
$$.8 \times 2 = 1.6$$
$$.6 \times 2 = 1.2$$
$$.2 \times 2 = 0.4$$
$$\vdots$$

Base 2, 8, 10, 16 Number Systems

Example: Show a column for ternary (base 3). As an extension of that, convert 14_{10} to base 3, using 3 as the divisor for the remainder method (instead of 2). Result is 112_3

Binary (base 2)	Octal (base 8)	Decimal (base 10)	Hexadecimal (base 16)
0	0	0	0
1	1	1	1
10	2	2	2
11	3	3	3
100	4	4	4
101	5	5	5
110	6	6	6
111	7	7	7
1000	10	8	8
1001	11	9	9
1010	12	10	A
1011	13	11	B
1100	14	12	C
1101	15	13	D
1110	16	14	E
1111	17	15	F

More on Base Conversions

- Converting among power-of-2 bases is particularly simple:

$$1011_2 = (10_2)(11_2) = 23_4$$

$$23_4 = (2_4)(3_4) = (10_2)(11_2) = 1011_2$$

$$101010_2 = (101_2)(010_2) = 52_8$$

$$01101101_2 = (0110_2)(1101_2) = 6D_{16}$$

- How many bits should be used for each base 4, 8, *etc.*, digit? For base 2, in which $2 = 2^1$, the exponent is 1 and so one bit is used for each base 2 digit. For base 4, in which $4 = 2^2$, the exponent is 2, so so two bits are used for each base 4 digit. Likewise, for base 8 and base 16, $8 = 2^3$ and $16 = 2^4$, and so 3 bits and 4 bits are used for base 8 and base 16 digits, respectively.

Binary Addition

- This simple binary addition example provides background for the signed number representations to follow.

Carry in →	0	0	0	0	1	1	1	1
Operands →	0	0	1	1	0	0	1	1
→	+ 0	+ 1	+ 0	+ 1	+ 0	+ 1	+ 0	+ 1
	0 0	0 1	0 1	1 0	0 1	1 0	1 0	1 1

Carry out ↑ ↑ Sum

Example:

Carry		1	1	1	1	0	0	0	0	
Addend: *A*		0	1	1	1	1	1	0	0	$(124)_{10}$
Augend: *B*	+	0	1	0	1	1	0	1	0	$(90)_{10}$
Sum		1	1	0	1	0	1	1	0	$(214)_{10}$

Signed Fixed Point Numbers

- For an 8-bit number, there are $2^8 = 256$ possible bit patterns. These bit patterns can represent negative numbers if we choose to assign bit patterns to numbers in this way. We can assign half of the bit patterns to negative numbers and half of the bit patterns to positive numbers.
- Four signed representations we will cover are:

Signed Magnitude

One's Complement

Two's Complement

Excess (Biased)

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

Signed Magnitude

- Also know as "sign and magnitude," the leftmost bit is the sign (0 = positive, 1 = negative) and the remaining bits are the magnitude.

- <u>Example:</u>

$+25_{10} = 00011001_2$

$-25_{10} = 10011001_2$

- Two representations for zero: $+0 = 00000000_2$, $-0 = 10000000_2$.
- Largest number is +127, smallest number is -127_{10}, using an 8-bit representation.

One's Complement

- The leftmost bit is the sign (0 = positive, 1 = negative). Negative of a number is obtained by subtracting each bit from 2 (essentially, *complementing* each bit from 0 to 1 or from 1 to 0). This goes both ways: converting positive numbers to negative numbers, and converting negative numbers to positive numbers.

- <u>Example:</u>

$+25_{10} = 00011001_2$

$-25_{10} = 11100110_2$

- Two representations for zero: $+0 = 00000000_2$, $-0 = 11111111_2$.

- Largest number is $+127_{10}$, smallest number is -127_{10}, using an 8-bit representation.

Two's Complement

- The leftmost bit is the sign (0 = positive, 1 = negative). Negative of a number is obtained by adding 1 to the one's complement negative. This goes both ways, converting between positive and negative numbers.

- Example (recall that -25_{10} in one's complement is 11100110_2):

$+25_{10} = 00011001_2$

$-25_{10} = 11100111_2$

- One representation for zero: $+0 = 00000000_2$, $-0 = 00000000_2$.
- Largest number is $+127_{10}$, smallest number is -128_{10}, using an 8-bit representation.

Excess (Biased)

- The leftmost bit is the sign (usually 1 = positive, 0 = negative). Positive and 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.

- <u>Example</u> (excess 128 "adds" 128 to the two's complement version, ignoring any carry out of the most significant bit) :

$+12_{10} = 10001100_2$

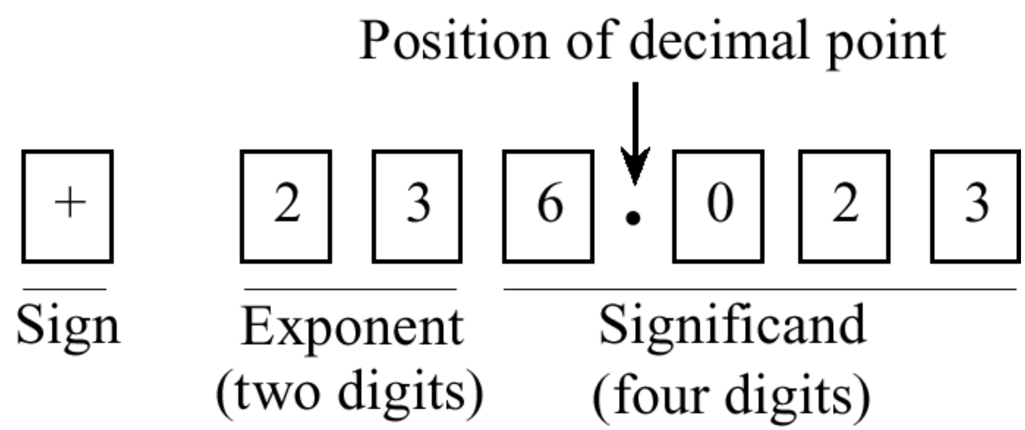
$-12_{10} = 01110100_2$

- One representation for zero: $+0 = 10000000_2$, $-0 = 10000000_2$.

- Largest number is $+127_{10}$, smallest number is -128_{10}, using an 8-bit representation.

Base 10 Floating Point Numbers

- Floating point numbers allow very large and very small numbers to be represented using only a 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 ($+6.023 \times 10^{23}$):



Normalization

- The base 10 number 254 can be represented in floating point form as 254×10^0, or equivalently as:

  25.4×10^1, or 2.54×10^2, or

  $.254 \times 10^3$, or $.0254 \times 10^4$, or

infinitely many other ways, which creates problems when making comparisons, with so many representations of the same number.

- Floating point numbers are usually *normalized*, in which the radix point is located in only one possible position for a given number.
- Usually, but not always, the normalized representation places the radix point immediately to the left of the leftmost, nonzero digit in the fraction, as in: $.254 \times 10^3$.

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.

  $.254 \times 10^3 = 254_{10}$. Using the remainder method, we find that $254_{10} = 376 \times 8^0$:

  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

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, we would like to characterize this floating point representation in terms of the largest positive representable number, the smallest (nonzero) positive representable number, the smallest gap between two successive numbers, the largest gap between two successive numbers, and the total number of numbers that can be represented.

Error, Range, and Precision (cont’)

- 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: $b^M \times b^{-s} = 8^{3 - 4} = 8^{-1}$

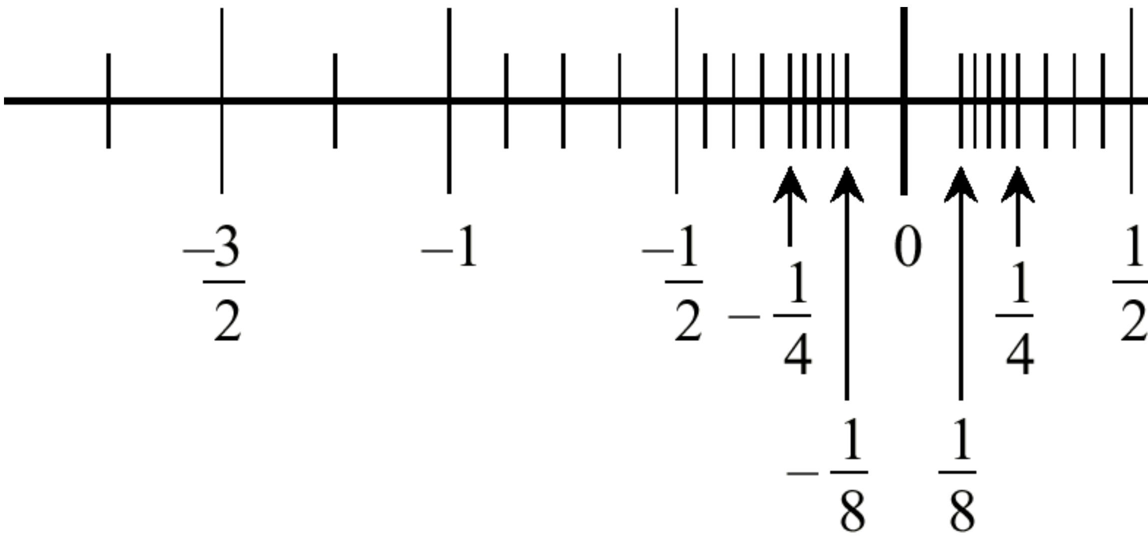
- Smallest gap: $b^m \times b^{-s} = 8^{-4 - 4} = 8^{-8}$

Error, Range, and Precision (cont')

$$\overset{\text{A}}{\underset{\text{Sign bit}}{2}} \times \overset{\text{B}}{\underbrace{((M - m) + 1)}_{\text{The number of exponents}}} \times \overset{\text{C}}{\underbrace{(b - 1)}_{\text{First digit of fraction}}} \times \overset{\text{D}}{\underbrace{b^{s-1}}_{\text{Remaining digits of fraction}}} + \overset{\text{E}}{\underset{\text{Zero}}{1}}$$

- Number of representable numbers: There are 5 components: (A) sign bit; for each number except 0 for this case, there is both a positive and negative version; (B) $(M - m) + 1$ exponents; (C) b - 1 values for the first digit (0 is disallowed for the first normalized digit); (D) b^{s-1} values for each of the s-1 remaining digits, plus (E) a special representation for 0. For this example, 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 in 16 bits, which is 2^{16}.

Example Floating Point Format



$$b = 2 \qquad M = +1$$
$$s = 3 \qquad m = -2$$

- 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.

Gap Size Follows Exponent Size

- The relative error is approximately the same for all numbers.
- If we take the ratio of a large gap to a large number, and compare that to the ratio of a small gap to a small number, then the ratios are the same:

$$\text{A large gap} \longrightarrow \qquad \frac{b^{M-s}}{b^M \times (1 - b^{-s})} = \frac{b^{-s}}{1 - b^{-s}} = \frac{1}{b^s - 1}$$
$$\text{A large number} \longrightarrow$$

$$\text{A small gap} \longrightarrow \qquad \frac{b^{m-s}}{b^m \times (1 - b^{-s})} = \frac{b^{-s}}{1 - b^{-s}} = \frac{1}{b^s - 1}$$
$$\text{A small number} \longrightarrow$$

Conversion Example

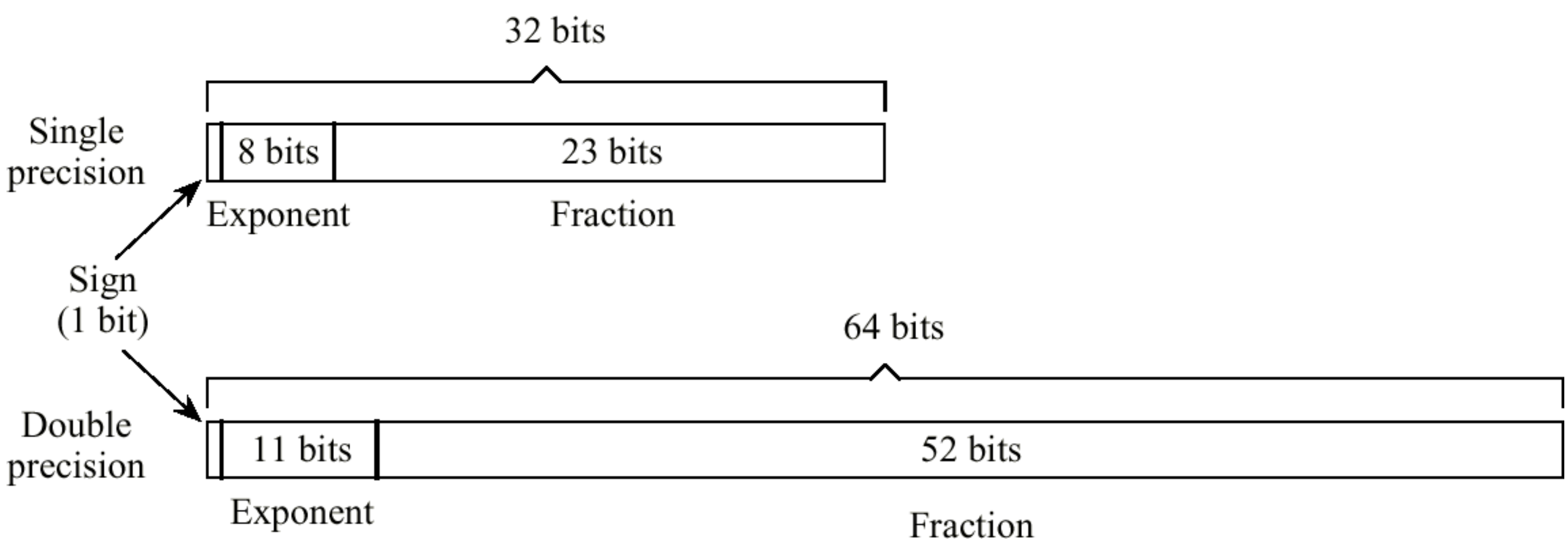
- <u>Example:</u> Convert $(9.375 \times 10^{-2})_{10}$ to base 2 scientific notation
- Start by converting from base 10 floating point to base 10 fixed point by moving the decimal point two positions to the left, which corresponds to the -2 exponent: .09375.
- Next, convert from base 10 fixed point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

$$.09375 \times 2 = 0.1875$$
$$.1875 \times 2 = 0.375$$
$$.375 \times 2 = 0.75$$
$$.75 \times 2 = 1.5$$
$$.5 \times 2 = 1.0$$

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

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

IEEE-754 Floating Point Formats



IEEE-754 Examples

	Value	Bit Pattern		
		Sign	Exponent	Fraction
(a)	$+1.101 \times 2^{5}$	0	1000 0100	101 0000 0000 0000 0000 0000
(b)	-1.01011×2^{-126}	1	0000 0001	010 1100 0000 0000 0000 0000
(c)	$+1.0 \times 2^{127}$	0	1111 1110	000 0000 0000 0000 0000 0000
(d)	$+0$	0	0000 0000	000 0000 0000 0000 0000 0000
(e)	-0	1	0000 0000	000 0000 0000 0000 0000 0000
(f)	$+\infty$	0	1111 1111	000 0000 0000 0000 0000 0000
(g)	$+2^{-128}$	0	0000 0000	010 0000 0000 0000 0000 0000
(h)	$+\mathrm{NaN}$	0	1111 1111	011 0111 0000 0000 0000 0000
(i)	$+2^{-128}$	0	011 0111 1111	0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000

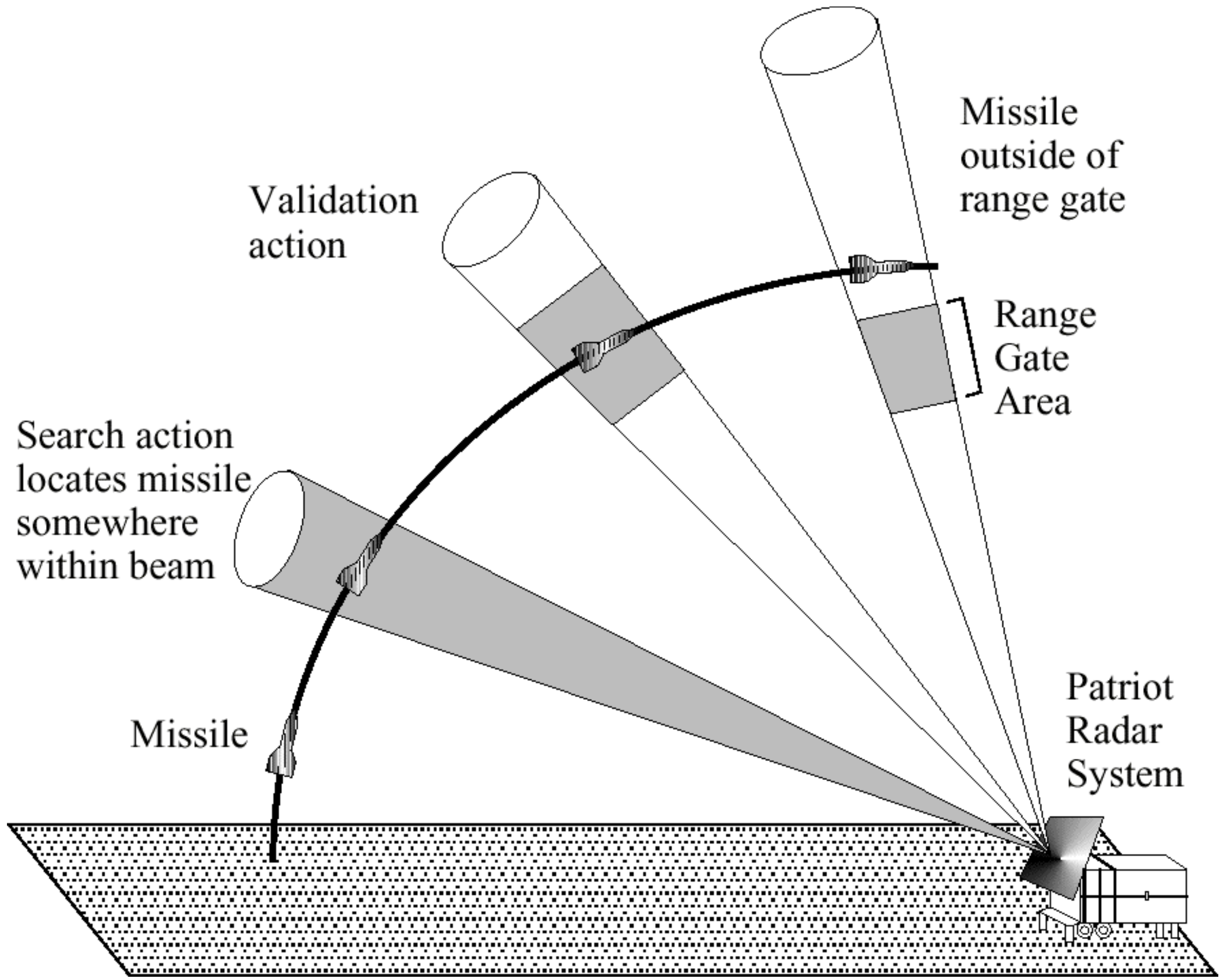
IEEE-754 Conversion Example

- Represent -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

Effect of Loss of Precision

- According to the General Accounting Office of the U.S. Government, a loss of precision in converting 24-bit integers into 24-bit floating point numbers was responsible for the failure of a Patriot anti-missile battery.



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}. Thus “a” is at $41_{16} + 20_{16} = 61_{16}$.
- The character “5” at position 35_{16} is different than the number 5. To convert character-numbers into number-numbers, subtract 30_{16}: $35_{16} - 30_{16} = 5$.

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

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

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	Đ	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