Encoding Integers

- Encoding = symbolic representation of a value, in some specified number of digits, in some specified alphabet
  - We will consider encodings using the binary alphabet (0,1)
- We'll look at encoding integers briefly now, in more depth later (Chapter 7)
  - Also: encoding real numbers (Chapter 8)
- *Signed magnitude* representation
  - Precede number with sign bit

    0 = positive, 1 = negative
  - Examples (5-bit signed magnitude)

    $+13_{10} = +1101_2 = 01101_{2sm}$

    $-13_{10} = -1101_2 = 11101_{2sm}$

Encoding Integers (cont.)

- *Two's complement* representation
  - Represent positive numbers in n-bit signed magnitude form
  - Represent negative numbers as 2^n–N
  - Examples (5-bit two's complement)

    $+13_{10} = +1101_2 = 01101_{2'scomp}$

    $-13_{10} = 32 - 13 = 19 = 10011_{2'scomp}$
  - Examples (8-bit two's complement)

    $+13_{10} = +0001101_2 = 00001101_{2'scomp}$

    $-13_{10} = 256 - 13 = 243 = 11110011_{2'scomp}$
    - Short cut: start with 5-bit representation, and extend (replicate) the sign to produce 3 more significant digits

Encoding Characters

- *ASCII* (American Standard Code for Information Interchange) is a fixed-length code, of length 7
  - Examples (see page 18 of text for complete list)

    ! 010 0001

    \+ 010 1011

    3 011 0011

    9 011 1001

    H 100 1000

    M 100 1101

    h 110 1000

    m 110 1101

    CR 000 1101 (carriage return)
- Most memory systems can store 8 bits at a time
  - Extended ASCII uses that 8th bit

Encoding Characters (cont.)

- *Huffman coding* is a variable-length code
  - Basic idea: use less bits to represent more common characters
- Simple example:
  - Given a set of data that contains 50,000 instances of the six characters a, c, g, k, p, and z, which occur with the following percent frequencies:

a 48%	c 9%	g 12%
k 4%	p 17%	z 10%

  - The Huffman coding for these characters would be:

a 0	c 1101	g 101
k 1100	p 111	z 100

- Algorithm (more details in text): merge nodes with smallest values; label branch with smallest value as 0, other as 1

Error Detection & Check Sums

- Most consumer products are identified by a Universal Product Code (UPC), printed as a bar code with a number below
  - Example: UPC for Kellogg's Froster Mini-Wheats is 0 38000 54283 1
- First digit indicates type of product, next 5 digits identify manufacturer, next 5 digits identify product, last digit is a *check digit* (or *check sum*)
- Check sum is computed as follows:
  - Sum digits 1, 3, 5, 7, 9, 11, multiply by 3
  - Sum digits 2, 4, 6, 8, 10, add to product
  - Check sum is the number that must be added to sum to make it a multiple of 10
  - Example: (0+8+0+5+2+3)(3)+(3+0+0+4+8)=69 check sum is 1

Parity

- A similar idea to check sums is that of *parity*:
  - Add a bit, called the *parity bit*, to the encoding of a number or character
  - For even parity, the sum of the encoded digits, plus the parity bit, must be even
  - For odd parity, the sum of the encoded digits, plus the parity bit, must be odd
- Example — ASCII plus an even parity bit

	ASCII	ASCII w/ even parity
3	011 0011	0 011 0011
H	100 1000	0 100 1000
h	110 1000	1 110 1000

- If there is an error in a *single* bit, it will be *detected* by a parity encoding with a *single* parity bit

Homework #2 — Due 9/14/98 (Part 2)

4. Give the 8-bit two's complement encoding of the following:

   -46_{10} 78_{10}

5. The UPC code for Ty Inc's Beanie Babie named "Bernie" begins 0 08421 04109. Show the computation of the check sum.