Announcements

- Please download “Error Detection and Correction” from class web site (“For Students” -> Fourth Edition -> Reading Supplements).

Error Detection And Correction

- The small size of the transistors, combined with cosmic ray effects causes occasional errors in stored information in large, dense RAM chips.
- These errors can be detected and corrected by employing error-detecting and –correcting codes in RAMs.

Parity Bit

- **To detect errors in data communication and processing, an additional bit is sometimes added to a binary code word to define its parity.**
- **A parity bit is the extra bit included to make the total number of 1's in the resulting code word either even or odd.**

Parity Bit Example

	With Even Parity	With Odd Parity
• **1000001**	**01000001**	**11000001**
	7 bits → 8 bits	
• **1010100**	**11010100**	**01010100**

Characteristics of Parity

- **Even parity is more common**
- **Parity may be used with binary numbers as well as with codes including ASCII**
- **The parity bit may be placed in any fixed position in the code**

Parity Bit Generating and Checking

- **An even parity bit is generated at the sending end for all 7-bit ASCII characters**
- **The 8-bit characters including the parity bits are transmitted to their destination**
- **The parity of each character is then checked at the receiving end**
- **If the parity of the received character is not even (assuming even parity is used), at least one bit has changed its value during the transmission**

Pros & Cons

- **If the number of bits changed is even, the error will not be detected.**
- **It can only detects one, three, or any odd number of errors in each character.**
- **Parity does not indicate which bit contained the error, even when it can detect it.**
- **The data must be discarded entirely, and re-transmitted from scratch.**
- It uses only a single bit, resulting in the least overhead.

Hamming Codes

- **The most common types of error-correcting codes used in RAM are based on the codes devised by R.W. Hamming.**
- ***K* parity bits are added to an *n*-bit data word, forming a new word of *n* + *k* bits.**
- **The bit positions are numbered in sequence from 1 to *n* + *k*.**
- **Those positions numbered with powers of two are reserved for the parity bits.**
- **The code can be used with words of any length.**

(11,7) Hamming Code Example

	p_1	p_2	d_1	p_3	d_2	d_3	d_4	p_4	d_5	d_6	d_7
Data word (without parity):			0		1	1	0		1	0	1
p_1	1		0		1		0		1		1
p_2		0	0			1	0			0	1
p_3				0	1	1	0				
p_4								0	1	0	1
Data word (with parity):	1	0	0	0	1	1	0	0	1	0	1

Calculation of Hamming code parity bits

7-bit data word "0110101“ (d - data bits, p - parity bits)

Hamming Code Example (Cont.)

	p_1	p_2	d_1	p_3	d_2	d_3	d_4	p_4	d_5	d_6	d_7	Parity check	Parity bit
Received data word:	1	0	0	0	1	1	0	0	1	0	0		
p_1	1		0		1		0		1		0	Fail	1
p_2		0	0			1	0			0	0	Fail	1
p_3				0	1	1	0					Pass	0
p_4								0	1	0	0	Fail	1

Checking of parity bits (switched bit highlighted)

- Assume the final bit gets corrupted and turned from 1 to 0
- Flag each parity bit as 1 when the even parity check fails

Hamming Code Example (Cont.)

	P_4	P_3	P_2	P_1	
Binary	1	0	1	1	
Decimal	8		2	1	$\Sigma = 11$

- Evaluate the value of the parity bits.
- The integer value of the parity bits is 11, signifying that the 11th bit in the data word (including parity bits) is wrong and needs to be flipped.

Hamming Code Example (Cont.)

- Flipping the 11th bit gives changes 10001100100 back into 10001100101.

- Removing the Hamming codes gives the original data word of 0110101.

How about parity bit error?

As parity bits do not check each other, if a single parity bit check fails and all others succeed, then it is the parity bit in question that is wrong and not any bit it checks.

Hamming codes with additional parity

- The extra parity bit applies to all bits after the Hamming code check bits have been added.
- Then all single-bit, two-bit and three-bit errors can be detected.
- two-bit errors can be distinguished from single-bit and three-bit errors.
- Single-bit errors can be corrected.

Two-bit errors

Two-bit errors can at least be recognized:

- When using correction, if a parity error is detected and the Hamming code indicates that there is an error, this error can be corrected.
- However, if a parity error is not detected but the Hamming code indicates that there is an error, this is assumed to have been due to a two-bit error, which is detected but cannot be corrected.
- Hamming Codes are capable of correcting one error or detecting two errors but not capable of doing both simultaneously.

Hamming Rule

$$d + p + 1 <= 2^p$$

Where d is the number of data bits and p is the number of parity bits

Suitable Code

- From the practical standpoint of communications, a (11,7) code is not a good choice, because it involves non-standard character lengths.
- Designing a suitable code requires that the ratio of parity to data bits and the processing time involved to encode and decode the data stream be minimized.
- A code that efficiently handles 8-bit data items is desirable.

4 Parity Bits

- Can provide error correction for five to eleven data bits.

- A (12,8) code then offers a reasonable compromise in the bit stream .

Why (12,8) code is desirable ?

The code enables data link packets to be constructed easily by permitting one parity byte to serve two data bytes.

Another Example (12,8)

Bit position	1	2	3	4	5	6	7	8	9	10	11	12
	P_1	P_2	1	P_4	1	0	0	P_8	0	1	0	0

Assume a 8-bit data word 11000100. We include 4 parity bits with this word and arranged the 12 bits as above. Please compute each parity bit value:

Calculating Parity Bits

$$P_1 = \text{XOR of bits } (3, 5, 7, 9, 11) = 1 \oplus 1 \oplus 0 \oplus 0 \oplus 0 = 0$$

$$P_2 = \text{XOR of bits } (3, 6, 7, 10, 11) = 1 \oplus 0 \oplus 0 \oplus 1 \oplus 0 = 0$$

$$P_4 = \text{XOR of bits } (5, 6, 7, 12) = 1 \oplus 0 \oplus 0 \oplus 0 = 1$$

$$P_8 = \text{XOR of bits } (9, 10, 11, 12) = 0 \oplus 1 \oplus 0 \oplus 0 = 1$$

- We're using Even Parity.

The Entire Code

Bit position	1	2	3	4	5	6	7	8	9	10	11	12
	0	0	1	1	1	0	0	1	0	1	0	0

The 8-bit data word is written into the memory together with the 4 parity bits as a 12-bit composite word. Substituting the 4 parity bits in their proper positions, we obtain the 12-bit composite word.

Checking

$$C_1 = \text{XOR of bits } (1, 3, 5, 7, 9, 11)$$

$$C_2 = \text{XOR of bits } (2, 3, 6, 7, 10, 11)$$

$$C_4 = \text{XOR of bits } (4, 5, 6, 7, 12)$$

$$C_8 = \text{XOR of bits } (8, 9, 10, 11, 12)$$

- When the 12 bits are read from memory, they are checked again for errors
- The parity of the word is checked over the same groups of bits, including their parity bits

How to locate the error bit?

$C=C_8C_4C_2C_1$

- If C=0, there is no error has occurred.

- If C!=0, the 4-bit binary number formed by the check bits gives the position of the erroneous bit if only a single bit is in error.

Three Cases

Bit position	1	2	3	4	5	6	7	8	9	10	11	12	
	0	0	1	1	1	0	0	1	0	1	0	0	No error
	1	0	1	1	1	0	0	1	0	1	0	0	Error in bit 1
	0	0	1	1	0	0	0	1	0	1	0	0	Error in bit 5

	C_8	C_4	C_2	C_1
No error	0	0	0	0
Error in bit 1	0	0	0	1
Error in bit 5	0	1	0	1

The error can then be corrected by complementing the corresponding bit.

An error can occur in the data or in one of the parity bits.

Algorithm for General Hamming

- **All bit positions that are powers of two are used as parity bits. (positions 1, 2, 4, 8, 16, 32, 64, etc.)**
- **All other bit positions are for the data to be encoded. (positions 3, 5, 6, 7, 9, 10, 11, 12, 13, 14, 15, 17, etc.)**
- **Each parity bit calculates the parity for some of the bits in the code word. The position of the parity bit determines the sequence of bits that it alternately checks and skips.**
  - **Position 1 (n=1): skip 0 bit (0=n-1), check 1 bit (n), skip 1 bit (n), check 1 bit (n), skip 1 bit (n), etc.**
  - **Position 2 (n=2): skip 1 bit (1=n-1), check 2 bits (n), skip 2 bits (n), check 2 bits (n), skip 2 bits (n), etc.**
  - **Position 4 (n=4): skip 3 bits (3=n-1), check 4 bits (n), skip 4 bits (n), check 4 bits (n), skip 4 bits (n), etc.**
  - **Position 8 (n=8): skip 7 bits (7=n-1), check 8 bits (n), skip 8 bits (n), check 8 bits (n), skip 8 bits (n), etc.**
  - **Position 16 (n=16): skip 15 bits (15=n-1), check 16 bits (n), skip 16 bits (n), check 16 bits (n), skip 16 bits (n), etc.**
  - **Position 32 (n=32): skip 31 bits (31=n-1), check 32 bits (n), skip 32 bits (n), check 32 bits (n), skip 32 bits (n), etc.**
  - **General rule for position *n*: skip *n*-1 bits, check *n* bits, skip *n* bits, check *n* bits...**
  - **And so on.**

Algorithm (Cont.)

- The parity bit at position 2^k checks bits in positions having bit k set in their binary representation.

- For instance, bit 13, i.e. $1101_{(2)}$, is checked by bits $1000_{(2)} = 8$, $0100_{(2)}=4$ and $0001_{(2)} = 1$.

Additional Parity Bit

- By adding another parity bit to the coded word, the Hamming code can be used to correct a single error and detect double errors.
- Appending P_{13} to the previous 12-bit coded word becomes 001110010100P_{13}, where P_{13} is evaluated from the XOR of the other 12 bits.

Additional Parity Bit (Cont.)

- This produces 0011100101001 (even parity)

If $C = 0$ and $P = 0$	No error occurred.
If $C \neq 0$ and $P = 1$	A single error occurred that can be corrected.
If $C \neq 0$ and $P = 0$	A double error occurred that is detected but cannot be corrected.
If $C = 0$ and $P = 1$	An error occurred in the P_{13} bit.

Note that this scheme will detect more than two erroneous bits in many cases, but is not guaranteed to detect all such errors.