Overview of Hamming Code Error Detection and Correction

*Note: Sub-titles are not captured in Xplore and should not be used

Christian Kfoury
line 2: *dept. name of organization (of Affiliation)*
line 3: *name of organization (of Affiliation)*
Montreal, QC., Canada
line 5: email address or ORCID

Pratham Patel
line 2: *dept. name of organization (of Affiliation)*
line 3: *name of organization (of Affiliation)*
Montreal, QC., Canada
pratham.patelpr@gmail.com

Ali Zoubeidi
line 2: *dept. name of organization (of Affiliation)*
line 3: *name of organization (of Affiliation)*
Montreal, QC., Canada
line 5: email address or ORCID

***Abstract*—**

I. Introduction

In today's society, digital technology has rapidly grown and has become a very essential component of our everyday lives. With this increase in digitalization, there is a significant amount of data generated every single day. All data must be stored and transmitted efficiently to ensure a user can access it whenever needed, however corruption of data can occur as data recording or transmission is never hundred percent perfect. The following error can occur because of the possibility of the noise being added to the data when the data is being transmitted from the transmitter and is received by a receiver [1].

Therefore, hamming code was introduced to avoid any distortion of data by Richard Hamming [2…Reference needed...]. Hamming code is an error detection and correction used in the digital communication system. [....can add more here.....]. In this report, we will provide an overview of the Hamming code and also implement it using MARIE machine architecture and assembly language. We will give a detailed calculation of the minimum number of parity bits needed for a message 2-byte length and discuss the details of how a sender generates the parity bits and how the receiver detects and corrects a 1-bit error.

II. DISCUSSION

A. *Calculating the Minimum Number of Parity Bits Needed*

The Hamming code uses parity bits to add redundancy to the data bits, to make it possible to detect and correct errors in the data that has been transmitted. […].

To calculate the minimum amount of parity bits needed it must follow some steps.

1. Determine the number of data bits to be transmitted or stored. This can be done by measuring the length of the sender message.
2. Calculate the number of parity bits needed according to the length of the sender message using the following formula:

   $$2^r \geq m + r + 1$$

   Where:

   r = redundancy

   m = number of data bits
3. Choose the smallest value of redundancy (r) that satisfies the equation above.

Example

Assume the sender message length is 2 bytes.

Step 1: Determine number of data bits.

The number of data bits is 16 bits, as 1 byte is equivalent to 8 bits, therefore 2 byte is 16 bits.

Step 2: Calculate number of parity bits needed.

We use the following formula:

$$2^r \geq m + r + 1$$

$$2^r \geq 16 + r + 1$$

$$2^r \geq 17$$

Step 3: Choose the smallest value of redundancy.

If r = 0 → $2^0 = 1 \geq 17 + 0\ (false)$

If r = 1 → $2^1 = 2 \geq 17 + 1\ (false)$

If r = 2 → $2^2 = 4 \geq 17 + 2\ (false)$

If r = 3 → $2^3 = 8 \geq 17 + 3\ (false)$

If r = 4 → $2^4 = 16 \geq 17 + 4\ (false)$

If r = 5 → $2^5 = 32 \geq 17 + 5\ (true)$

Therefore, the smallest value of redundancy that satisfies the inequality is 5.

B. *How Sender Generates the Parity Bits*

Step 1: After establishing the number of parity bits, those parity bits need to be assigned to all the bit positions. To do this, each bit position can be expressed as the sums of those number in the form of power of 2.

$1 = 2^0$

$2 = 2^1$

$3 = 2^1 + 2^0$

$4 = 2^2$

$5 = 2^2 + 2^0$

$6 = 2^2 + 2^1$

$7 = 2^2 + 2^1 + 2^0$

$8 = 2^3$

$9 = 2^3 + 2^0$

$10 = 2^3 + 2^1$

$11 = 2^3 + 2^1 + 2^0$

$12 = 2^3 + 2^2$

$13 = 2^3 + 2^2 + 2^0$

$14 = 2^3 + 2^2 + 2^1$

$15 = 2^3 + 2^2 + 2^1 + 2^0$

$16 = 2^4$

$17 = 2^4 + 2^0$

$18 = 2^4 + 2^1$

$19 = 2^4 + 2^1 + 2^0$

$20 = 2^4 + 2^2$

$21 = 2^4 + 2^2 + 2^0$

Step 2: Write down which bits do the parity bits check out.

As seen on the previous step, we can see that:

Bit 1 checks: 3, 5, 7, 9, 11, 13, 15, 17, 19, 21

Bit 2 checks: 3, 6, 7, 10, 11, 14, 15, 18, 19

Bit 3 checks: 5, 6, 7, 12, 13, 14, 15, 20, 21

Bit 4 checks: 9, 10, 11, 12, 13, 14, 15

Bit 5 checks: 17, 18, 19, 20, 21

Step 3: XOR each bit to perform even parity calculation and insert the result of each parity bit in their corresponding parity bit position.

As an example, let’s the 2-byte data:

1	0	0	0	1	P	1	0	1	1	0	1	0	P	1	1	0	P	0	P	P
21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1

Bit 1: 0 XOR 0 XOR 1 XOR 0 XOR 0 XOR 1 XOR 1 XOR 1 XOR 1 = 1

Bit 2: 0 XOR 1 XOR 1 XOR 1 XOR 0 XOR 0 XOR 1 XOR 0 XOR 0 = 0

Bit 3: 0 XOR 1 XOR 1 XOR 1 XOR 1 XOR 0 XOR 1 XOR 0 XOR 1 = 0

Bit 4: 0 XOR 1 XOR 0 XOR 1 XOR 1 XOR 0 XOR 1 = 0

Bit 5: 1 XOR 0 XOR 0 XOR 0 XOR 1 = 0

The final message is:

1	0	0	0	1	0	1	0	1	1	0	1	0	0	1	1	0	0	0	0	1
21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1

C. *How Receiver Detects and Corrects a 1-bit error*

Let’s introduce an error in bit position b16, the code word would be the following 1 0001 1101 1010 0110 0001.

1	0	0	0	0	0	1	0	1	1	0	1	0	0	1	1	0	0	0	0	1
21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1

Step 1: For the receiver to detect the error write down which bits do the parity bits check out.

We can see that:

Bit 1 checks: 3, 5, 7, 9, 11, 13, 15, 17, 19, 21

Bit 2 checks: 3, 6, 7, 10, 11, 14, 15, 18, 19

Bit 3 checks: 5, 6, 7, 12, 13, 14, 15, 20, 21

Bit 4 checks: 9, 10, 11, 12, 13, 14, 15

Bit 5 checks: 17, 18, 19, 20, 21

Step 2: XOR each bit to perform even parity calculation and insert the result of each parity bit in their corresponding parity bit position.

Bit 1: 0 XOR 0 XOR 1 XOR 0 XOR 0 XOR 1 XOR 1 XOR 0 XOR 0 XOR 1 = 0

Bit 2: 0 XOR 1 XOR 1 XOR 1 XOR 0 XOR 0 XOR 1 XOR 0 XOR 0 = 0

Bit 3: 0 XOR 1 XOR 1 XOR 1 XOR 1 XOR 0 XOR 1 XOR 0 XOR 1 = 0

Bit 4: 0 XOR 1 XOR 0 XOR 1 XOR 1 XOR 0 XOR 1 = 0

Bit 5: 0 XOR 0 XOR 0 XOR 0 XOR 1 = 1

Bit 5 has produced an error.

Step 3: Detecting the single-bit

By looking at Parity Bit 1 and Bit 5, we can see that it shows errors. These two parities show both check 17, 19, 21, so the single bit error must be in either bit 17, bit 19 or bit 21.

However, since bit 2 checks bit 19 and bit 3 checks bit 21 and indicates no error in the subset of bits, therefore the error must be in bit 17.

III. Conclusion

In conclusion, Hamming code is a very useful technique used in digital communication, in order to avoid corruption of data. As shown in the report, Hamming code simply, adds extra bits to the data words [...]

Acknowledgment *(Heading 5)*

The preferred spelling of the word "acknowledgment" in America is without an "e" after the "g". Avoid the stilted expression "one of us (R. B. G.) thanks ...". Instead, try "R. B. G. thanks...". Put sponsor acknowledgments in the unnumbered footnote on the first page.

References

[1] Hajjdiab, H. 2023. COMP-228: System Hardware. Chapter 2: Data Representation in Computer Systems. Concordia University, Montreal, Canada, 2023. [PowerPoint].

[2] 2