San Francisco Bay University

CE305 - Computer Organization
2023 Fall Homework #2

Due day: 10/15/2023

Instruction:

1. Homework answer sheet should contain the original questions and corresponding answers.
2. Answer sheet must be in PDF file format with Github links for the programming questions, but MS Word file can't be accepted. As follows is the answer sheet name format.
<course_id>_week<week_number>_StudentID_FirstName_LastName.pdf
3. The program name in Github must follow the format like
<course_id>_week<week_number>_q<question_number>_StudentID_FirstName_LastName
4. Show screenshot of all running results, including the system date/time.
5. Only accept homework submission uploaded via Canvas.
6. Overdue homework submission can't be accepted.
3. Takes academic honesty and integrity seriously (Zero Tolerance of Cheating & Plagiarism)

1. Cyclic Redundancy Check (CRC) is one of the popular coding and decoding techniques in the data transmitted over the network for error detection and correction. Given $x^5 + x^2 + 1$ as a CRC generation polynomial from International Telegraph and Telephone Consultative Committee (CCITT), write the encoding and decoding *def* functions in Python for the only 4-bits original binary data. The examples and testcases of the encoding and decoding processes are shown as follows for your programming. After that, discuss how many bits errors CRC can detect.

```
def encoding(msg, poly):
  """
   org_sig1 = '1010'            # original binary data
   poly = '100101'              #  x^5 + x^2 + 1  =>  b5b4b3b2b1b0 = 100101
   encoding (org_sig1, poly) # find the reminder from 1010 00000 % 100101 = 00111
    '1010 00111'                # encoded output

   org_sig2 = '1100'            # original binary data
   poly = '100101'
   encoding (org_sig2, poly) # find the reminder from 1100 00000 % 100101 = 11001
    '1100 11001'                # encoded output
  """
```

def decoding(rcv, poly):

"""

received_sig1 = '1010 00111' # if receiving the data without error

poly = '100101' # $x^5 + x^2 + 1$ => $b_5 b_4 b_3 b_2 b_1 b_0 = 100101$

decoding (received_sig1, poly) # 1010 00111 % 100101 = 00000 (reminder is zero)

'No error'

received_sig2 = '1010 01111' # if receiving the data with 1-bit error

poly = '100101'

decoding (received_sig2, poly) # 1010 01111 % 100101 = 01000 (reminder is NOT zero)

'Error'

received_sig3 = '1100 11001' # if receiving the data without error

poly = '100101'

decoding (received_sig3, poly) # 1100 11001 % 100101 = 00000 (reminder is zero)

'No error'

received_sig4 = '1100 11111' # if receiving the data with 2-bits error

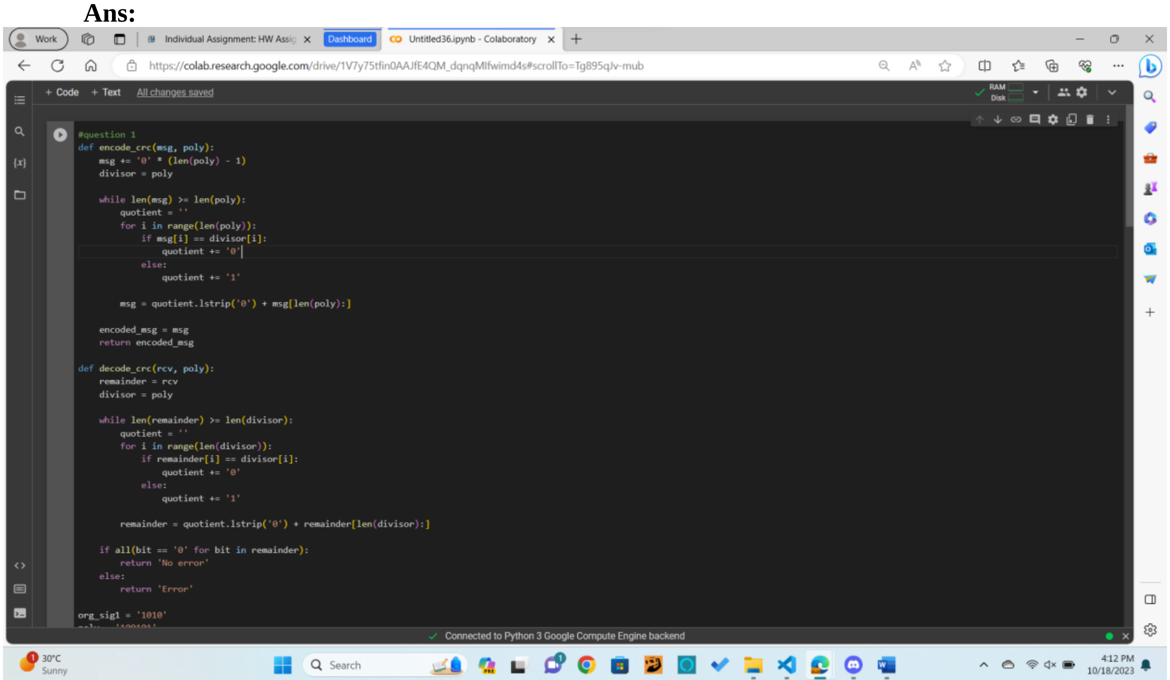
poly = '100101'

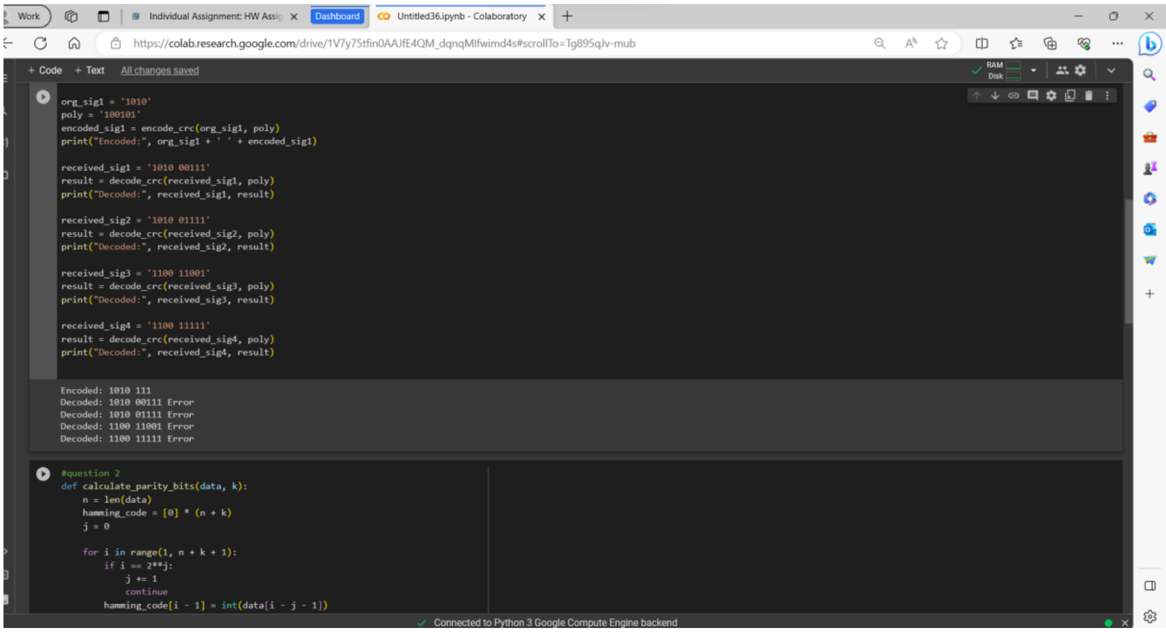
decoding (received_sig4, poly) # 1100 11111% 100101 = 00110 (reminder is NOT zero)

'Error'

"""

Ans:





```
org_sig1 = '1010'
poly = '100101'
encoded_sig1 = encode_crc(org_sig1, poly)
print("Encoded:", org_sig1 + ' ' + encoded_sig1)

received_sig1 = '1010 00111'
result = decode_crc(received_sig1, poly)
print("Decoded:", received_sig1, result)

received_sig2 = '1010 01111'
result = decode_crc(received_sig2, poly)
print("Decoded:", received_sig2, result)

received_sig3 = '1100 11001'
result = decode_crc(received_sig3, poly)
print("Decoded:", received_sig3, result)

received_sig4 = '1100 11111'
result = decode_crc(received_sig4, poly)
print("Decoded:", received_sig4, result)
```

```
Encoded: 1010 111
Decoded: 1010 00111 Error
Decoded: 1010 01111 Error
Decoded: 1100 11001 Error
Decoded: 1100 11111 Error
```

```
#question 2
def calculate_parity_bits(data, k):
    n = len(data)
    hamming_code = [0] * (n + k)
    j = 0

    for i in range(1, n + k + 1):
        if i == 2**j:
            j += 1
            continue
        hamming_code[i - 1] = int(data[i - j - 1])
```

2. Hamming code is one important error correcting code in computer science and telecommunication as well. Standard Hamming code can only detect and correct a **single bit** error. Encoding method is shown by an example as follows.

e.g.
Original data with 7 bits in binary: 1001011
Step1: Find how many extra parity bits are needed by the following inequality.

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

where *m* is the number of bits in original data, 7, and *k* is a positive integer as the number of the parity bits by trying the value from *1* until meet the inequality such as:

$2^1 \geq 7 + 1 + 1$ => Not True
$2^2 \geq 7 + 2 + 1$ => Not True
$2^3 \geq 7 + 3 + 1$ => Not True
$2^4 \geq 7 + 4 + 1$ => True

Therefore, *k* = 4, which means that 4-bits extra parities are needed to add to the original data 1001011

Step2: Find the bit position for the extra parities.
Since *k* = 4 in the example, the position extra parity bit should be at 2^{i-1},where *i* is from 1 to 4, i.e., $2^{1-1} = 1, 2^{2-1} = 2, 2^{3-1} = 4, 2^{4-1} = 8$

Bit Position	1	2	3	4	5	6	7	8	9	10	11
Parities and Original data	P_1	P_2	1	P_4	0	0	1	P_8	0	1	1
Labels of Original data			X_3		X_5	X_6	X_7		X_9	X_{10}	X_{11}

Step3: Calculate each parity bit.

- Create a table as follows.

Parity Positions		8	4	2	1
		P_8	P_4	P_2	P_1
X_3	3 in binary	0	0	1	1
X_5	5 in binary	0	1	0	1
X_6	6 in binary	0	1	1	0
X_7	7 in binary	0	1	1	1
X_9	9 in binary	1	0	0	1
X_{10}	10 in binary	1	0	1	0
X_{11}	11 in binary	1	0	1	1

- Get the calculation equations for each parity bit, such as P_1, P_2, P_4, P_8 by each X_i with 1's value in the columns.

$$
\begin{aligned}
P_1 &= X_3 \; xor \; X_5 \; xor \; X_7 \; xor X_9 \; xor \; X_{11} &&= 1 \oplus 0 \oplus 1 \oplus 0 \oplus 1 = 1 \\
P_2 &= X_3 \; xor \; X_6 \; xor \; X_7 \; xor X_{10} \; xor \; X_{11} &&= 1 \oplus 0 \oplus 1 \oplus 1 \oplus 1 = 0 \\
P_4 &= X_5 \; xor \; X_6 \; xor \; X_7 &&= 0 \oplus 0 \oplus 1 \qquad = 1 \\
P_8 &= X_9 \; xor \; X_{10} \; xor \; X_{11} &&= 0 \oplus 1 \oplus 1 \qquad = 0
\end{aligned}
$$

As follows are the final encoding results.

Bit Position	1	2	3	4	5	6	7	8	9	10	11
Parities and Original data	1	0	1	1	0	0	1	0	0	1	1
Labels of data	P_1	P_2	X_3	P_4	X_5	X_6	X_7	P_8	X_9	X_{10}	X_{11}

The decoding process is very similar to the encoding shown as follows with an example.

e.g., Sending bit stream is from the above example.

Bit Position	1	2	3	4	5	6	7	8	9	10	11
Parities and Original data	1	0	1	1	0	0	1	0	0	1	1
Labels of data	P_1	P_2	X_3	P_4	X_5	X_6	X_7	P_8	X_9	X_{10}	X_{11}

But the received message is a different one with a single bit error at Position 6, like the below.

Bit Position	1	2	3	4	5	6	7	8	9	10	11
Parities and Original data	1	0	1	1	0	1	1	0	0	1	1
Labels of data	P_1	P_2	X_3	P_4	X_5	X_6	X_7	P_8	X_9	X_{10}	X_{11}

Step1: Detect the error and position, like the following example.

$$
\begin{aligned}
h_1 &= P_1 \; xor \; X_3 \; xor \; X_5 \; xor \; X_7 \; xor X_9 \; xor \; X_{11} &&= 1 \oplus 1 \oplus 0 \oplus 1 \oplus 0 \oplus 1 = 0 \\
h_2 &= P_2 \; xor \; X_3 \; xor \; X_6 \; xor \; X_7 \; xor X_{10} \; xor \; X_{11} &&= 0 \oplus 1 \oplus 1 \oplus 1 \oplus 1 \oplus 1 = 1 \\
h_3 &= P_4 \; xor \; X_5 \; xor \; X_6 \; xor \; X_7 &&= 1 \oplus 0 \oplus 1 \oplus 1 \qquad = 1 \\
h_4 &= P_8 \; xor \; X_9 \; xor \; X_{10} \; xor \; X_{11} &&= 1 \oplus 0 \oplus 1 \oplus 1 \qquad = 0
\end{aligned}
$$

Step2: Correct the value from 1 to 0 at the error location Position 6 from $h_1 h_2 h_3 h_4 = 0110(binary)$. Of course, If $h_1 h_2 h_3 h_4 = 0000$, no error.

Based on the above encoding and decoding methods, write the program in Python to implement Hamming code algorithm.

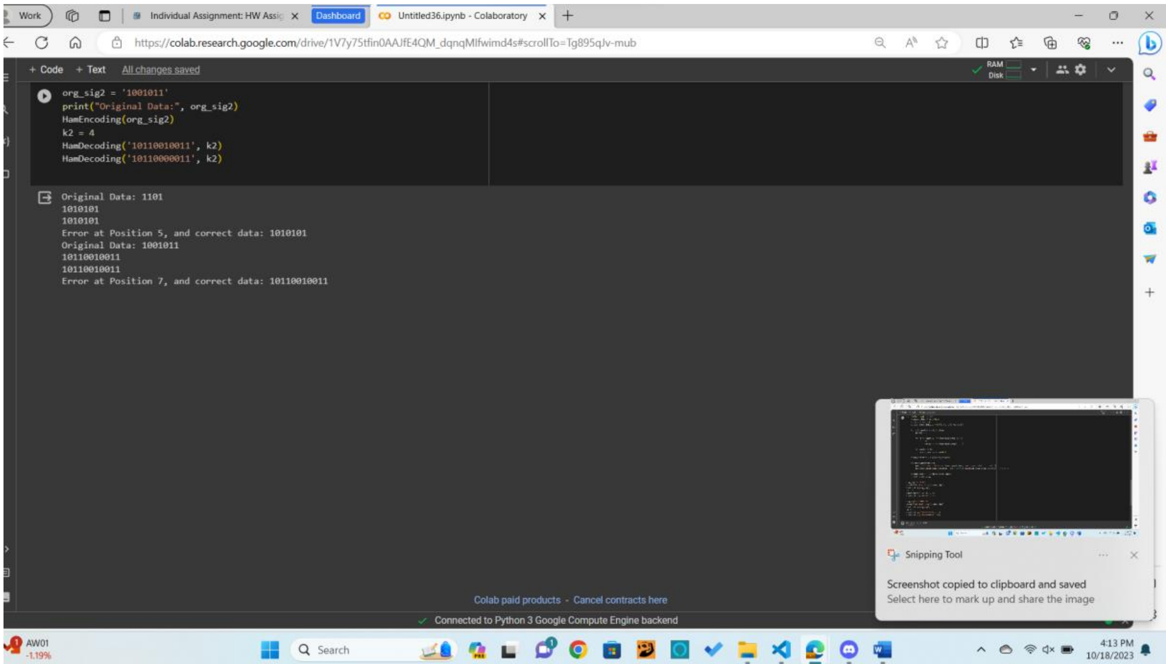
```
def HamEncoding(msg):
  """
   org_sig1 = '1101'          # original binary data
   HamEncoding(org_sig1)
    k = 3                     # need to show the number of extra parity bits
    '1010101'                 # encoded output

   org_sig2 = '1001011'       # original binary data
   HamEncoding (org_sig2)
    k = 4                     # need to show the number of extra parity bits
   '10110010011'              # encoded output

   """

def HamDecoding(rcv, k):
   """
   received_sig1 = '1010101'      # if receiving the data without error
   k = 3
   HamDecoding(received_sig1, k)
   'No error'

   received_sig2 = '1010001'      # if receiving the data with 1-bit error at Position 5
   k = 3
   HamDecoding(received_sig2, k)
   'Error at Position 5, and correct data: 1010101'


    received_sig3 = '10110010011' # if receiving the data without error
    k = 4
    HamDecoding(received_sig3, k)
    'No error'

    received_sig4 = '10110000011' # if receiving the data 1-bit error at Position 7
    k = 4
    HamDecoding(received_sig4, k)
    'Error at Position 7, and correct data: 10110010011

   """
```

Ans:

```python
#question 2
def calculate_parity_bits(data, k):
    n = len(data)
    hamming_code = [0] * (n + k)
    j = 0

    for i in range(1, n + k + 1):
        if i == 2**j:
            j += 1
            continue
        hamming_code[i - 1] = int(data[i - j - 1])

    for j in range(k):
        parity_bit_position = 2**j
        parity_bit = 0

        for i in range(1, n + k + 1):
            if i & parity_bit_position != 0:
                parity_bit ^= hamming_code[i - 1]

        hamming_code[parity_bit_position - 1] = parity_bit

    return ''.join(map(str, hamming_code))

def HamEncoding(msg):
    data = list(msg)
    k = 0

    while 2**k < len(data) + k + 1:
        k += 1

    encoded_msg = calculate_parity_bits(data, k)
    print(encoded_msg)

def HamDecoding(rcv, k):
    received_data = list(rcv)
    error_indices = []
    parity_check_indices = [2**i for i in range(k)]
```

```python
def HamDecoding(rcv, k):
    received_data = list(rcv)
    error_indices = []
    parity_check_indices = [2**i for i in range(k)]

    for i in parity_check_indices:
        parity = 0

        for j in range(1, len(received_data) + 1):
            if j & i:
                parity ^= int(received_data[j - 1])

        if parity != 0:
            error_indices.append(i)

    error_position = sum(error_indices)

    if error_position > 0:
        print(f'Error at Position {error_position}, and correct data: ', end='')
        received_data[error_position - 1] = str(int(received_data[error_position - 1]) ^ 1)

    decoded_msg = ''.join(received_data)
    print(decoded_msg)

org_sig1 = '1101'
print("Original Data:", org_sig1)
HamEncoding(org_sig1)
k1 = 3
HamDecoding('1010101', k1)
HamDecoding('1010001', k1)

org_sig2 = '1001011'
print("Original Data:", org_sig2)
HamEncoding(org_sig2)
k2 = 4
HamDecoding('10110010011', k2)
HamDecoding('10110000011', k2)
```

```
Original Data: 1101
```



Swekchha Hamal, 19700.