第08次組語實習課

學生：林培瑋

2023 Advanced Mixed-Operation System (AMOS) Lab.

Tamkang University
Department of Electrical and Computer Engineering
No.151, Yingzhuan Rd., Tamsui Dist., New Taipei City 25137, Taiwan (R.O.C.)

EXAMPLE 7.7 HAMMING CODES

2023 Advanced Mixed-Operation System (AMOS) Lab.

Tamkang University
Department of Electrical and Computer Engineering
No.151, Yingzhuan Rd., Tamsui Dist., New Taipei City 25137, Taiwan (R.O.C.)

HAMMING CODES

- 錯誤檢測： 漢明碼能夠檢測傳輸過程中發生的位元錯誤。通過在原始數據中**添加校驗位元**，接收方可以檢查這些校驗位元以確定數據是否受到損壞。
- 錯誤糾正： 除了檢測錯誤，漢明碼還能夠糾正特定數量的位元錯誤。這種能力使得即使在數據傳輸過程中發生了一些錯誤，系統仍然能夠正確地**還原原始數據**。
- 通信系統： 在通信系統中，漢明碼常用於**提高數據的可靠性**。當訊號在傳輸過程中受到干擾或衰減時，漢明碼可以幫助檢測和糾正可能發生的位元錯誤。
- 儲存系統： 在儲存媒體上，如硬碟驅動器或閃存，漢明碼也被廣泛應用。這有助於確保儲存的數據在讀取時不受損壞，提高儲存系統的可靠性。

Parity bit(check bit)

Consider the idea of adding a bit, called a checksum, to a value that indicates the *parity* of the bits in that value. For example, if you had the 7-bit number

1010111

and we counted the number of ones in the value, 5 in this case, adding a 1 at the beginning of the value would make the parity *even*, since the number of ones (including the parity bit) is an even number. Our new value would be

11010111

7位資料（1的個數）	帶有核對位元的位元組	
	偶核對位元	奇核對位元
0000000（0）	00000000	00000001
1010001（3）	10100011	10100010
1101001（4）	11010010	11010011
1111111（7）	11111111	11111110

Even parity & Odd parity

- Even parity(偶數檢查位元)

代表給定的數值的二進位之中，值為1的個數為偶數個

- Odd parity(奇數檢查位元)

代表給定的數值的二進位之中，值為1的個數為奇數個

p.129 EXAMPLE 7.7 HAMMING CODES

If the data were transmitted this way, the receiver could detect an error in the byte sent if one of the data bits changes, since the parity would suddenly become *odd*. Note that if two of the bits changed, then we could not detect an error, since the parity remains even.

One type of Hamming code can be constructed by using four checksum bits placed in strategic locations. If a 12-bit value is constructed using 8 bits of data and four checksum bits as shown below, then we can use the checksum bits to detect up to two errors in the data and even correct a single bit error.

EX：

C0	d0	d1	d3	d4	d6
1	0	0	1	0	0

(C0=d0 XOR d1 XOR d3 XOR d4 XOR d6)

Original 8-bit value

d_7	d_6	d_5	d_4	d_3	d_2	d_1	d_0

Modified 8-bit value

11	10	9	8	7	6	5	4	3	2	1	0
d_7	d_6	d_5	d_4	c_3	d_3	d_2	d_1	c_2	d_0	c_1	c_0

→C_0的結果能產生偶數個1

The checksum bits c_3, c_2, c_1, and c_0 are computed as follows:

0 1 3 4 6

Checksum bit c_0 should produce even parity for bits 0, 2, 4, 6, 8, and 10. In other words, we're checking a bit, skipping a bit, checking a bit, etc.

Checksum bit c_1 should produce even parity for bits 1, 2, 5, 6, 9, and 10. In other words, we're checking two bits, skipping two bits, checking two bits, etc.

Checksum bit c_2 should produce even parity for bits 3, 4, 5, 6, and 11. Now we're checking four bits, skipping four bits, etc.

Checksum bit c_3 should produce even parity for bits 7, 8, 9, 10, and 11.

p.129 EXAMPLE 7.7 HAMMING CODES

```
        AREA HAMMING, CODE

        ENTRY

; Registers used:
; R0 - temp
; R1 - used to hold address of data
; R2 - holds value to be transmitted
; R4 - temp
main
        MOV r2, #0                      ; clear out transmitting reg
        ADR r1, arraya                  ; start of constants
        LDRB r0, [r1]

        ;
        ; calculate c0 using bits       76543210
        ;                                * ** **
        ; even parity, so result of XORs is the value of c0
        ;
        MOV r4, r0                      ; make a copy
        EOR r4, r4, r0, ROR #1          ; 1 XOR 0
        EOR r4, r4, r0, ROR #3          ; 3 XOR 1 XOR 0
        EOR r4, r4, r0, ROR #4          ; 4 XOR 3 XOR 1 XOR 0
        EOR r4, r4, r0, ROR #6          ; 6 XOR 4 XOR 3 XOR 1 XOR 0
        AND r2, r4, #1                  ; create c0 -> R2
        ;
        ; calculate c1 using bits       76543210
        ;                                ** ** *

        MOV r4, r0
        EOR r4, r4, r0, ROR #2          ; 2 XOR 0
        EOR r4, r4, r0, ROR #3          ; 3 XOR 2 XOR 0
        EOR r4, r4, r0, ROR #5          ; 5 XOR 3 XOR 2 XOR 0
        EOR r4, r4, r0, ROR #6          ; 6 XOR 5 XOR 3 XOR 2 XOR 0
        AND r4, r4, #1                  ; isolate bit
        ORR r2, r2, r4, LSL #1          ; 7 6 5 4 3 2 c1 c0
```

p.129 EXAMPLE 7.7 HAMMING CODES

```
;
; calculate c1 using bits    76543210
;                             ** ** *

MOV r4, r0
EOR r4, r4, r0, ROR #2       ; 2 XOR 0
EOR r4, r4, r0, ROR #3       ; 3 XOR 2 XOR 0
EOR r4, r4, r0, ROR #5       ; 5 XOR 3 XOR 2 XOR 0
EOR r4, r4, r0, ROR #6       ; 6 XOR 5 XOR 3 XOR 2 XOR 0
AND r4, r4, #1               ; isolate bit
ORR r2, r2, r4, LSL #1       ; 7 6 5 4 3 2 c1 c0
;
; calculate c2 using bits    76543210
;                            *   ***
ROR r4, r0, #1               ; get bit 1
EOR r4, r4, r0, ROR #2       ; 2 XOR 1
EOR r4, r4, r0, ROR #3       ; 3 XOR 2 XOR 1
EOR r4, r4, r0, ROR #7       ; 7 XOR 3 XOR 2 XOR 1
AND r4, r4, #1               ; isolate bit
ORR r2, r2, r4, ROR #29      ; 7 6 5 4 c2 2 c1 c0
;
; calculate c3 using bits    76543210
;                            ****
ROR r4, r0, #4               ; get bit 4
EOR r4, r4, r0, ROR #5       ; 5 XOR 4
```



```
EOR r4, r4, r0, ROR #6       ; 6 XOR 5 XOR 4
EOR r4, r4, r0, ROR #7       ; 7 XOR 6 XOR 5 XOR 4
AND r4, r4, #1
;
; build the final 12-bit result
;
ORR r2, r2, r4, ROR #25      ; rotate left 7 bits
```

p.129 EXAMPLE 7.7 HAMMING CODES

```
        AND r4, r0, #1              ; get bit 0 from original
        ORR r2, r2, r4, LSL #2      ; add bit 0 into final
        BIC r4, r0, #0xF1           ; get bits 3,2,1
        ORR r2, r2, r4, LSL #3      ; add bits 3,2,1 to final
        BIC r4, r0, #0x0F           ; get upper nibble
        ORR r2, r2, r4, LSL #4      ; r2 now contains 12 bits
                                    ; with checksums
done    B      done
        ALIGN

arraya
        DCB  0xB5
        DCB  0xAA
        DCB  0x55
        DCB  0xAA

        END
```

Q&A

Thanks for your attention !!