Number Representation

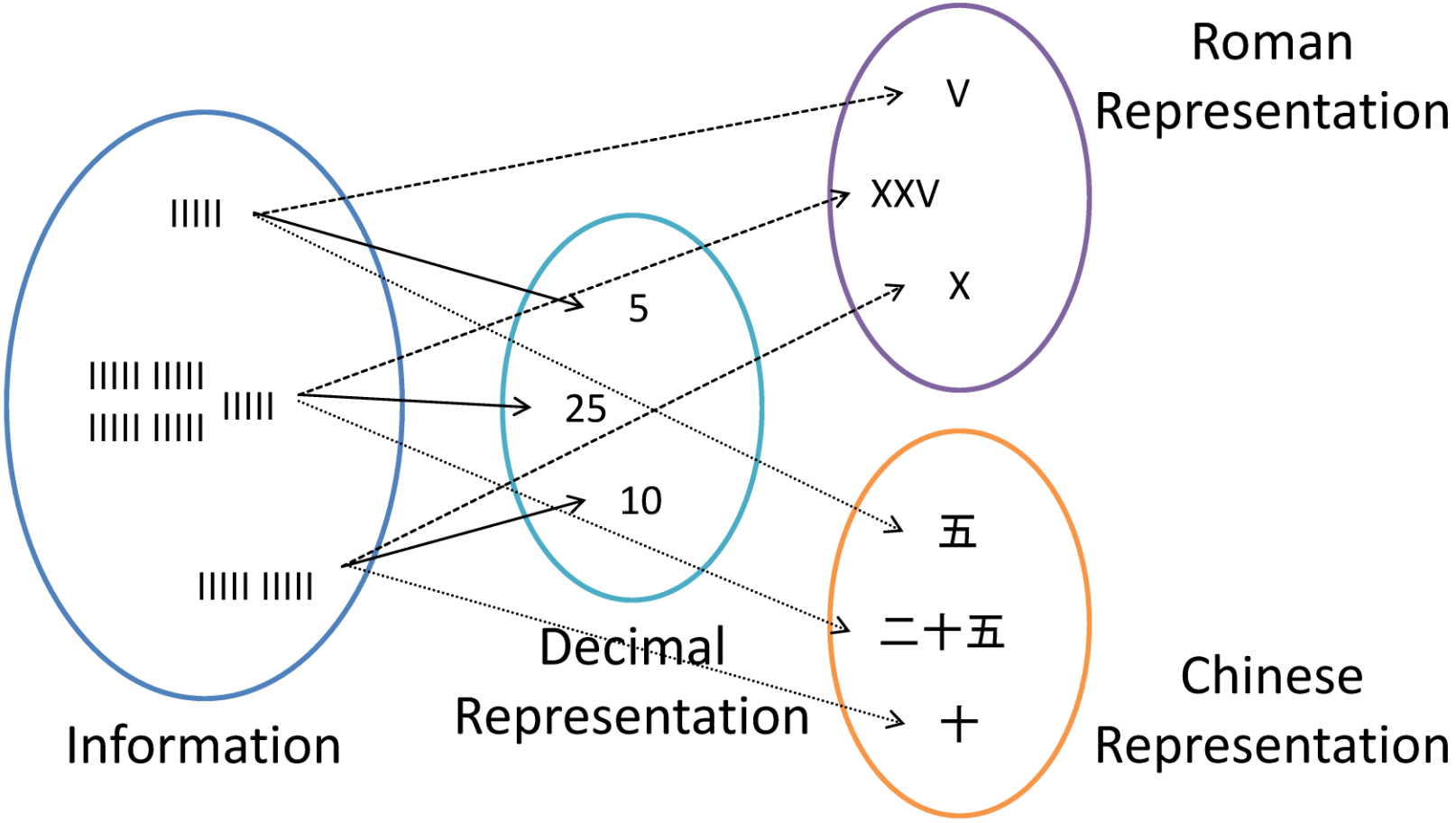
Networks and Embedded Software

Module 5.4.2

by Wolfgang Neff

Numeral Systems (1)

- Numeral systems encode numbers



Numeral Systems (2)

- Example: Decimal System
  - Base: 10 (number of digits)
  - Digits: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9
  - The value of a digit depends on its position
    - Example
      - $2012_{dec} = 2 \cdot 10^3 + 0 \cdot 10^2 + 1 \cdot 10^1 + 2 \cdot 10^0$
        $2012_{dec} = 2 \cdot 1000 + 0 \cdot 100 + 1 \cdot 10 + 2 \cdot 1$
        $2012_{dec} = 2012_{val}$ (value)

Numeral Systems (3)

- Values of numeral representation in general

$$a_n a_{n-1} \cdots a_1 a_0 = \sum_{i=0}^{n} a_i \cdot b^i$$

  - a_i: the digit at position *i*
  - *b*: the number of digits
  - Example
    - $2012_{dec} = \sum_{i=0}^{3} a_i \cdot 10^i = 2 \cdot 10^3 + 0 \cdot 10^2 + 1 \cdot 10^1 + 2 \cdot 10^0$

Numeral Systems (4)

- Basic concepts
  - Base: number of digits
    - 10 (we use the decimal system as example)
  - Digits: representation of the digits
    - 0,1,2,3,4,5,6,7,8,9
  - Value of a representation
    - $2012_{dec} = 2 \cdot 10^3 + 0 \cdot 10^2 + 1 \cdot 10^1 + 2 \cdot 10^0 = 2012_{val}$
  - Range: number of possible values = $b^{\text{length of number}}$
    - 4 digits (0 … 9999) = 10^4 different values

Numeral Systems (5)

- Basic concepts (continued)
  - Representation
    - 2012_{val} = 2012 ÷ 10 = 201 remainder 2
      201 ÷ 10 = 20 remainder 1
      20 ÷ 10 = 2 remainder 0
      2 ÷ 10 = 0 remainder 2

Reading Direction

The algorithm ends here

    - A value of 2012 is represented by 2012_{dec}

Numeral Systems (6)

- The hexadecimal system
  - Base: 16
  - Digits: 0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F
  - Value
    - $2012_{hex} = 2 \cdot 16^3 + 0 \cdot 16^2 + 1 \cdot 16^1 + 2 \cdot 16^0$
      $2 \cdot 4096 + 0 \cdot 256 + 1 \cdot 16 + 2 \cdot 1$
      8210_{val}
  - Range
    - 4 digits (0 … $FFFF_{hex}$) = 16^4 (65536) different values

Numeral Systems (7)

- The hexadecimal system (continued)
  - Representation
    - 2012_{val} = 2012 ÷ 16 = 125 remainder 12 (C)
      125 ÷ 16 = 7 remainder 13 (D)
      7 ÷ 16 = 0 remainder 7
    - A value of 2012 is represented by $7DC_{hex}$

Numeral Systems (8)

- The binary system
  - Base: 2
  - Digits: 0,1
  - Value
    - $1011_{bin} = 1 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0$
      $1 \cdot 8 + 0 \cdot 4 + 1 \cdot 2 + 1 \cdot 1$
      11_{val}
  - Range
    - 4 digits ($0 \ldots 1111_{bin}$) = 2^4 (16) different values

Numeral Systems (9)

- The binary system (continued)
  - Representation
    - 57_{val} =

$$
\begin{aligned}
57 \div 2 &= 28 \text{ remainder } 1 \\
28 \div 2 &= 14 \text{ remainder } 0 \\
14 \div 2 &= 7 \text{ remainder } 0 \\
7 \div 2 &= 3 \text{ remainder } 1 \\
3 \div 2 &= 1 \text{ remainder } 1 \\
1 \div 2 &= 0 \text{ remainder } 1
\end{aligned}
$$

    - A value of 57 is represented by 111001_{bin}

Binary Numbers (1)

- Computers use the binary system
  - Because they are digital circuits
  - Therefore they are based on Boolean algebra
- The binary digits are called bits
- Some groups of bits have special names
  - Nibble (4 bits), byte (8 bits), word (16 bits or more)
- Data is encoded as binary numbers
  - Numbers, text, images etc.

Binary Numbers (2)

- Fixed-length binary numbers
  - The length of binary numbers is limited
  - The first and the last digit have a name
    - Rightmost bit: LSB (Least Significant Bit $2^0=1$)
    - Leftmost bit: MSB (Most Significant Bit $2^7=128$)

Bit	7	6	5	4	3	2	1	0
Value	128	64	32	16	8	4	2	1
Number	1	1	1	0	1	0	0	1

MSB

LSB

Binary Numbers (3)

- Use exponential calculus to get the range
  - 8 Bits: $2^8 = 256$ values → Range is 0 … 255
  - 16 Bits: $2^{16} = 2^{10+6} = 2^{10} \cdot 2^6 \approx 1000*64 \approx 64000$
    $2^{16} = 65536$ → Range is 0 … 65535
  - 32 Bits: $2^{32} = 2^{10+10+10+2} = 2^{10} \cdot 2^{10} \cdot 2^{10} \cdot 2^2 \approx$
    $1000*1000*1000*4 \approx 4{,}000{,}000{,}000$
    $2^{32} = 4{,}294{,}967{,}296$

Binary Numbers (4)

- Nibbles (4 bits) can easily converted to hex

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

- Examples
  - $0000\ 0000_{bin} \leftrightarrow 00_{hex}$
  - $0110\ 0100_{bin} \leftrightarrow 64_{hex}$
  - $1011\ 1100_{bin} \leftrightarrow BC_{hex}$
  - $1111\ 1111_{bin} \leftrightarrow FF_{hex}$

Binary Numbers (5)

- Rapid conversion Bin → Dec
  - Write 1 above the LSB.
  - Double until MSB is reached.
  - Add all values where the digit is 1.
- Example:

128	64	32	16	8	4	2	1
1	0	1	1	0	0	1	1

  - $1011\ 0011_{bin} = 128+32+16+2+1 = 179_{dec}$

Binary Numbers (6)

- Rapid Conversion Dec → Bin
  - Double 1 until it is greater than the number to convert.
  - Try to subtract these values. If it is possible the digit is 1 otherwise it is 0.
- Example: $179_{dec} = 10110011_{bin}$

179	179	51	51	19	3	3	3	1
256	128	64	32	16	8	4	2	1
0	1	0	1	1	0	0	1	1

Calculus (1)

- Binary Addition
  - Same method as decimal addition
  - Noteworthy facts:
    - Leading Zeros
    - Carry
    - Overflow

1st Number	1	9	4	5
2nd Number	0	5	1	2
Carry	1	0	0	0
Result	2	4	5	7

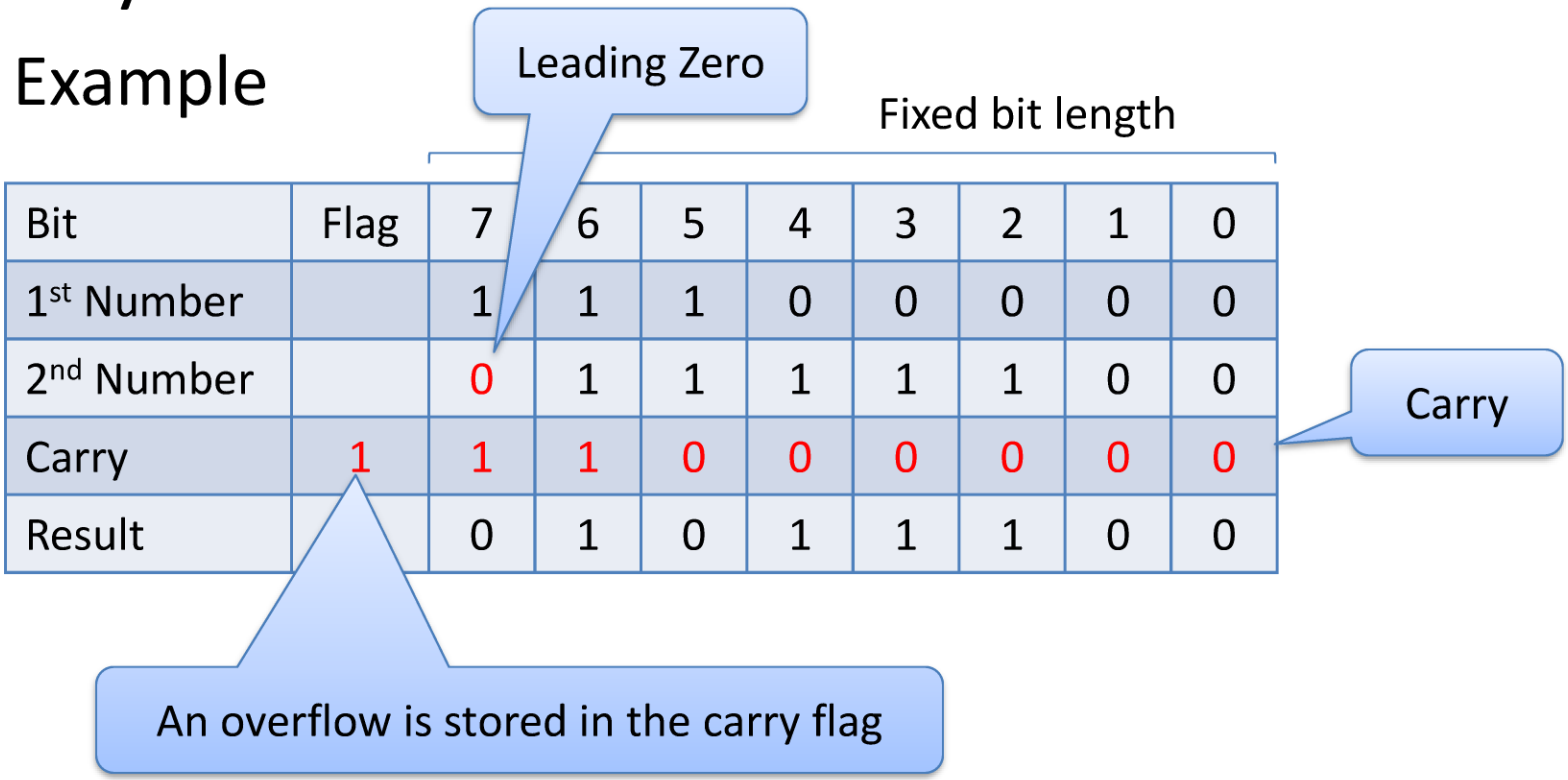
Carry

Leading Zero

No Overflow

Calculus (2)

- Binary Addition
  - Example



Bit	Flag	7	6	5	4	3	2	1	0
1st Number		1	1	1	0	0	0	0	0
2nd Number		0	1	1	1	1	1	0	0
Carry	1	1	1	0	0	0	0	0	0
Result		0	1	0	1	1	1	0	0

Calculus (3)

- Binary Addition
  - We encounter the following situations

1st Bit		0		0		1		1		0		0		1	
2nd Bit		0		1		0		1		0		1		0	
Carry		0		0		0		0		1		1		1	
Result	dec	0		1		1		2		1		2		2	
	bin	0	0	0	1	0	1	1	0	0	1	1	0	1	0	1	1

Calculus (4)

- Binary Subtraction
  - Subtraction works with borrowing

Borrowed from next bit

1st Bit	bin	0	0	0	1	1	0	0	1	1	0	0	1	1	0	1	1
	dec	0		1		2		1		2		1		2		3	
2nd Bit		0		0		1		1		0		0		1		1	
Borrow		0		0		0		0		1		1		1		1	
Result		0		1		1		0		1		0		0		1	

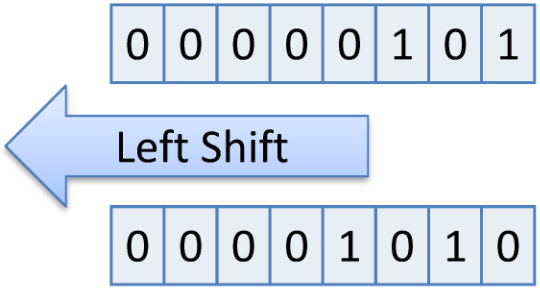
Calculus (5)

- Binary Multiplication
  - Same method as decimal multiplication
  - Example: 13*5 = 65 = 1000001_{bin}

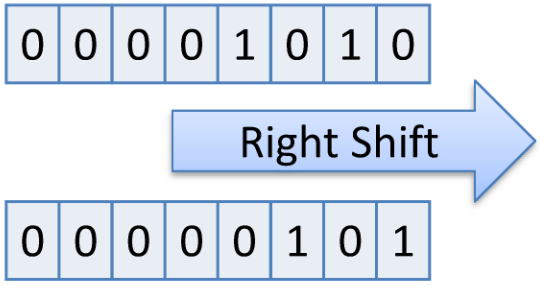
		1	1	0	1			*	1	0	1
		1	1	0	1						
+			0	0	0	0					
+				1	1	0	1				
	1	0	0	0	0	0	1				

Calculus (6)

- Shifting
  - A left shift doubles the value of the number
    - 5*2 = 10



  - A right shift halves the value of the number
    - 10 / 2 = 5

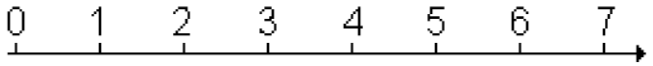


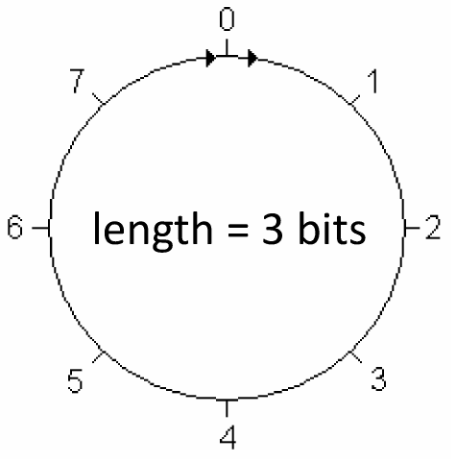
Calculus (6)

- Binary Division
  - Same method as decimal division.
  - Sometimes replaced by shifting because of speed.
    - Example: 2,54*15 = 254*15/100 = 38
      - Multiply with an appropriate power of two (2^7=128)
        2,54 = 2,54*2^7 / 2^7 = 325 / 2^7
      - Multiply with numerator and then shift
        2,54*15 = 325*15 / 2^7 = 4875 / 2^7 = 4875 >> 7 = 38
      - This is only an approximation but may be very fast
        - » 254/100 = 2,540 vs. 325/128 = 2,539
        - » E. g. AVR: with division 250 cycles vs. 16 cycles with shift

Signed Numbers (1)

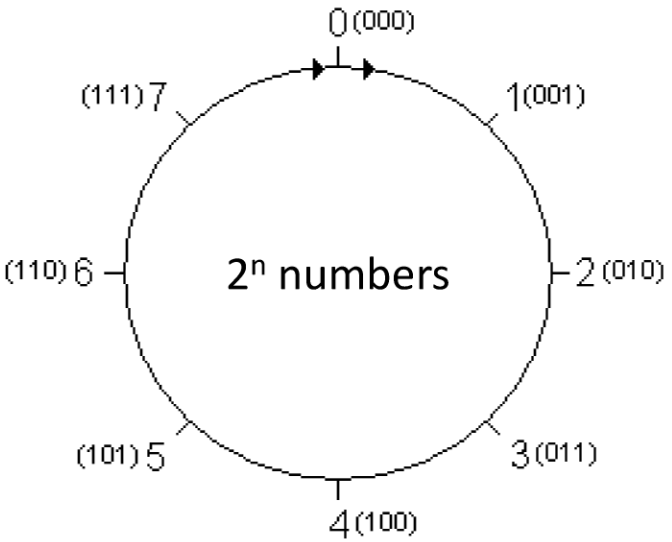
- Fixed-length binary numbers have a range.
- If we exceed the range we restart from zero.
- Therefore the number ray becomes a circle.

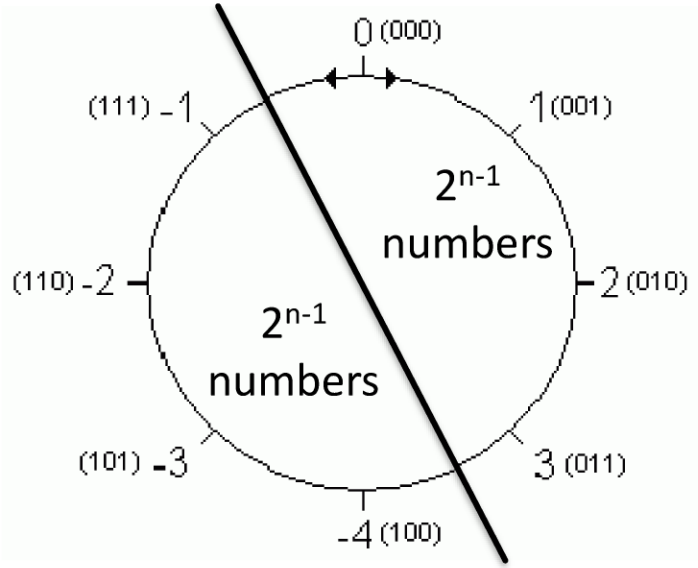




Signed Numbers (2)

- We get negative numbers if we go beyond zero.





Signed Numbers (3)

- We have to sacrifice positive numbers to get negative numbers.
- Negative numbers have its MSB is set.
- The range of negative number is
  - Positive number range: $0...2^{n-1}-1$
  - Negative number range: $-1...-2^{n-1}$

    n: bit length of number

Signed Numbers (4)

- Two's complement
  - Used to negate a number
    - Complement all bits
    - Add a value of one

Number	0	0	0	1	1	0	0	0
Complement	1	1	1	0	0	1	1	1
+1	0	0	0	0	0	0	0	1
Carry	0	0	0	0	1	1	1	0
Result	1	1	1	0	1	0	0	0

Signed Numbers (5)

- Rapid conversion Bin → signed Dec
  - Negate the MSB
  - Add the rest
- Example: $B3_{hex} = -77_{dec}$

128	64	32	16	8	4	2	1
1	0	1	1	0	0	1	1

  - $10110011_{bin} = -128+32+16+2+1 = -77_{dec}$

Signed Numbers (6)

- Rapid conversion negative Dec → Bin
  - Add number to range
  - Convert result to binary
- Example: $-77_{dec} = B3_{hex}$
  - Bit length = 8 → Range: $2^8 = 256$
  - $256 - 77 = 179$
  - $179_{dec} = 10110011_{bin} = B3_{hex}$

Fixed-Point Numbers (1)

- Fixed-point numbers are numeral systems with a negative index:

$$a_n a_{n-1} \cdots a_1 a_0 . a_{-1} \cdots a_{-m} = \sum_{i=-m}^{n} a_i \cdot b^i$$

  - There are *m* positions after the decimal point
  - There are *n+1* positions before the decimal point

Fixed-Point Numbers (2)

- Example: a base 16 fixed-point number
  - Number: 20.12_{hex}
  - Value:
    - $20.12_{hex} = 2 \cdot 16^{1} + 0 \cdot 16^{0} + 1 \cdot 16^{-1} + 2 \cdot 16^{-2}$
      $2 \cdot 16 + 0 \cdot 1 + 1 \cdot {}^{1}/_{16} + 2 \cdot {}^{1}/_{256}$
      $2 \cdot 16 + 0 \cdot 1 + 1 \cdot 0.0625 + 2 \cdot 0.00390625$
      32.0703125
  - You have to round to preserve the number of positions after the decimal point.

Fixed-Point Numbers (2)

- Example: a base 16 fixed-point number
  - Representation
    - 32.07 = 32 + 0.07
    - 32 = 20_{hex}

      0.07 = $0.07 \cdot 16 = 1$ remainder 0.12

      $0.12 \cdot 16 = 1$ remainder 0.92

      $0.92 \cdot 16 = 14$ remainder 0.72

      $0.72 \cdot 16 = 11$ remainder 0.52

      Division by $^1/_{16}$

      Reading Direction
    - Stop when you have got enough positions
    - The value of 32.07 is $20.11EB\ldots_{hex}$
    - Round-off errors are almost inevitable

Fixed-Point Numbers (3)

- Binary fixed-point numbers
  - Notation of binary fractional numbers
    - s(n.m): signed binary fractional - n bits before the decimal point and m bits after. MSB is sign bit.
    - u(n.m): unsigned binary fractional – n bits before the decimal point and m bits after it. No sign bit.
  - Binary fractional calculus
    - d(n.m) + d(n.m) = d(n+1.m) = d(n.m) and carry flag
    - $d(n_1.m_1) \cdot d(n_2.m_2) = d(n_1+n_2.m_1+m_2)$

Fixed-Point Numbers (4)

- d(1.n) binary fractional numbers
  - often used in digital signal processing (DSP)
  - Multiplication modifies the number format
    - $d(1.n) \cdot d(1.n) = d(2.2 \cdot n)$
    - A left shift is necessary to obtain $d(1.2 \cdot n+1)$
    - An overflow might occur indicated by the carry flag
  - Some microcontroller have special instructions
    - AVR: FMUL – Fractional Multiply Unsigned
      $s(1.7) \cdot s(1.7) \rightarrow s(1.15)$

Fixed-Point Numbers (5)

- Rapid conversion d(n,m) → Dec
  - Start with 1 at the decimal point.
  - Double it until you reach the MSB.
  - Half it until you reach the LSB.
- Example: $10110011_{s(1.7)} = B3_{s(1.7)}$

-1	$^1/_2$	$^1/_4$	$^1/_8$	$^1/_{16}$	$^1/_{32}$	$^1/_{64}$	$^1/_{128}$
1	0	1	1	0	0	1	1

  - $B3_{s(1.7)} = -1 + {}^1/_4 + {}^1/_8 + {}^1/_{64} + {}^1/_{128} = -0.6015625_{dec}$

Fixed-Point Numbers (6)

- Rapid Conversion Dec → Bin
  - Choose a binary fractional format and write its values above the positions.
  - Try to subtract these values. If it is possible then the digit is 1 otherwise it is 0.
- Example: $-0.60_{dec} = 10110011_{s(1.7)}$

-0.6	0.4	0.4	0.15	0.025	0.025	0.025	0.009375
-1	$^1/_2$	$^1/_4$	$^1/_8$	$^1/_{16}$	$^1/_{32}$	$^1/_{64}$	$^1/_{128}$
1	0	1	1	0	0	1	1

Fixed-Point Numbers (7)

- Example: sin(0…2π) in s(1.7) fractional binaries

n	0	1	2	3	4	5	6	7
x	0,00	0.039	0.79	1.18	1.157	1.196	2.36	2.75
sin(x)	0.00	0.38	0.71	0.92	1.00	0.92	0.71	0.38
s(1.7)	00_{hex}	30_{hex}	$5A_{hex}$	76_{hex}	$7F_{hex}$	76_{hex}	$5A_{hex}$	30_{hex}

1.0 is not in the range of s(1.7). We use 7E instead. This is called saturation.

n	8	9	10	11	12	13	14	15
x	3.14	3.53	3.93	4.32	4.71	5.11	5.50	5.89
sin(x)	0.00	-0.38	-0.71	-0.92	-1.00	-0.92	-0.71	-0.38
s(1.7)	00_{hex}	CF_{hex}	$A5_{hex}$	89_{hex}	80_{hex}	89_{hex}	$A5_{hex}$	CF_{hex}

Floating-Point Numbers (1)

- Do not have a fixed decimal point position.
- Are expressed by a mantissa and an exponent.
- Are usually normalized:
  - Only one digit before the decimal point
  - $20.12_{dec} \rightarrow 2.012_{dec} \cdot 10^{1}$ (normalized)
- Floating-point numbers are standardized:
  - IEEE 754: Standard for Floating-Point Arithmetic
  - Commonly used: single and double precision

Floating-Point Numbers (2)

- IEEE 754 floating-point binary numbers
  - Format of a half precision floating-point

1st Byte								2nd Byte							
S	C	C	C	C	C	F	F	F	F	F	F	F	F	F	F
Sign	Character							Fraction							

  - Value of a half precision floating-point
    $= (-1)^S \cdot \text{Mantissa} \cdot 2^{\text{Exponent}}$
    - Bias = 15
    - Exponent = Character – Bias (-14 … +15)
    - Mantissa = $1.\text{Fraction}_{bin}$

Character 00000 and 11111 reserved for ±∞ and NaN (Not a Number)

Floating-Point Numbers (3)

- Example: value of half precision $B248_{hex}$

1st Byte								2nd Byte							
S	C	C	C	C	C	F	F	F	F	F	F	F	F	F	F
1	1	0	0	0	0	1	0	0	1	0	0	1	0	0	0

  - Value:
    - Exponent = Character - Bias = 16 – 15 = 1
    - Mantissa = 1. + Fraction = 1.1001001000 = 1,5703125
    - Value = $(-1)^S \cdot \text{Mantissa} \cdot 2^{\text{Exponent}} = -1 \cdot 1{,}141571 \cdot 2$
    - Half precision $B248_{hex} = -3.140625_{dec}$

Floating-Point Numbers (4)

- Example: π as a half precision binary number
  - Representation
    - 3.14159 = 11.00100100001111_{bin}
      $1.100100100001111_{bin} \cdot 2^1$
    - Character = Exponent + Bias = 1 + 15 = 16 = 10000_{bin}
    - Fraction = 1001001000_{bin}

1st Byte								2nd Byte							
S	C	C	C	C	C	F	F	F	F	F	F	F	F	F	F
0	1	0	0	0	0	1	0	0	1	0	0	1	0	0	0

    - π = 4248_{hex} represented as a half precision binary

Floating-Point Numbers (5)

- Characteristics of IEEE 754 floating-points

Characteristic	Half precision	Single precision	Double precision
Size	16 bit	32 bit	64 bit
Character	5 bit	8 bit	11 bit
Fraction	10 bit	23 bit	52 bit
Exponent	-14 … 15	-126 … 127	-1022 … 1023
Bias	15	127	1023