IV.1. Alpha-numerical Codes

Alpha-numerical Codes

- the computer cannot represent characters directly
  - or any non-numerical information: images etc.
- each character is associated a unique number
  - the character is encoded
  - encoding can be at hardware level (elementary representation) or at software level

Standards

- ASCII
  - each character - 7 bits plus one parity bit
- EBCDIC
  - former competitor of ASCII
- ISO 8859-1
  - extends the ASCII code
- Unicode, UCS
  - non-latin characters

ASCII Code

- small letters are assigned consecutive codes
  - in the order given by the English alphabet
  - 'a' - 97; 'b' - 98; ...; 'z' - 122
- similarly - capitals (65, 66, ..., 90)
- similarly - characters that display decimal digits
  - attention: character '0' has code 48 (not 0)
- lexicographic comparison - binary comparison circuit

IV.2. Internal Number Representation

Positional Representation

- also a representation
  - 397 is not a number, but a number representation
- invented by Indians/Arabs
- implicit factor attached to each position in the representation
- essential in computer architecture
  - allows efficient computing algorithms

Base (Radix)

- any natural number d>1
- the set of digits for base d: {0,1,…,d-1}
- computers work with base d=2
  - technically: 2 digits - easiest to implement
  - theoretically: base 2 "matches" Boole logic
    - symbols and operations
    - operations can be implemented by Boole functions

Limits

- in practice, the number of digits is finite
- example - unsigned integers
  - 1 byte wide: $0 \div 2^{8}-1$ (= 255)
  - 2 bytes wide: $0 \div 2^{16}-1$ (= 65535)
  - 4 bytes wide: $0 \div 2^{32}-1$ (= 4294967295)
- any number that falls outside the limits cannot be represented correctly

Positional Writing

- consider base $d \in N^{*}-\{1\}$
- and the representation given by the string

  $a_{n-1}a_{n-2}\ldots a_1 a_0 a_{-1}\ldots a_{-m}$
- the corresponding number is

$$\sum_{i=-m}^{n-1} \left(a_i \times d^i\right)_{(10)}$$

- d^i is the implicit factor for position i
  - including negative powers

Converting from Base d to Base 10

- according to the previous formula
- the decimal point stays in the same position
- example

$5E4.D_{(16)} = 5\times16^2 + 14 \times16^1 + 4 \times16^0 +13 \times16^{-1} = 20480 + 3584 + 64 + 0.8125 = 24128.8125_{(10)}$

Converting from Base 10 to Base d

Example: $87.35_{(10)} = 1010111.01(0110)_{(2)}$

integer part

$87 / 2 = 43$ remainder 1

$43 / 2 = 21$ remainder 1

$21 / 2 = 10$ remainder 1

$10 / 2 = 5$ remainder 0

$5 / 2 = 2$ remainder 1

$2 / 2 = 1$ remainder 0

$1 / 2 = 0$ remainder 1

$87_{(10)} = 1010111_{(2)}$

(digits are considered bottom-up)

fractional part

$0.35 \times 2 = 0.7 + 0$

$0.7 \times 2 = 0.4 + 1$

$0.4 \times 2 = 0.8 + 0$

$0.8 \times 2 = 0.6 + 1$

$0.6 \times 2 = 0.2 + 1$

$0.2 \times 2 = 0.4 + 0$

$0.4 \times 2 = 0.8 + 0$

(period)

$0.35_{(10)} = 0.01(0110)_{(2)}$

Conversions between Bases

- one base is a power of the other base
  - $d_1 = d_2^k \Rightarrow$ to each digit in base d_1 correspond exactly k digits in base d_2
- both bases are powers of the same number *n*
  - conversion can be made through base *n*

$703.102_{(8)} = 111\ 000\ 011.001\ 000\ 010_{(2)} =$

$= 0001\ 1100\ 0011.0010\ 0001\ 0000_{(2)} =$

$= 1C3.21_{(16)}$

Approximation and Overflow

- a representation has *n* digits for the integer part and *m* digits for the fractional part
  - *n* and *m* are finite
- if the number requires more than *n* digits for the integer part, overflow occurs
- if the number requires more than *m* digits for the fractional part, approximation occurs
  - at most 2^{-m}

IV.3. BCD and Excess Representations

BCD Representation

- numbers are represented as strings of digits in base 10
  - each digit is represented on 4 bits
- utility
  - business applications (financiar etc.)
  - base 10 displays (temperature etc.)
- arithmetical operations - hard to perform
  - addition - cannot simply use a binary adder

BCD Addition (1)

5	=	0101	+		5	=	0101	+
3	=	0011			8	=	1000	
$8_{(10)}$	=	1000	$= 8_{BCD}$		$13_{(10)}$	=	1101	
							$\neq 13_{BCD} =$	
							$= 0001\ 0011$	

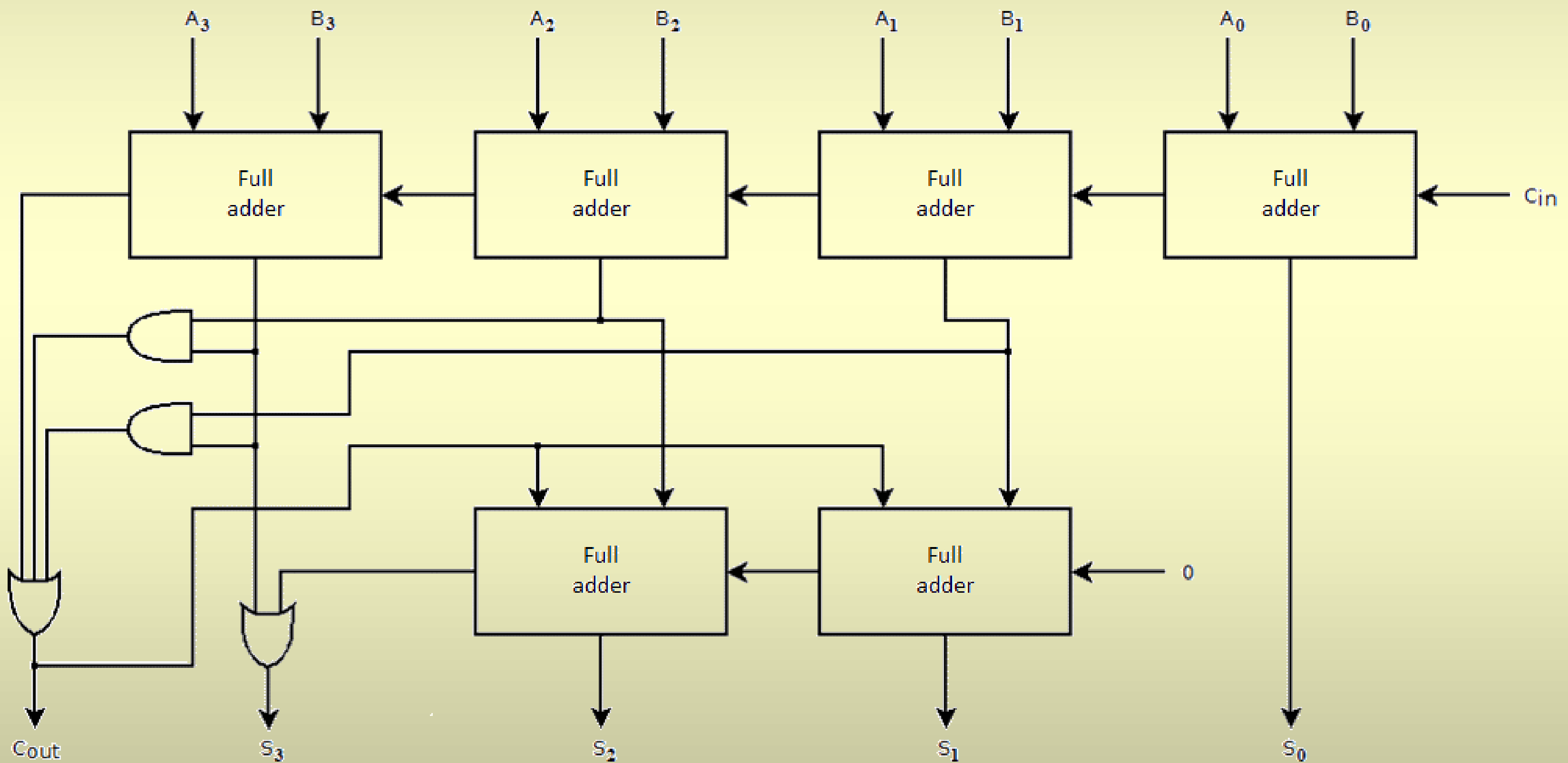
problems occur when the sum of the BCD digits exceeds 9

BCD Addition (2)

- solution
  - add 6 (0110) when the sum exceeds 9
- homework: why?

$$
\begin{array}{rcrl}
5 & = & 0101 & + \\
8 & = & \underline{1000} & \\
 & & 1101 & + \\
6 & = & \underline{0110} & \\
 & & 1\ 0011 & = 13_{BCD}
\end{array}
\qquad
\begin{array}{rcrl}
9 & = & 1001 & + \\
7 & = & \underline{0111} & \\
16_{(10)} & = & 1\ 0000 & \neq 16_{BCD} \\
6 & = & \underline{0110} & \\
 & & 1\ 0110 & = 16_{BCD}
\end{array}
$$

BCD Adder



A3
B3
A2
B2
A1
B1
A0
B0
Full adder
Full adder
Full adder
Full adder
Cin
Full adder
Full adder
0
Cout
S3
S2
S1
S0

Excess Representation

- based on positional writing
  - non-negative numbers
  - on n bits, the interval of numbers that can be represented is $0 \div 2^n-1$
- the Excess-k representation
  - for each bit string, subtract k from its value given by positional writing
  - the interval that can be represented: $-k \div 2^n-k-1$

Example: Excess-5

Binary	Decimal	Excess-5	Binary	Decimal	Excess-5
0000	0	-5	1000	8	3
0001	1	-4	1001	9	4
0010	2	-3	1010	10	5
0011	3	-2	1011	11	6
0100	4	-1	1100	12	7
0101	5	0	1101	13	8
0110	6	1	1110	14	9
0111	7	2	1111	15	10

IV.4. Fixed-point Representations

Numerical Representations: Problems

- sign representation
  - no special symbol available, only digit symbols
- decimal point
  - must know its position at each moment
- arithmetic operations
  - implementation - as efficient as possible
  - not possible for all operations at the same time
  - we must decide which operations to optimize

Fixed-point Encodings

- sign - use one of the bits
- decimal point
  - always the same position in the bit string
    - no need to explicitly memorize the position
- operations with efficient implementation
  - addition, subtraction
- encodings - on **n**+**m** bits (**n**≥1, **m**≥0)
  - **m**=0 - integer numbers
  - **n**=1 - subunit numbers

Redundant Encodings

- redundant encoding
  - there is at least one number with two distinct representations
  - problems with arithmetic operations
- encodings used in practice
  - positive number representation - same as for unsigned numbers; different only for negative numbers
  - some have two distinct representations for 0

Sign-magnitude Representation

- notation: A+S

$$val_{A+S}^{n,m}(a_{n-1}a_{n-2}\ldots a_1a_0a_{-1}\ldots a_{-m})=$$

$$=\begin{cases} a_{n-2}\times 2^{n-2}+\ldots+a_{-m}\times 2^{-m} & \text{if } a_{n-1}=0 \\ -(a_{n-2}\times 2^{n-2}+\ldots+a_{-m}\times 2^{-m}) & \text{if } a_{n-1}=1 \end{cases}$$

- similar to base 2 writing
  - the leftmost bit encodes the sign
  - decimal point - implicit

Sign-magnitude - Limits

- on n+m bits - 2^{n+m} distinct representations
  - but only $2^{n+m}-1$ distinct numbers
  - redundant: $val^{n,m}_{A+S}(00...0)=val^{n,m}_{A+S}(10...0)=0$
- extreme values that can be represented

  $max^{n,m}_{A+S}=val^{n,m}_{A+S}(01...1)=2^{n-1}-2^{-m}$

  $min^{n,m}_{A+S}=val^{n,m}_{A+S}(11...1)=-(2^{n-1}-2^{-m})$
  - so one can represent numbers within the interval $[-(2^{n-1}-2^{-m}); +(2^{n-1}-2^{-m})]$

Sign-magnitude - Precision

- numbers that can be represented exactly start from min=$-(2^{n-1}-2^{-m})$
  - and continue with step 2^{-m}
- the other numbers within the interval
  - approximation
  - error - at most 2^{-m}
  - so precision is 2^{-m}
- for fixed n+m
  - bigger numbers = poorer precision

Examples (1)

$$val_{A+S}^{8,0}(00110011)=2^5+2^4+2^1+2^0=51$$

$$val_{A+S}^{6,2}(00110011)=2^3+2^2+2^{-1}+2^{-2}=12.75$$

or

$$val_{A+S}^{6,2}(00110011)=val_{A+S}^{8,0}(00110011):2^2=51:4=12.75$$

$$val_{A+S}^{4,4}(00110011)=2^1+2^0+2^{-3}+2^{-4}=3.1875$$

or

$$val_{A+S}^{4,4}(00110011)=val_{A+S}^{8,0}(00110011):2^4=51:16=3.1875$$

Examples (2)

$$val_{A+S}^{8,0}(10110011)=-\left(2^5+2^4+2^1+2^0\right)=-51$$

$$val_{A+S}^{4,4}(10110011)=-(2^1+2^0+2^{-3}+2^{-4})=-3.1875$$

or

$$val_{A+S}^{4,4}(10110011)=val_{A+S}^{8,0}(10110011):2^4=-51:16=-3.1875$$

$$min_{A+S}^{8,0}=val_{A+S}^{8,0}(11111111)=-127$$

$$min_{A+S}^{4,4}=val_{A+S}^{4,4}(11111111)=-7.9375$$

or

$$min_{A+S}^{4,4}=min_{A+S}^{8,0}:2^4=-127:16=-7.9375$$

Examples (3)

$$max^{8,0}_{A+S}=val^{8,0}_{A+S}(01111111)=127$$

$$max^{4,4}_{A+S}=val^{4,4}_{A+S}(01111111)=7.9375$$

or

$$max^{4,4}_{A+S}=max^{8,0}_{A+S}:2^4=127:16=7.9375$$

- intervals for representation
  - $A+S^{8,0}$: [-127; 127] → 255 numbers, step 1
  - $A+S^{4,4}$: [-7.9375; 7.9375] → 255 numbers, step 0.0625 (=1:16)

Operations in A+S

- addition/subtraction
  - determine the sign of the result (comparison)
  - apply classic algorithms
- multiplication/division
  - similar to classic algorithms
- more complex than we wish
  - we cannot simply use a "classic" adder for computing the sum

One's Complement Representation

- notation: C_1

$$val_{C_1}^{n,m}(a_{n-1}a_{n-2}\ldots a_1 a_0 a_{-1}\ldots a_{-m})=$$

$$=\begin{cases} a_{n-2}\times 2^{n-2}+\ldots+a_{-m}\times 2^{-m} & \text{if } a_{n-1}=0 \\ (a_{n-2}\times 2^{n-2}+\ldots+a_{-m}\times 2^{-m})-(2^{n-1}-2^{-m}) & \text{if } a_{n-1}=1 \end{cases}$$

- homework: prove that the value is negative for $a_{n-1} = 1$
  - so a_{n-1} stands for the sign

One's Complement - Limits

- on n+m bits - 2^{n+m} distinct representations
  - but only $2^{n+m}-1$ distinct representations
  - redundant: $val_{C_1}^{n,m}(00...0)=val_{C_1}^{n,m}(11...1)=0$
- extreme values that can be represented

  $max_{C_1}^{n,m}=val_{C_1}^{n,m}(01...1)=2^{n-1}-2^{-m}$

  $min_{C_1}^{n,m}=val_{C_1}^{n,m}(10...0)=-(2^{n-1}-2^{-m})$
  - so one can represent numbers within the interval $[-(2^{n-1}-2^{-m}); +(2^{n-1}-2^{-m})]$

One's Complement - Precision

- numbers that can be represented exactly start from min=$-(2^{n-1}-2^{-m})$
  - and continue with step 2^{-m}
- the other numbers within the interval
  - approximation
  - error - at most 2^{-m}
  - so precision is 2^{-m}
- for fixed n+m
  - bigger numbers = poorer precision

Complementing

- representations of positive numbers - easy to determine
- harder for negative numbers
- is there a relation between the representations of numbers q and $-q$?
- yes: representation of $-q$ is achieved by negating all bits in the representation of q
  - commutative operation - also holds for $q < 0$

Examples (1)

$\mathrm{val}_{C_1}^{8,0}(00110011)=2^5+2^4+2^1+2^0=51$

$\mathrm{val}_{C_1}^{6,2}(00110011)=2^3+2^2+2^{-1}+2^{-2}=12.75$

or

$\mathrm{val}_{C_1}^{6,2}(00110011)=\mathrm{val}_{C_1}^{8,0}(00110011):2^2=51:4=12.75$

$\mathrm{val}_{C_1}^{4,4}(00110011)=2^1+2^0+2^{-3}+2^{-4}=3.1875$

or

$\mathrm{val}_{C_1}^{4,4}(00110011)=\mathrm{val}_{C_1}^{8,0}(00110011):2^4=51:16=3.1875$

Examples (2)

$$val_{C_1}^{8,0}(11001100)=(2^6+2^3+2^2)-(2^7-2^0)=-51$$

$$val_{C_1}^{4,4}(11001100)=(2^2+2^{-1}+2^{-2})-(2^3-2^{-4})=-3.1875$$

or

$$val_{C_1}^{4,4}(11001100)=val_{C_1}^{8,0}(11001100):2^4=-51:16=-3.1875$$

$$min_{C_1}^{8,0}=val_{C_1}^{8,0}(10000000)=0-(2^7-2^0)=-127$$

$$min_{C_1}^{4,4}=val_{C_1}^{4,4}(10000000)=0-(2^3-2^{-4})=-7.9375$$

or

$$min_{C_1}^{4,4}=min_{C_1}^{8,0}:2^4=-127:16=-7.9375$$

Examples (3)

$\max_{C_1}^{8,0} = \mathrm{val}_{C_1}^{8,0}(01111111) = 127$

$\max_{C_1}^{4,4} = \mathrm{val}_{C_1}^{4,4}(01111111) = 7.9375$

or

$\max_{C_1}^{4,4} = \max_{C_1}^{8,0} : 2^4 = 127:16 = 7.9375$

- intervals for representation
  - $C_1^{8,0}$: [-127; 127] $\rightarrow$ 255 numbers, step 1
  - $C_1^{4,4}$: [-7.9375; 7.9375] $\rightarrow$ 255 numbers, step 0.0625 (=1:16)

Operations in C_1

- can we add two numbers in C_1 with a "classic" adder?
- yes, but in two steps
  - in the second step, add the carry out to the result (from the first step)
  - so two adders are needed for addition
- subtraction: add the first operand to the symmetric of the second operand

Two's Complement Representation

- requirements
  - non-redundant representation
    - a single representation for 0
  - the sum of two numbers can be computed with a single adder
    - just as for unsigned numbers
    - gain - a single addition operation implemented in the processor for both signed and unsigned data types

Two's Complement

- notation: C_2

$$\mathrm{val}_{C_2}^{n,m}(a_{n-1}a_{n-2}\ldots a_1 a_0 a_{-1}\ldots a_{-m}) =$$

$$= \begin{cases} a_{n-2}\times 2^{n-2}+\ldots+a_{-m}\times 2^{-m} & \text{if } a_{n-1}=0 \\ (a_{n-2}\times 2^{n-2}+\ldots+a_{-m}\times 2^{-m})-2^{n-1} & \text{if } a_{n-1}=1 \end{cases}$$

- homework: prove that the value is negative for $a_{n-1} = 1$
  - so a_{n-1} stands for the sign

Two's Complement - Limits

- on n+m bits - 2^{n+m} distinct representations
  - and 2^{n+m} distinct numbers
  - 00...0 - the only representation for 0
- extreme values that can be represented

$$max_{C_2}^{n,m}=val_{C_2}^{n,m}(01...1)=2^{n-1}-2^{-m}$$

$$min_{C_2}^{n,m}=val_{C_2}^{n,m}(10...0)=-2^{n-1}$$

  - so one can represent numbers within the interval $[-2^{n-1}; +(2^{n-1}-2^{-m})]$ - asymmetrical

Two's Complement - Precision

- numbers that can be represented exactly start from min=-2^{n-1}
  - and continue with step 2^{-m}
- the other numbers within the interval
  - approximation
  - error - at most 2^{-m}
  - so precision is 2^{-m}
- for fixed n+m
  - bigger numbers = poorer precision

Complementing (1)

- is there a relation between the representations of numbers q and $-q$?
- yes: representation of $-q$ is the two's complement of the representation of q
  - negate all bits and add 0...01
  - just as for C_1, the operation is commutative - can be applied regardless of the sign of q

Complementing (2)

- example

  $q = 77$ is represented 01001101 in $C_2^{8,0}$

  $-q = -77$ is represented 10110010 + 00000001 = 10110011

- homework
  - the C_2 N-bit representation of the negative integer q is in fact the N-bit representation of the number $q + 2^N = 2^N - |q|$

Examples (1)

$$val_{C_2}^{8,0}(00110011)=2^5+2^4+2^1+2^0=51$$

$$val_{C_2}^{6,2}(00110011)=2^3+2^2+2^{-1}+2^{-2}=12.75$$

or

$$val_{C_2}^{6,2}(00110011)=val_{C_2}^{8,0}(00110011):2^2=51:4=12.75$$

$$val_{C_2}^{4,4}(00110011)=2^1+2^0+2^{-3}+2^{-4}=3.1875$$

or

$$val_{C_2}^{4,4}(00110011)=val_{C_2}^{8,0}(00110011):2^4=51:16=3.1875$$

Examples (2)

$$\mathrm{val}_{C_2}^{8,0}(11001101)=(2^6+2^3+2^2)\text{-}(2^7\text{-}2^0)=\text{-}51$$

$$\mathrm{val}_{C_2}^{4,4}(11001101)=(2^2+2^{-1}+2^{-2})\text{-}(2^3\text{-}2^{-4})=\text{-}3.1875$$

or

$$\mathrm{val}_{C_2}^{4,4}(11001101)=\mathrm{val}_{C_2}^{8,0}(11001101):2^4=\text{-}51:16=\text{-}3.1875$$

$$\mathrm{min}_{C_2}^{8,0}=\mathrm{val}_{C_2}^{8,0}(10000000)=0\text{-}2^7=\text{-}128$$

$$\mathrm{min}_{C_2}^{4,4}=\mathrm{val}_{C_2}^{4,4}(10000000)=0\text{-}2^3=8$$

or

$$\mathrm{min}_{C_2}^{4,4}=\mathrm{min}_{C_2}^{8,0}:2^4=\text{-}128:16=\text{-}8$$

Examples (3)

$$\max_{C_2}^{8,0} = \mathrm{val}_{C_2}^{8,0}(01111111) = 127$$
$$\max_{C_2}^{4,4} = \mathrm{val}_{C_2}^{4,4}(01111111) = 7.9375$$
or
$$\max_{C_2}^{4,4} = \max_{C_2}^{8,0} : 2^4 = 127:16 = 7.9375$$

- intervals for representation
  - $C_2^{8,0}$: [-128; 127] $\rightarrow$ 256 numbers, step
  - $C_2^{4,4}$: [-8; 7.9375] $\rightarrow$ 256 numbers, step 0.0625 (=1:16)

Conclusions

- C_2 is the most widely used representation
  - non-redundant
  - addition/subtraction - same implementation as for unsigned numbers
- in practice - integer data types from the programming languages
  - special case (m=0)
  - for real (actually rational) numbers, floating-point representations are used