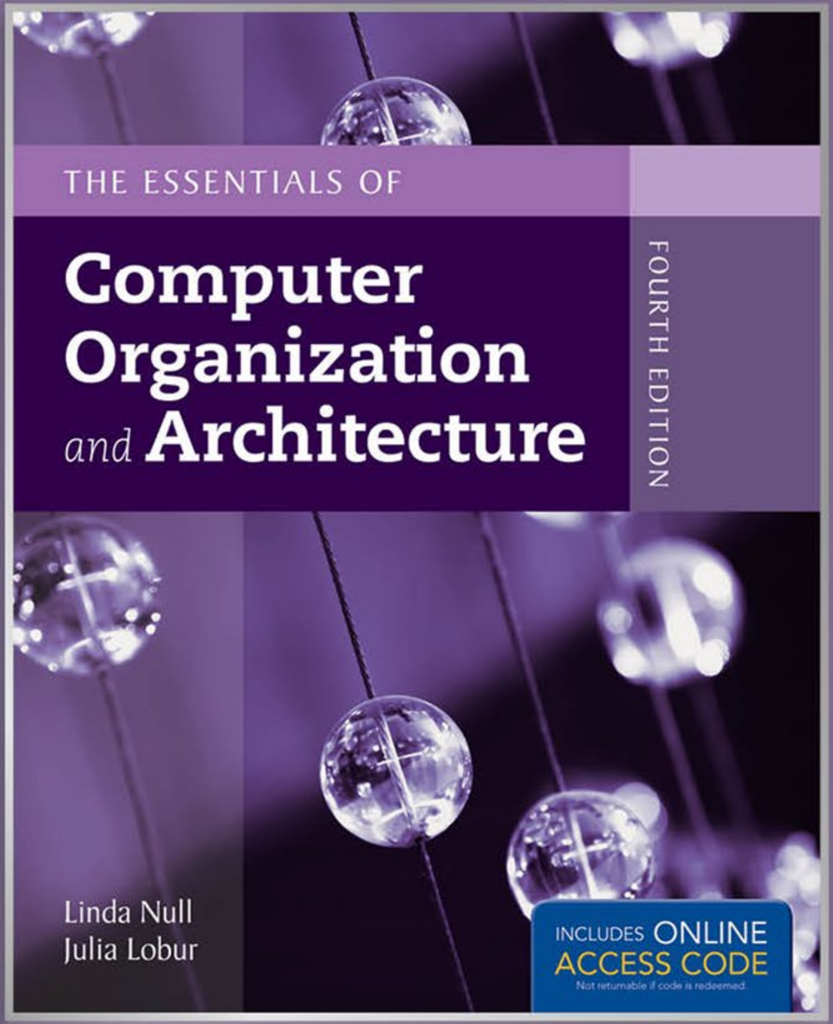
Chapter 2.7

Error Detection and Correction


THE ESSENTIALS OF
Computer Organization and Architecture
FOURTH EDITION
Linda Null
Julia Lobur
INCLUDES ONLINE ACCESS CODE
Not returnable if code is redeemed.

2.7 Error Detection and Correction

- It is physically impossible for any data recording or transmission medium to be 100% perfect 100% of the time over its entire expected useful life.
- As more bits are packed onto a square centimeter of disk storage, as communications transmission speeds increase, the likelihood of error inevitably increases.
- Thus, error detection and correction is critical to accurate data transmission, storage and retrieval.
- Check digits, appended to the end of a long number, can provide some protection against data input errors.
  - The last characters of ISBNs and UP student numbers are check digits.

2.7 Parity

- The simplest error detection method is that of parity.
  - Given a bit string of length N
    - We rather transmit a string of length N+1
    - The highest order bit, the parity bit, is used to store additional information about the lower order bit.
  - Two options
    - Odd parity:
    - If the number of 1's in the N-bits is odd then the parity bit is 0, else 1
    - Even Parity:
    - If the number of 1's in the N-bits is even then the parity bit is 0, else 1

2.7 Parity

- Assume even parity is being used:
- Original string: 111 0101
  - # of 1’s is odd so parity bit is 1
- Sent string is: 1111 0101
  - Observe that the number of 1’s are now even.

2.7 Parity

- While the use of a parity bit is simple it has two major issues:
  - It can only usually detect at most 1 error!
  - It only tells that an error occurred, not where the error occurred.

2.7.1 Cyclic Redundancy Check

- A cyclic redundancy check (CRC) is a type of checksum that is used in error detection for large blocks of data.
- Checksums and CRCs are examples **of *systematic error*** *detection*.
- In ***systematic error detection*** a group of error control bits is appended to the end of the block of transmitted data.
  - This group of bits is called a ***syndrome***.
  - Original information is unchanged by the addition of the error-checking bits.
- CRCs use polynomials over the modulo 2 arithmetic field.

The mathematical theory behind modulo 2 polynomials is beyond our scope. However, we can easily work with it without knowing its theoretical underpinnings.

$x^3 + x^1 + 1$

1 0 1 1

$\downarrow$

x^2 is not there

0 - 0 = 0

0 - 1 = 1

1 - 0 = 1

1 - 1 = 0

```
              101
      1001 ) 10111010
             1001
             ----
              0101
ADD -->       0000
because       ----
does           1010
not divide     1001
into           ----
                01110
                 1001
                 ----
                 0111
```

$x^2 + 1$
1 0 1

bits that we add is n-1 times smalle

```
          1000
      ___________
101 | 10101010 00
      101
      ---
        101
        101
        -----
        00000 00
            000
            ---
            000

10101010 00 crc bits
```

```
               1
101 |  1 0 0 0 0   0 0
       1 0   1
       ---------
         0   1 0 0 0
             1 0   1
           -----------
           0 0 1 0 0
                 1 0   1
                 0 0   1
```

2.7.1 Cyclic Redundancy Check

- Modulo 2 arithmetic works like clock arithmetic.
- In clock arithmetic, if we add 2 hours to 11:00, we get 1:00.
- In modulo 2 arithmetic if we add 1 to 1, we get 0. The addition rules couldn't be simpler:

0 + 0 = 0	0 + 1 = 1
1 + 0 = 1	1 + 1 = 0

- Example: Find the sum of 1011_2 and 110_2 modulo 2.

```
1 0 1 1
0 1 1 0 +( mod 2)
---------
1 1 0 1
```

2.7.1 Cyclic Redundancy Check

- Example: Find the **quotient** and **remainder** when **1111101** is divided by **1101** in modulo 2 arithmetic.
  - As with traditional division, we note that the dividend is divisible once by the divisor.
  - We place the divisor under the dividend and perform modulo 2 subtraction.

```
              1
      _______________
1101 ) 1111101   Protect
       1101
       ----
       0010
```

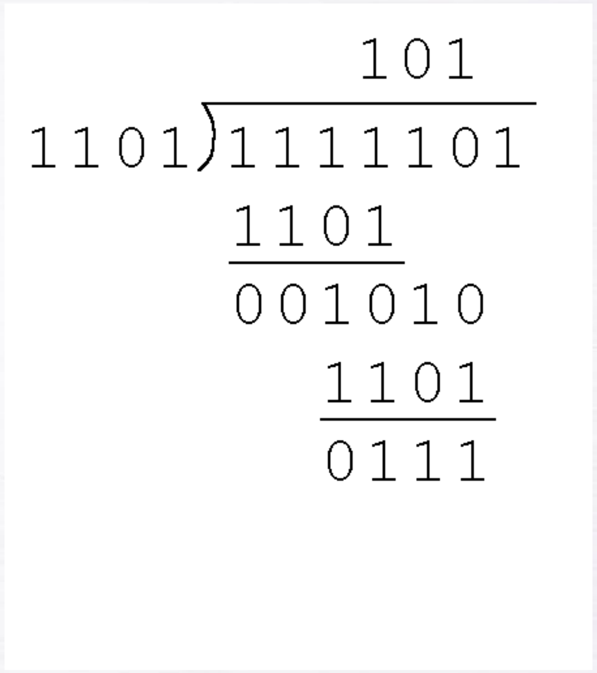
2.7.1 Cyclic Redundancy Check

- Find the quotient and remainder when 1111101 is divided by 1101 in modulo 2 arithmetic…
  - Now we bring down the next bit of the dividend.
  - We see that 00101 is not divisible by 1101. So we place a zero in the quotient.

```
          10
     ________
1101)1111101
     1101
     ----
     00101
```

2.7.1 Cyclic Redundancy Check

- Find the quotient and remainder when 1111101 is divided by 1101 in modulo 2 arithmetic…
  - 1010 is divisible by 1101 in modulo 2.
  - We perform the modulo 2 subtraction.



```
          101
     ________
1101)1111101
     1101
     ------
     001010
       1101
       ----
       0111
```

2.7.1 Cyclic Redundancy Check

- Find the quotient and remainder when 1111101 is divided by 1101 in modulo 2 arithmetic…
  - We find the quotient is 1011, and the remainder is 0010.
- This procedure is very useful to us in calculating CRC syndromes.

```
        1011
     _______
1101)1111101
     1101
     ----
     001010
       1101
       ----
       01111
        1101
        ----
        0010
```

Note: The divisor in this example corresponds to a modulo 2 polynomial: $\mathbf{X^3 + X^2 + 1}$**.**

2.7.1 Cyclic Redundancy Check

- How do we use this to perform error detection?
- Both parties decide on a CRC-Polynomial (P)
- The algorithm is then as follows for sender:
  - 1) Given a information block I, Left shift $|P| - 1$ bits.
  - 2) Perform modulo 2 division of I by P
    - Find the remainder, R
  - 3) Set $I = I + R$ (normal binary addition)
- The algorithm is the as follows for a receiver
  - 1) Receive I
  - 2) Perform modulo 2 division of I by P
  - 3) If $R = 0$ then transmission was successful
  - 4) Else there was an error.

2.7.1 Cyclic Redundancy Check (example)

- $I = 1001011_2$ and $P = 1011_2$
- $I = I \ll 3 = 1001011000_2$
- Module 2 divide I by P
  - $Q = 1010100$ and $R = 100$ (check at home)
- I= I + R= 1001011100
- Transmit.
  - Then if Module 2 divide I by P results in $R = 0$, transmission successful.
  - If $R \neq 0$ at least 1 bit error occurred.

2.7.1 Cyclic Redundancy Check

- Questions to think about
  - If P=11, how does CRC relate to parity?

2.7 Error Detection and Correction

- Data transmission errors are easy to fix once an error is detected.
  - Just ask the sender to transmit the data again.
- In computer memory and data storage, however, this cannot be done.
  - Too often the only copy of something important is in memory or on disk.
- Thus, to provide data integrity over the long term, error ***correcting*** **codes** are required.
  - We want to be able to detect and correct.
- **Hamming codes** and **Reed-Solomon** codes are two important error correcting codes.

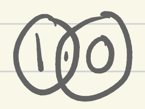
2.7.2 Hamming Codes

- Hamming codes are code words formed by adding redundant check bits, or parity bits, to a data word.
- The memory word itself consists of *m* bits, but *r* redundant bits are added to allow for error detection and/or correction.
- The *Hamming distance* between two code words is the number of bits in which two code words differ (3 in example).

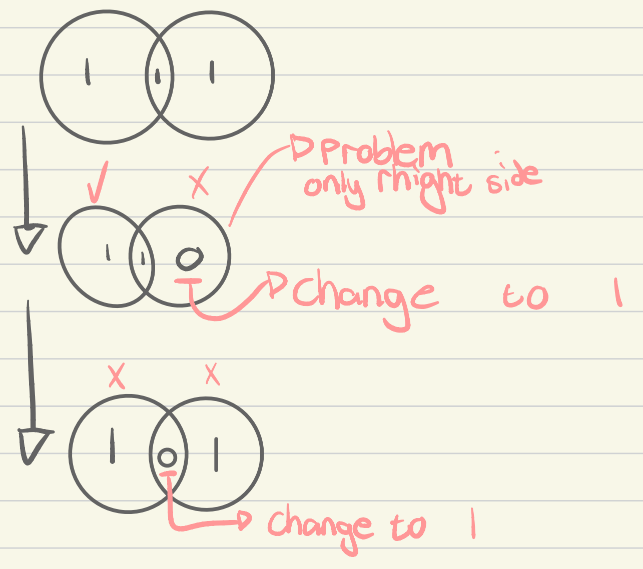
1	0	0	0	1	0	0	1
1	0	1	1	0	0	0	1

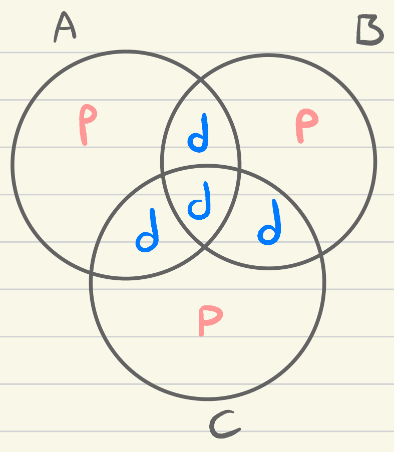
- The minimum Hamming distance for a code is the smallest Hamming distance between *all* pairs of words in the code.
- Example:
  - Assume memory with 2 data bits and 1 parity bit (appended at the end) that uses even parity (number of 1s in code word must be even).
  - Possible words in the code: 000, 011, 101, 110 (other possible bit patterns are invalid).
  - What is the minimum Hamming distance for this code?

Hamming



not working





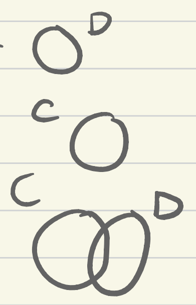
d=databit
P = Paritybits

2.7.2 Hamming Codes

- The minimum Hamming distance for a code, **D(min)**, determines its error detecting and error correcting capability.
- For a single-parity bit code:
  - Only single-bit errors can be detected, because D(min) is 2 (two bit errors can convert a valid code into another valid code).
- In general (for detection):
  - For any code word, *X,* to be interpreted as a different valid code word, *Y*, at least D(min) single-bit errors must occur in *X*.
  - Thus, to detect *k* (or fewer) single-bit errors, the code must have a Hamming distance of D(min) = *k* + 1.
  - Conversely, Hamming codes can always detect D(min) – 1 errors.
- For correction:
  - Hamming codes can correct $\left\lfloor \frac{D(\min) - 1}{2} \right\rfloor$ errors.
  - Or, the Hamming distance of a code must be at least 2*k* + 1 for it to be able to correct *k* errors.

Even with *k* errors, the word will still be closest to the original word.

A	B	C	D	
0	0	0	1	=
0	0	1	0	=
0	0	1	1	=
0	1	0	0	
0	1	0	1	
0	1	1	1	



15 − 4 = 11 bits

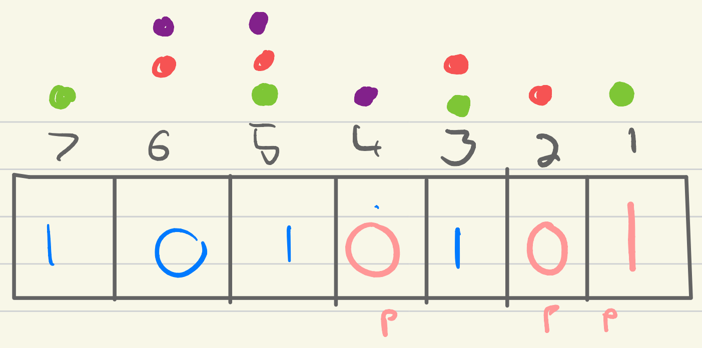
31 − 5 = 26

$2^{\text{Parity bits}} - 1 = n$

n − Parity bits = data bits

1011 evenparity

		A	B	C
p	1	0	0	1
p	2	0	1	0
	3	0	1	1
p	4	1	0	0
	5	1	0	1
	6	1	1	0
	7	1	1	1



7	6	5	4	3	2	1
1	0	1	0	1	0	1
			p		p	p

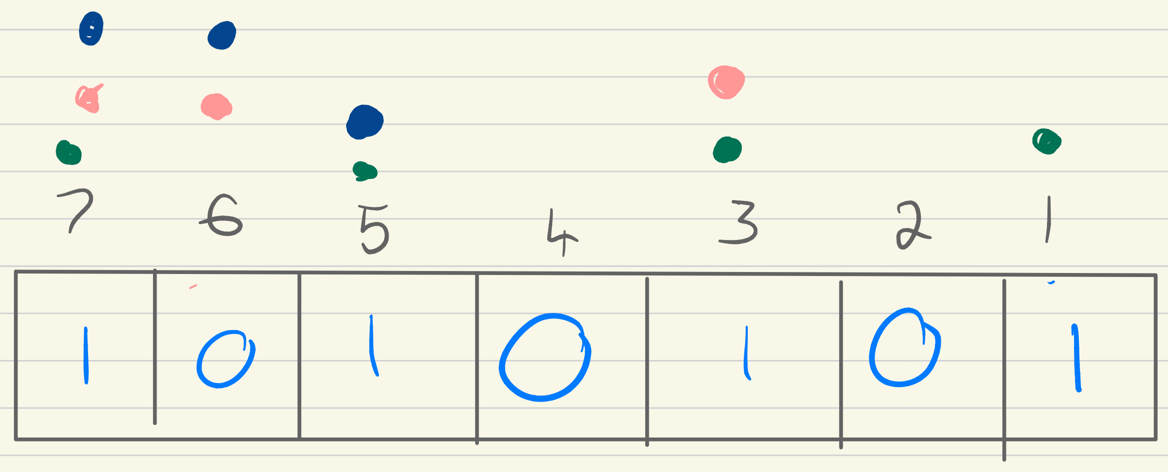
set C
looks at all the odd numbers

B look where ther is a one

A look for 1

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

3 bits $2^3 - 1 = 7$ bits 3 parity



P	1	0	0	1
P	2	0	1	0
	3	0	1	1
P	4	1	0	0
	5	1	0	1
	6	1	1	0
	7	1	1	1

$\underline{1011}$

$2^n - 1 - n$

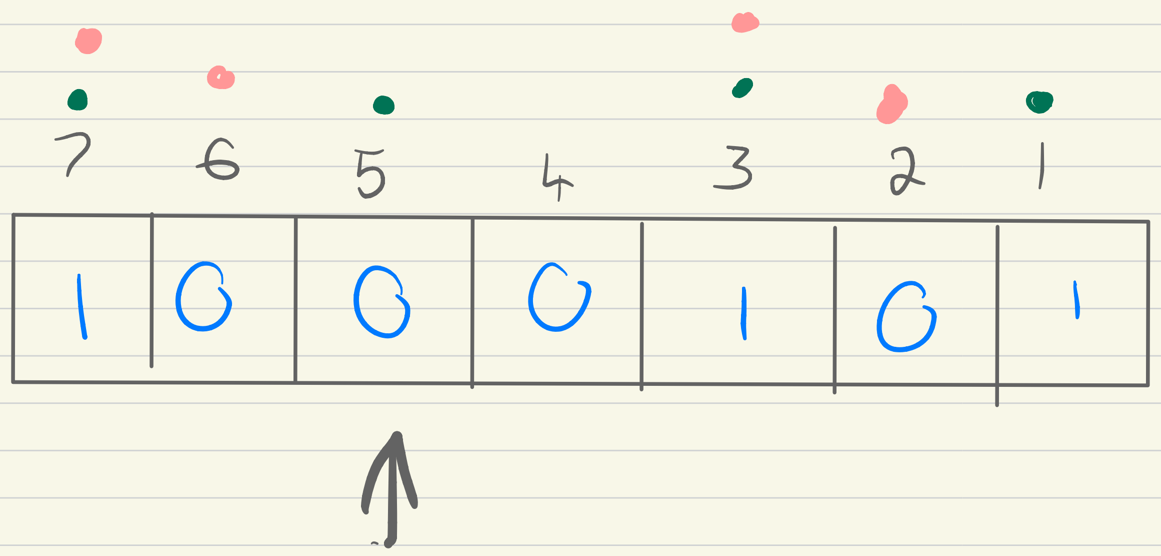
$= 2^3 - 1 - 3$

$= 4$

4 2 1

1 0 1

4 0 1 = 5

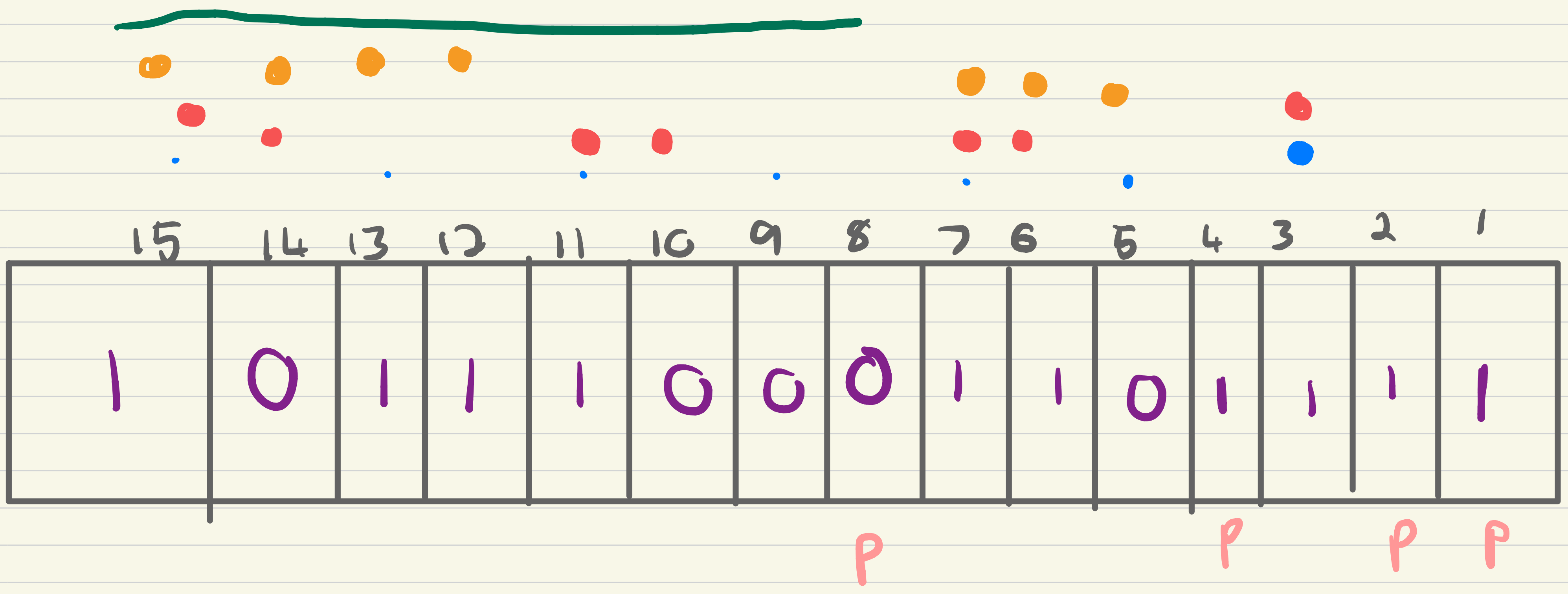


4 bit parity Check

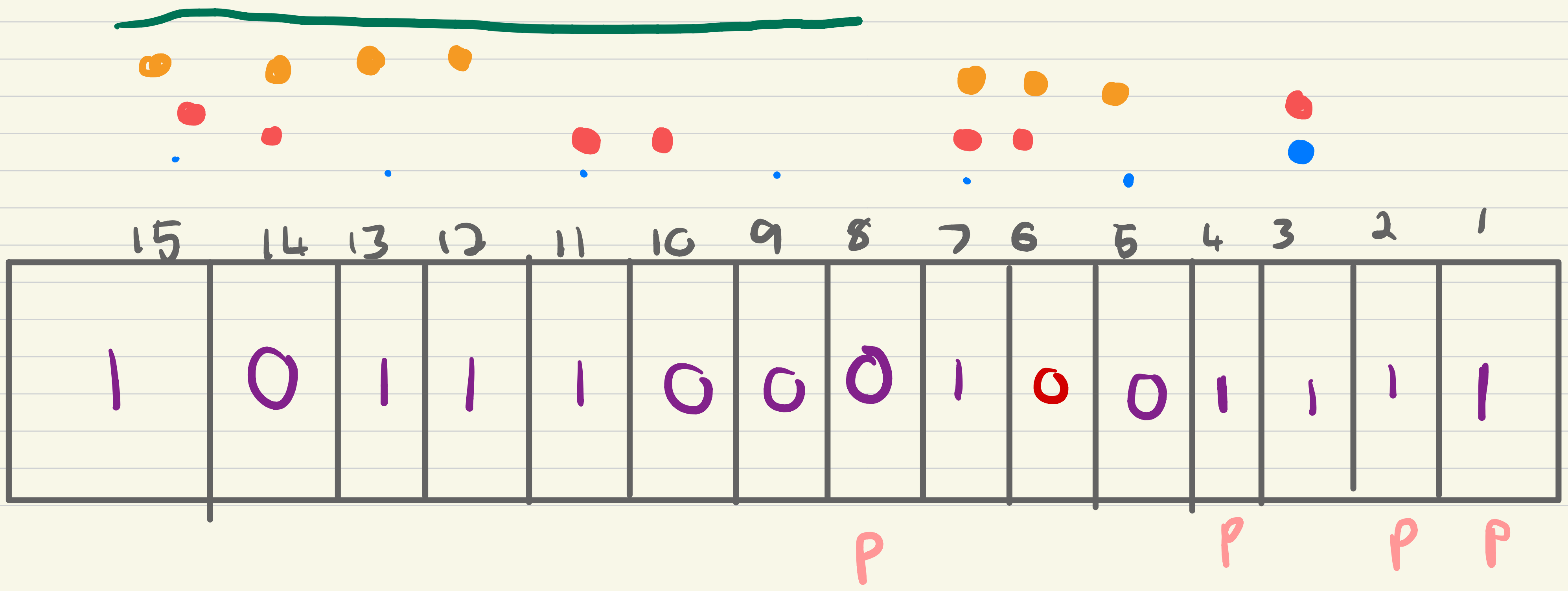
1011 1001 101

$$2^4 - 1 - 4 = 11 \text{ bits}$$

1	0	0	0	1	P
2	0	0	1	0	P
3	0	0	1	1	
4	0	1	0	0	P
5	0	1	0	1	
6	0	1	1	0	
7	0	1	1	1	
8	1	0	0	0	P
9	1	0	0	1	
10	1	0	1	0	
11	1	0	1	1	
12	1	1	0	0	
13	1	1	0	1	
14	1	1	1	0	
15	1	1	1	1	



0 1 1 0



Example 2.41

- Suppose we have the following code:

```
0 0 0 0 0
0 1 0 1 1
1 0 1 1 0
1 1 1 0 1
```

- What is D(min)?
  - Answer: D(min) = 3
- How many single-bit errors can be detected?
  - Answer: up to 2 single-bit errors
- How many single-bit errors can be corrected?
  - Answer: 1 single-bit errors can be corrected
- What is the correct code of the invalid code word 10000?
  - Difference vector to all code words: [1, 4, 2, 3], so correction is 00000 (closest code word). Assumption: the minimum number of possible errors has occurred. Correction might not be correct!
- What is the correct code of the invalid code word 11000?

Distance vector: [2, 3, 3, 2]. Cannot make the correction.

2.7.2 Hamming Codes
Single bit correction

- Suppose we have a set of *n*-bit code words consisting of *m* data bits and *r* (redundant) check bits.
- How many legal code words?
  - Answer: 2^m
- How many illegal code words at a Hamming distance of 1 from each valid code word (i.e. bit strings with single-bit errors)?
  - Answer: *n* (an error can occur in any of the *n* bit positions)
- How many total bit patterns (valid and invalid) possible?
  - Answer: 2^n or $2^{(m+r)}$
- For each valid codeword, we have (*n*+1) bit patterns (1 legal and *n* illegal). This gives us the inequality:

$$(n + 1) \times 2^m \leq 2^n$$

2.7.2 Hamming Codes

- From previous slide:

$$(n + 1) \times 2^{m} \leq 2^{n}$$

- Because $n = m + r$, we can rewrite the inequality as:

$$(m + r + 1) \times 2^{m} \leq 2^{m + r}$$

or $$(m + r + 1) \leq 2^{r}$$

- This inequality gives us a lower limit on the number of check bits that we need to construct a code with *m* data bits and *r* check bits that corrects all single-bit errors.
- For example, say we have data words of length 4. What is the minimum number of check bits needed for correcting single-bit errors?

$$(4 + r + 1) \leq 2^{r}$$

r must be greater than or equal to 3.

- We should always use the smallest value of *r* that makes the inequality true.

2.8 Error Detection and Correction

- For example, say we have data words of length 8. What is the minimum number of check bits needed for correcting single-bit errors?

  $(8 + r + 1) \leq 2^{r}$

  r must be greater than or equal to 4.

- How long will the resulting code words be?

  Answer: 12

- So how do we assign values to check bits so that we can achieve detection and correction?

The Hamming Algorithm

1. Determine the number of check bits, r, necessary for the code and then number the n bits, right to left, starting with 1 (not 0).
2. Each bit whose bit number is a power of 2 is a parity bit (positions 1,2,4,8,etc) and the others are data bits (positions 3,5,7,9,10,etc).
3. Each parity bit calculates the parity of 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: check 1 bit, skip 1 bit, check 1 bit, skip 1 bit, etc. (1, 3, 5, 7, 9, …)

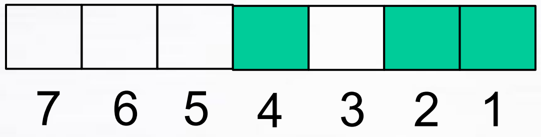
   Position 2: check 2 bits, skip 2 bits, check 2 bits, etc. (2,3,6,7,10,11,…)

   Position 4: check 4 bits, skip 4 bits, check 4 bits, etc. (4,5,6,7,12,13,14,15,20,...)

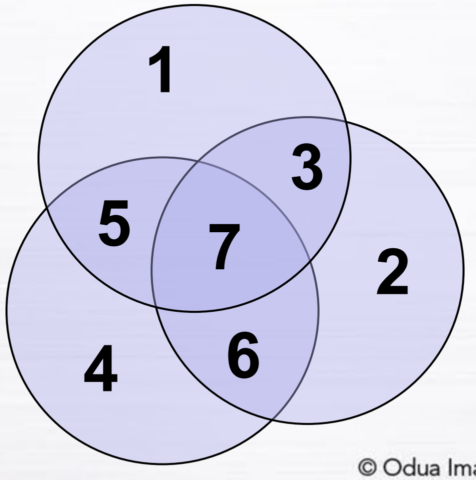
   …and so on.
4. Set each parity bit to 1 if the number of 1's in the positions it checks is odd, or to 0 if the number of 1's in the position it checks is even (even parity).

2.7.2 Hamming Codes

- Consider the example with words of length 4, with 3 check bits.
- Code words will be of length 7 (white cells are data and green cells are parity bits):



  - The set of bits using parity bit 1: 1, 3, 5, 7 (check 1, skip 1, …)
  - The set of bits using parity bit 2: 2, 3, 6, 7 (check 2, skip 2, …)
  - The set of bits using parity bit 4: 4, 5, 6, 7 (check 4)



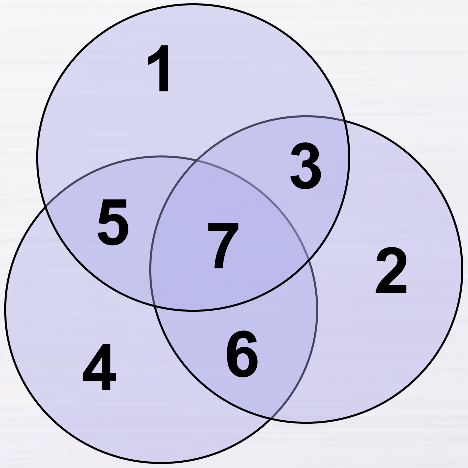
Notice that the parity bits (1, 2, 4) are each only in one set.
Bit position 7 is in all three sets.
Using this system, if a bit changes, it can be deduced which one changed.

2.7.2 Hamming Codes

- Example: Determine the Hamming code word for 4-bit data 1101.
- Code words will be of length 7 (white cells are data and green cells are parity bits):

1	1	0	🟩	1	🟩	🟩
7	6	5	4	3	2	1

  - The set of bits using parity bit 1: 1, 3, 5, 7 (check 1, skip 1, …)
  - The set of bits using parity bit 2: 2, 3, 6, 7 (check 2, skip 2, …)
  - The set of bits using parity bit 4: 4, 5, 6, 7 (check 4)
- Determine the parity bit values:
  - Parity bit 1: value is set to 0 (since 3=1, 5=0, 7=1)
  - Parity bit 2: value is set to 1 (since 3=1, 6=1, 7=1)
  - Parity bit 4: value is set to 0 (since 5=0, 6=1, 7=1)
- Hamming code word: 1100110
- How to determine where an error occurred?
  - Add the positions of parity bits that produce the error!



2.7.2 Hamming Codes

- What is the Hamming code of data word 1101 0110?
- Data word of length 8 requires 4 parity bits.
- Which will be the check (parity) bits?

				P				P		P	P
12	11	10	9	8	7	6	5	4	3	2	1

- What is the full Hamming code?

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

2.7.2 Hamming Codes

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

- There is an error in the code word above. Correct it.
  - Bit 1 checks 1, 3, 5, 7, 9, and 11. (1,0,0,0,1,1)
    - *This is incorrect as we have a total of 3 ones (which is not even parity).*
  - Bit 2 checks bits 2, 3, 6, 7, 10, and 11. (0,0,1,0,0,1)
    - The parity is correct.
  - Bit 4 checks bits 4, 5, 6, 7, and 12. (1,0,1,0,1)
    - *This parity is incorrect, as we 3 ones.*
  - Bit 8 checks bit 8, 9, 10, 11, and 12. (1,1,0,1,1)
    - This parity is correct.
  - The common elements in the two sets are 5 and 7, but 7 is part of bit 2 check, so cannot be the problem.
  - Therefore, the bit that changed was 5, so flip the bit in position 5.
  - Alternatively, simply add 1 + 4 = 5, and flip bit 5.