CS/ECE 552 Introduction to Computer Architecture

ERROR DETECTION AND ERROR CORRECTION THROUGH REDUNDANCY

Different components of a computer system basically perform the following functions:

1. Information processing - logic elements, CPU etc.
2. Information storage - RAM registers, etc.
3. Information transmission - interconnects, lines etc.

All these components are amenable to failures and, therefore, can produce erroneous output(s). Failures can be permanent (e.g., a line is permanently open, a memory cell is incapable of storing a 0, etc.) or intermittent (a loose connection). In either case, these failures manifest themselves, provided proper conditions are present, as erroneous outputs, i.e., some of the output bits are changed. (In the literature, permanent faults have often been modeled as stuck-at faults, i.e., a line will stay at logic 1 or 0 no matter what signal is applied to it. Although most permanent faults can be represented by a stuck-at fault model, this is not true for all permanent faults). Our concern here is *to detect an erroneous output and if possible take a corrective action.* We will achieve this by coding information, i.e., by appending some redundant bits to the information bits in such a manner that the errors can be detected and possibly corrected by special logic circuits. Let us demonstrate the principal by a simple example.

Duplication: Let us consider the case where each information bit x is duplicated (for example transmitted over two lines or stored at two locations in RAM). If only *one error* is likely to occur then at receiving end instead of xx we will receive $x\bar{x}$ or $\bar{x}x$ (but not xx or $\overline{xx}$). Thus we will immediately know that *an error has occurred* whenever we receive two bits which are not equal. This is error detection. Of course, we will not be able to determine what was transmitted. Note that we assume that retransmission of the same information is not allowed because if retransmission were allowed then it would be possible to develop a strategy which will help locate the fault with duplication. However, in true sense this is quadruplication-once in space and once in time. A corrective action is possible if information is *triplicated* and we assume that no more than a *single error* takes place.

Model

Although different devices fail in different modes, thus causing different error types (error patterns), some common assumptions can be made about them. For semiconductor RAM and information transmission, these assumptions are very close to reality. These assumptions are:

1. Errors occur randomly
2. Single bit errors are more probable than multiple bit errors.

In this note we shall discuss methods of handling random errors. Also, the probability of *multiple* errors will be assumed to be negligible, but we will still be interested in *detecting* multiple errors (at least a class of them) to minimize any chance of not detecting a fatal error.

Error Detection

One of the simplest schemes used for *single error detection* is the use of a check bit called *parity bit*. Let $A = \{a_0, a_1, \cdots, a_{n-1}\}$ be an n-bit word. A parity bit p is defined as

$$p = a_0 \oplus a_1 \oplus \cdots \oplus a_{n-1}$$

where $\oplus$ denotes the Exclusive-OR operation. In this scheme whenever a word A is stored in RAM it is stored along with p. On reading RAM, parity is recomputed and checked against stored value of p. An error is said to have occurred if computed p and stored p do not match.

It is not difficult to see that the above scheme will detect not only a single error but odd number of errors in the stored word. Note also that the error(s) detected need not be confined to the word A but an error in the parity bit will also be detected.

In the above scheme the total number of 1's in the vector $\{a_0, a_1, \cdots, a_{n-1}\}$ is *even*; therefore, it is called an *even parity scheme*. If following equation is used to compute p then it is called an odd parity scheme.

$$p = 1 \oplus a_0 \oplus a_1 \oplus \cdots \oplus a_{n-1}$$

Error Correction

Let us define a term and look at the properties of binary vectors.

Definition: For two binary vectors A = $\{a_0, a_1, \cdots, a_{n-1}\}$ and B = $\{b_0, b_1, \cdots, b_{n-1}\}$ the **Hamming distance** H(A,B) is

$$\sum_{i=0}^{i=n-1} (a_i \oplus b_i)$$

Example For A = (0 1 0 0), B = (0 0 1 0) H(A,B) = 2

Another way to look at the Hamming distances is that if A and B are nodes of a binary n-cube then the Hamming distance between them is the minimum number of links traversed to get from node A to B (or from node B to A). It is also evident that two adjacent nodes on the binary n-cube have a Hamming distance of one.

In order to introduce redundancy, we add extra bits to a data word and the result is called a *codeword*. The extra bits are selected in such a way that not only the Hamming distance between any pair of code words is large but the minimum of all the Hamming distances between all pairs of code words in the code set or code space (set containing all the code words) is maximized.

Consider the case of appending a parity bit (even parity, say) to each data word. As a result every code word will have even numbers of 1's in it. Hence the code space consists of all words containing even number of 1's. From this we can conclude that the minimum Hamming distance for such a *code* will be at least two.

Minimum Hamming distance and error detection/correction capability are closely related.

If Hamming distance between two codewords is α then no $\alpha-1$ errors can transform a codeword into another codeword. Thus all $\alpha-1$ errors in such a code (code space) will be detectable.

The following two results can be derived from the above discussion:

Result 1: All errors of α or fewer bits in a codeword can be *detected* if and only if the minimum Hamming distance of the code is α+1.

Result 2: All errors of t or fewer bits in a codeword can be *corrected* if and only if minimum Hamming distance of the code is 2t+1.

This follows from the fact that any codeword with t errors will still be at least distance t+1 from any other codeword. Thus a scheme which will *map* erroneous information to the nearest (in the sense of Hamming distance) codeword will cause t errors to be corrected.

The above two results when merged together provide the following result:

Result 3: A code is a t error-correcting, α error detecting (α >= t) code if and only if its minimum Hamming distance is t+α+1.

The relation between Hamming distance (H), number of bit errors that can be detected (ED) and number of bit errors that can be corrected (EC) is given in the following table.

H	ED	EC
1	0	0
2	1	0
3	1	1
	2	0
4	2	1
	3	0
5	2	2
	3	1
	4	0
6	3	2
	4	1
	5	0
7	3	3
	4	2
	5	1
	6	0

Hamming Code

In this scheme, several check bits (*k*) are generated for a data word of *m* bits by using multiple parity checks on certain subsets of the data bits. The check bits are then combined with the data bits to form a codeword. Note that for a code to have single error detection and single error correction capability the combination of *k* check bits must be able to identify any one of the (*m*+*k*) faulty position in case of single error and also it should be able to indicate that no bit is faulty. Thus the relation between *m* and *k* is

$$2^k \geq m+k+1 \tag{1}$$

We explain the construction of a Hamming code by way of an example. A Hamming code can be described by its "parity check matrix" *P* conisting of *n* columns (*n*=*m*+*k*), each corresponding to one of the n bits of the encoded word, and k rows each corresponding to one of the parity check bits. The elements of the matrix are 0s and 1s; the positions of the 1s in the i^{th} row indicate which bit positions are involved in the parity check equation. Similarly, the positions of 1s in the j^{th} column indicate the parity pattern corresponding to the j^{th} bit.

It can be seen that if m = 4, then the smallest k satisfying the equation (1) is k = 3. Thus 3 check bits have to be appended to the 4 data bits in order for the Hamming code to be single-error correcting. The bit positions of the code are labeled, with numbers 1 through 7:

Bit positions	1	2	3	4	5	6	7
Bit names	C_1	C_2	b_1	C_3	b_2	b_3	b_4

The bit positions corresponding to powers of 2 are used as check bits C_1, C_2 and C_3 respectively. The other bit positions correspond to the data bits b_1 to b_4. The parity check matrix for the Hamming code with m = 4, k = 3 is

	C_1	C_2	b_1	C_3	b_2	b_3	b_4
	1	0	1	0	1	0	1
P =	0	1	1	0	0	1	1
	0	0	0	1	1	1	1

It can be seen from the parity check matrix that

$$C_1 = b_1 \oplus b_2 \oplus b_4$$

$$C_2 = b_1 \oplus b_3 \oplus b_4$$

$$C_3 = b_2 \oplus b_3 \oplus b_4$$

For example, if $b_1 b_2 b_3 b_4 = 1010$ then $C_1 = 1$, $C_2 = 0$ and $C_3 = 1$. Thus the corresponding codeword is

Bit positions	1	2	3	4	5	6	7
Codeword	1	0	1	1	0	1	0

Suppose that the encoded word is stored in the memory and on a read operation bit 3 changes from 1 to 0. To determine whether the word is correct or not, the checkbits for the data word are recomputed. These new checkbits are compared against the checkbits readout with the data. The vector so obtained after comparing is called *syndrome*. For our example the new checkbits are

$$C'_1 = 0,\;\; C'_2 = 1 \text{ and } C'_3 = 1$$

The syndrome S is $e_3 e_2 e_1$ where

$$e_1 = C_1 \oplus C'_1;\;\; e_2 = C_2 \oplus C'_2 \text{ and } e_3 = C_3 \oplus C'_3.$$

Thus S = 0 1 1 for our example and this corresponds to bit position 3 which must be in error. For corrective action, bit position 3 must be inverted. Note that S = 0 0 0 implies no error.

Hamming codes with distance 3 can detect two-bit errors but can only be used to correct single-bit errors. However, as evident from previous discussion, a Hamming code with a minimum distance of 3 can not simultaneously detect 2 errors and correct a single error. For example, if bits 3 and 5 are erroneous in the previous example then the received information word will be,

Bit positions	1	2	3	4	5	6	7
Received word	1	0	0	1	1	1	0

Then $C'_1 = 1$, $C'_2 = 1$ and $C'_3 = 0$. Hence the syndrome (error address) is $e_3 = 1(1 \oplus 0)$, $e_2 = 1(0 \oplus 1)$ and $e_1 = 0(1 \oplus 1)$, which points to bit position 110(=6_{10}). But bit position 6 is not erroneous! Thus any attempt to execute double-bit error correction with distance-3 Hamming code will result in an incorrect result.

The single-error correcting Hamming code can be converted into a distance-4 code with the addition of another parity check bit at bit position 8.

Bit positions	1	2	3	4	5	6	7	8
Bit names	C_1	C_2	b_1	C_3	b_2	b_3	b_4	C_4

This bit checks parity over the entire eight-bit word. When the overall parity check of the encoded word is correct and the syndrome is zero, there is no bit error. If the overall parity is wrong and the syndrome has a nonzero value, there is a single, correctable bit error. If the overall parity of the encoded word is correct but the syndrome is nonzero, there is a noncorrectable double bit error in the word. Note that this conclusion is based on the assumption that no more than two errors can take place.

Other Schemes

Many error coding schemes also exist which suit different needs. An example is *k* out of 2*k* code. In this code information is coded such that each codeword is of length 2*k* and contains exactly *k* 1's in it. Such codes are useful for the design of checkers. However, Hamming codes and their variations, e.g., modified Hamming code in the form of odd-weighted column code, are by far most popular for use in memories.

References

[1] Peterson, W. W. and E. J. Weldon, *Error Correcting Codes*, MIT Press (1972).

[2] Lala, P. K, *Fault-Tolerant and Fault Testable Hardward Design*, Prentice Hall (1984) (pp. 94-95).