

## Floating Point Numbers

- Fixed point numbers assumed that the radix point was in a fixed position in the word, i.e., the number of integer places and fractional places were defined a priori and fixed.
- This implies representing a narrow range of numbers with the number of fractional digits defining the resolution via the basic increment  $r^{-n}$ .
- Such a narrow range in magnitude is not acceptable for most scientific computations.
- The **floating point** representation is used in these cases.
- The radix point is fixed in the word as it is stored BUT an exponent is also stored which is applied to the value of the fixed point word to scale it.

## Examples:

- General format:  $M \times r^E$ , where  $M$  is the mantissa and  $E$  is the exponent.
- The mantissa and exponent are signed values that can be encoded in various ways.
- The mantissa is usually fractional with a radix point assumed to be to the left of all digits.
- The exponent is an integer.
- Note normalization in mantissas below to have leading nonzero.
- For binary codes the leading (1) is often assumed. (0 must have a special representation)

Value	Encoding
512.2	$.5122 \times 10^3$
0.0032	$.3200 \times 10^{-2}$
10.5	$.(1)0101 \times 2^4$

## Floating Point Arithmetic

- Multiplication is easy: Multiply mantissas and add exponents, then normalize resulting mantissa.
- Note that when multiplying two  $m$  bit mantissas the result is a  $2m$  bit mantissa. It may be necessary to truncate digits from the product after normalization.
- Addition requires more care. Before adding the exponents must be made the same. Therefore the mantissa of one of the operands must be shifted.
- Note that information is sometimes lost. (Although relative error stays small.)

$$\begin{array}{rcl} & 453 & + 5 \\ .453 \times 10^3 & + & .500 \times 10^1 \\ .453 \times 10^3 & + & .005 \times 10^3 \\ & = & .458 \times 10^3 \end{array}$$

$$\begin{array}{rcl} & 453 & + 5.21 \\ .453 \times 10^3 & + & .521 \times 10^1 \\ .453 \times 10^3 & + & .005 \times 10^3 \\ & = & .458 \times 10^3 \end{array}$$

## Error Detection and Correction

- encoded data when read or transmitted can be corrupted by permanent or transient failures in the digital hardware
- this can be modeled as transmitting a binary pattern that is an encoded version of accurate information and receiving a binary pattern that may be the same as that transmitted or that may have had 1 or more bits altered.
- If  $b$  bits are used to encode the data and if there are  $2^b$  patterns that represent accurate information then error detection and correction is not possible.
- This is due to the fact that any perturbation of the  $b$  bits sent still results in  $b$  bits being received and constituting a legitimate pattern (there are no illegitimate patterns).
- Solution add  $e$  bits to the pattern so that there are still  $2^b$  legitimate patterns, called **codewords**, but they are encoded in  $b + e$  bits. There are therefore many illegitimate patterns that could be received.
- The development of codes that allow error detection and correction is a huge body of literature. We present the basic goals of the development.

**Definition:** If  $x$  and  $y$  are two  $b$  bit binary patterns the **Hamming distance**  $\delta(x, y)$  between  $x$  and  $y$  is defined to be the number of bit positions in which the patterns differ.

**Examples:**

$$x = 1101 \quad y = 1101 \quad \delta(x, y) = 0$$

$$x = 0001 \quad y = 1001 \quad \delta(x, y) = 1$$

$$x = 0000 \quad y = 0110 \quad \delta(x, y) = 2$$

$$x = 0000 \quad y = 1011 \quad \delta(x, y) = 3$$

$$x = 1011 \quad y = 0100 \quad \delta(x, y) = 4$$

**Basic requirement 1:** In order to **detect** up to  $e$  bit errors in a codeword, all pairs of codewords must have a Hamming distance greater than  $e + 1$ .

Suppose  $x_1 = 000$  is transmitted. The possible patterns for the received word  $y$ , **assuming no errors or a single bit error**

$y$	000	100	010	001
-----	-----	-----	-----	-----

If the code has a minimum Hamming distance of 2 then the nearest codewords to  $x_1$  are a subset of:

$$\begin{aligned}x_2 &= 110 \\x_3 &= 101 \\x_4 &= 011\end{aligned}$$

Therefore,  $y$  can be identified as a noncodeword and the single bit error detected if it in fact occurred.

Note, however, that the same  $y$  (other than 000) could have resulted from transmitting one of the codewords  $x_2, x_3$ , or  $x_4$  with the appropriate single bit error. For example,  $x_2 = 110$  could have been sent and the first bit corrupted via error to get  $y = 010$ .

In either case an error would have been detected but it would not be possible to determine which codeword was actually sent, i.e., the single bit error could not be corrected.

- The simplest form of single bit error detecting codes is the **parity** code.
- Suppose all codewords have an even number of bits set to 1 (0 is considered an even number) then the minimum Hamming distance is 2. This is clear since changing a single bit in a codeword adds or removes a single 1 bit and therefore results in an odd number of 1 bits in the received word.
- The same is true if all codewords have an odd number of bits set to 1.
- $2^b$  distinct codewords can be encoded via a parity code to detect single bit errors by using a single extra bit, i.e., a binary pattern of length  $b + 1$  is used.
- Such codes have  $2^b$  codewords and  $2^b$  noncode-words (the best ratio possible).
- The even parity encoding simply adds a 1 to the  $b$  bits of data to be encoded if the ones count is odd; otherwise the extra bit is set to 0.
- The odd parity encoding simply adds a 1 to the  $b$  bits of data to be encoded if the ones count is even; otherwise the extra bit is set to 0.

## Parity Encoding of 4 symbols using 3 bits

2-bit symbol	even parity encoding	odd parity encoding
00	000	001
01	011	010
10	101	100
11	110	111

- note the odd parity codewords are the noncode-words for the even parity code and vice versa.
- All single bit errors are detected.
- All errors changing an odd number of bits are detected.
- Codewords are easily recognized by counting 1s in word received.

**Basic requirement 2:** There must be an efficient way of determining if a received binary pattern is a code-word or not.

## Error Correcting

Suppose an even parity code is used and 010 is received. The possibilities are:

sent	error position
110	$b_2$
011	$b_0$
000	$b_1$

In order to correct we must have a minimum Hamming distance of 3.

Trivial example of a code with a minimum Hamming distance of 3.

1-bit symbol	codeword
0	000
1	111

- If 011 or 101 or 110 is received via a single bit error it is properly corrected to 111.
- If 001 or 100 or 010 is received via a single bit error it is properly corrected to 000.
- But, for example, if 011 is received via a two bit error after transmitting 000 then it is miscorrected to 111.

- We can detect and correct 1 bit errors.
- We can detect 2 bit errors.
- We cannot do both with  $\delta_{min} = 3$ .
- In general, if  $\delta_{min} = e$  then you can correct  $e$  bit errors OR detect  $2e$  bit errors.
- Extra distance must be added to detect and correct reliably.
- $\delta_{min} = 2e + d + 1$  allows the code to detect and correct errors in up to  $e$  bits AND detect errors in up to  $e + d$  bits.
- Errors in more than  $e + d$  bits will be miscorrected. This is unavoidable anytime codes correct automatically.
- probabilities are assigned to the various errors for a given piece of hardware and the codes constructed accordingly.

## Hamming Codes

- Most memory codes and many transmission codes correct single bit and detect double bit errors.
- We need a way to generate such codes for arbitrary word lengths with the number of parity bits growing slowly.
- The idea of parity codes can be used to create codes with distances 3 and 4 by mixing several parity codes together.
- Due to Hamming (1950)
- Overlapping subwords are protected by parity bits.
- Results of independent parity checks identifies locations of errors
- Requires only a logarithmic number of parity bits.

- Assume that we have  $n = 2^m - 1$  bits in the word.
- The Hamming code uses  $m$  parity bits (which is approximately  $\log n$ ).
- Each bit will be included in multiple parity groups in which even parity is maintained. (Except the parity bits themselves which are only in their individual groups)
- Number the bits  $1, 2, \dots, 2^m - 1$ .
- Determine the  $m$  bit binary representation of  $i$ . Then the  $i$ -th bit of the word is included in the parity groups corresponding to the position of 1s in this representation.
- To satisfy the fact that parity bits are in only 1 group we must take the parity bits to be in positions  $1, 2, 4, \dots$  since these have encodings  $001, 010, 100$  with only one position set to 1
- Note that physically the parity bits can be anywhere in the word, this ordering is necessary to determine groupings only.

position	binary	parity bits involved
1*	001	1
2*	010	2
3	011	1,2
4*	100	4
5	101	1,4
6	110	2,4
7	111	1,2,4

\* indicates parity bits

7	6	5	4	3	2	1	bits
0	1	1	0*	1	0*	0*	correct
1	1	1	0*	1	0*	0*	bad bit 7
-	-	-	*	-	*	*	parity errors

Binary pattern of incorrect parity indicates position to be corrected.

## More Detail on Previous Example

We want to include 4 bits of information using the 7 bit Hamming code with  $\delta_{min} = 3$ . Suppose the word we want to transmit or store is 0111. This is placed in bit positions 7,6,5, and 3 of the codeword in the first step of encoding via the Hamming code. These are the information bits of the codeword. Bits 1,2, and 4 are parity bits and must be computed to complete the encoding.

7	6	5	4	3	2	1	bits
0	1	1	-	1	-	-	codeword

Parity bit 1 is computed by forcing even parity in the bits in positions 1,3,5, and 7. There are two 1s in the information bits in those positions so the parity bit in position 1 is set to 0.

7	6	5	4	3	2	1	bits
0	1	1	-	1	-	0	codeword

Parity bit 2 is computed by forcing even parity in the bits in positions 2,3,6 and 7. There are two 1s in the information bits in those positions so the parity bit in position 2 is set to 0.

7	6	5	4	3	2	1	bits
0	1	1	-	1	0	0	codeword

Parity bit 4 is computed by forcing even parity in the bits in positions 4,5,6 and 7. There are two 1s in the information bits in those positions so the parity bit in position 4 is set to 0.

7	6	5	4	3	2	1	bits
0	1	1	0	1	0	0	codeword

This completes the generation of the codeword that is transmitted or stored.

Now suppose when receiving the 7 bit word or in reading it from memory we corrupt bit 7, i.e., it goes from  $0 \rightarrow 1$  As shown in the example.

7	6	5	4	3	2	1	bits
1	1	1	0	1	0	0	received

The first step in the decoding process is to determine if the pattern is indeed a codeword or if the parity is incorrect. This is done by checking if the parity in the received word is still correct.

The parity group for bit 1 is the bits in positions 1,3,5, and 7. There are **three** 1s in bits in those positions so there is a parity error in group 1.

The parity group for bit 2 is the bits in positions 2,3,6, and 7. There are **three** 1s in bits in those positions so there is a parity error in group 2.

The parity group for bit 4 is the bits in positions 4,5,6, and 7. There are **three** 1s in bits in those positions so there is a parity error in group 3.

- The parity is incorrect in one or more groups.
- Therefore, the received word is **not a codeword**
- We can record the parity groups that failed with a three bit pattern  $(g_4, g_2, g_1) = 111$  by making the corresponding bit 1 for those groups that failed.
- To determine which bit is in error we must identify which parity bits would have changed following a single bit error in each of the 7 positions.

- If bit 1 had been changed during transmission the parity bit in position 1 would have changed and  $(g_4, g_2, g_1) = 001$  would be generated.
- If bit 2 had been changed during transmission the parity bit in position 2 would have changed and  $(g_4, g_2, g_1) = 010$  would be generated.
- If bit 3 had been changed during transmission the parity bits in position 1 and position 2 would have changed and  $(g_4, g_2, g_1) = 011$  would be generated.
- If bit 4 had been changed during transmission the parity bit in position 4 would have changed and  $(g_4, g_2, g_1) = 100$  would be generated.
- If bit 5 had been changed during transmission the parity bits in position 1 and position 4 would have changed and  $(g_4, g_2, g_1) = 101$  would be generated.
- If bit 6 had been changed during transmission the parity bits in position 2 and position 4 would have changed and  $(g_4, g_2, g_1) = 110$  would be generated.
- If bit 7 had been changed during transmission the parity bits in position 1, position 2 and position 4 would have changed and  $(g_4, g_2, g_1) = 111$  would be generated.

- So if we assume that any error that occurs is a **single bit error** then the only one that could have produced the change to the parity bits in positions 1,2, and 4 is a change to bit 7. Note that the pattern that records which parity checks failed during decoding for this example,  $(g_4, g_2, g_1) = 111$ , is simply a binary number equal to 7. It is used to indicate which position must be changed to correct the single bit error.
- The received word is then corrected by restoring bit 7 to a value of 0, i.e., undoing the error that occurred during transmission.
- The grouping of the bits to set each of the parity bits is made so that a single error to the word in each bit **results in a unique pattern for**  $(g_4, g_2, g_1)$  during the decoding process and this pattern gives the position of the error. This grouping to define the parity groupings is Hamming's insight.
- If an error had occurred in bit 5 rather than 7 the decoding process about would have produced  $(g_4, g_2, g_1) = 101$ . A change to bit 5 is the only error that can produce that pattern.

- All single bit errors can be corrected therefore it has  $\delta_{min} \geq 3$
- All two bit errors can be detected due to  $\delta_{min}$
- 2 bit errors cannot all be corrected however.
- Suppose bits 6 and 7 are in error in previous example, then parity bits 2 and 4 still yield even parity, BUT since bit 7 also influences parity bit 1, we can detect the error but not correct it -  $\delta_{min} = 3$ .
- As with all distance 3 codes we can choose to correct single bit or detect double bit errors. Not both.
- To get  $\delta_{min} = 4$  we must add another parity bit that involves all of the  $2^m$  bits in the word.
- many other more advanced codes are possible e.g. Reed-Solomon and BCH codes. Math is based on polynomials over finite fields.