King Fahd University of Petroleum and Minerals
Computer Engineering Department
College of Computing and Mathematics

COE 301: Computer Organization
Assignment 2, Term 251
Error Detection using Cyclic Redundancy Check (CRC)

Overview:

Error detection is an important topic in data communication. Among the most popular techniques used in this regard is cyclic redundancy code (CRC) calculation. In this assignment, we want to implement a CRC generator and verifier using MIPS assembly languages.

The goal of this assignment is to protect an array of bytes, by generating the CRC code of the array, to detect, if possible, errors during data transmission. At the receiver, a similar operation will be performed to detect any errors\changes in the data during the data exchange.

Cyclic Redundancy Check (CRC)

Cyclic Redundancy Check (CRC) codes are among the best mechanisms available to detect and/or correct errors in communications transmissions. CRC uses polynomial modulo-2 division (different than regular integer division) and it can be done using shifting and XOR-ing. An n-bit CRC code is generated based on an (n+1)-bit divisor called generator polynomial. In this assignment, we will use a 9-bit generator to generate an 8-bit CRC codes. The generator polynomial that we are going to use is

$$\mathbf{G(x) = x^8 + x^7 + x^5 + x^2 + x^1 + x^0 = 110100111 = 0x1A7}$$

The steps used in CRC for one byte are summarized in the following.

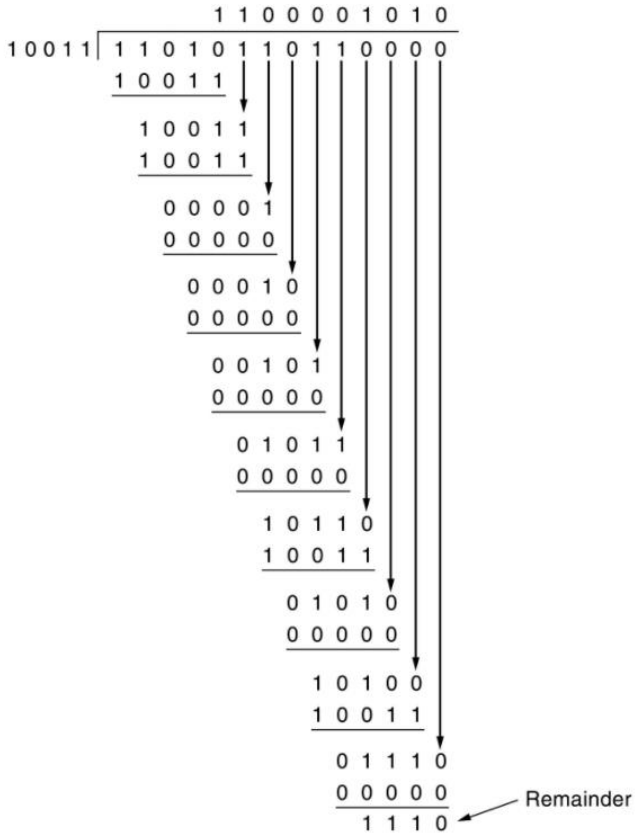
- A string of *n* zeros (0's) is appended to the data. The value of *n* in this assignment is 8.
- The newly extended data is divided by the generator using binary division (see below). The remainder generated from this division is the CRC code.
- The CRC code replaces the appended 0s at the end of the original data. This final unit (i.e. the data and the CRC) is the frame to be sent over the channel.
- The receiver treats the received frame (data + CRC) as one entity. This entity is divided by the same divisor that was used to generate the CRC. If the remainder of the division is zero, it means the received data is error-free. However, If the remainder is not zero this is interpreted as the existence of error, and the data is discarded.

CRC Generation

Binary division is used to calculate CRC. It is similar to integer division except that XOR operation is used instead of subtraction. Assume that we need to generate the CRC code of data = 1101011011 using generator =10011 (n=5)

1. The data is appended with 4 zeros to become 11010110110000
2. The appended data is divided by the generator as shown below. Notice that the subtractions in the division process are performed modulo 2, using exclusive-or (XOR).

3. The remainder (1110) is the CRC code used for error detection and it is appended to the message



The same procedure is also used for error detection. The data along with the CRC code are divided by the generator and the remainder is checked. If the remained is 0, then the data is error free, otherwise the data is corrupted (erroneous)

CRC for an Array of Bytes

To generate CRC for multiple elements in an array, the CRC of the first byte is generated then it is XORed with the second byte and the CRC code is regenerated for the XORed result. The same procedure is followed for the remain bytes/elements of the array and the CRC code of the last element is considered the CRC code for the whole array/ The following steps explain this process.

Assume data = 0x01, 0x02 and generator is 0x11D = $(1\ 0001\ 1101)_2$

1. Init crc = 0x00
2. *'Xor-in'* next input byte 0x01: 0x00 ⊕ 0x01 = 0x01.
3. Calculate CRC-8 of 0x01 using binary division ➔ crc = 0x1D.
4. *'Xor-in'* next input byte 0x02: 0x1D ⊕ 0x02 = 0x1F.
5. Calculate CRC-8 of 0x1F using binary division ➔ crc = 0x76.
6. Repeat steps 4 & 5 for all remaining bytes in the array

For error checking, the same procedure is followed except that the CRC value should be initialized to the CRC code instead of 0 (i.e. the data is appended with the CRC before the starting the binary division operation)

Tasks

You are required to implement the following functions:

1. Implement (MIPS) function **`crc_byte`** that generates the CRC code for a single byte according to the procedure described in the CRC generation section. The function should use the generator
   $G(x) = x^8 + x^7 + x^5 + x^2 + x^1 + x^0 = 110100111 = 0x1A7$
   The function takes the input data (byte), that it will generate CRC for, in register **$a0** and returns the generated CRC code in register $v0
2. Implement MIPS function **`crc_array`** that takes an array of bytes and returns the CRC code of the whole array as described in the (CRC for an Array of Bytes) section above. The function takes the following inputs:
   a. **$a0:** the address of the array (i.e. the memory address of the first element in the array)
   b. **$a1:** the number of elements in the array
   The function should call the function **`crc_byte`** and return the CRC code for the whole array in register $v0
3. Implement (MIPS) function **`check_byte`** that checks if a given byte + CRC is error-free or not. The function should use the generator
   $G(x) = x^8 + x^7 + x^5 + x^2 + x^1 + x^0 = 110100111 = 0x1A7$
   The function takes the input data (byte) in register **$a0**, the CRC code in register $a1 and returns true ($v0 = 1) if the input byte is error free, and false ($v0 = 0) otherwise. The function could use **`crc_byte`** function
4. Implement (MIPS) function **`check_array`** that checks if a given array + CRC is error-free or not. The function takes the following inputs:
   a. **$a0:** the address of the array (i.e. the memory address of the first element in the array)
   b. **$a1:** the number of elements in the array
   c. **$a2:** the CRC code
   The function returns true ($v0 = 1) if the input byte is error free, and false ($v0 = 0) otherwise

Submission Guidelines:

- The due date is **Saturday Nov. 1, 2025 end of the day**.
- The assignment can be solved individually or collaboratively in groups of two students. Both team members should be from the same lecture section and should be declared in Gradescope during submission.
- The submitted file should be renamed as "**assign2.asm**"
- Submit your assignment through Gradescope. Email submissions are not accepted.
- The autograder in Gradescope is designed to give you feedback and help you in debugging. Passing the autograder tests does not guarantee a full mark in the assignment as the tests are not comprehensive and does not test all cases. It is your responsibility to make sure that the submitted code is correct and can handle all cases.
- Late submission is not acceptable.
- Using any AI tool for coding or debugging is not allowed.

- The submitted code should include the implementation of the functions listed in the tasks section without testing code. The autograder will call your functions from and pass parameters as described. The structure of the submitted file should be as follows

```
.globl crc_byte
.globl crc_array
.globl check_byte
.globl check_array

crc_byte:
# function implementation
     Jr $ra
crc_array:
# function implementation
     Jr $ra
check_byte:
# function implementation
     Jr $ra
Check_array:
# function implementation
     Jr $ra
```

- You can write a test code to test your function (but it should not be included in the submission). Here are some sample input and output for checking the code correctness:

Input	**Output (CRC)**
0x01	0xA7
“C” “O” “E” “3” “0” “1”	0xDE
0x01 0x02 0x03	0xFC
0x02	0xE9
0x01 0x02 0x03 0x04	0xC2