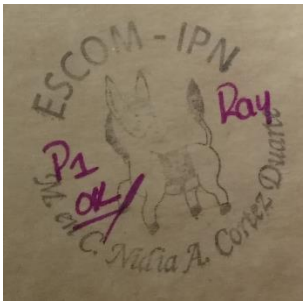
INSTITUTO POLITÉCNICO NACIONAL
ESCUELA SUPERIOR DE CÓMPUTO
INGENIERÍA EN SISTEMAS COMPUTACIONALES

PRÁCTICA 1

"MULTIPLICACIÓN EN GF(2^8)"

ALUMNO: MENDOZA RAMIREZ RAYMUNDO

PROFESOR: NIDIA ASUNCIÓN CORTEZ DUARTE



7CM1

Multiplicación

La multiplicación se efectúa a nivel de bits siguiendo el procedimiento estándar de una multiplicación convencional. No obstante, para obtener el resultado, se emplea una operación XOR con el fin de evitar el manejo de acarreos.

Reducción

La reducción puede llevarse a cabo de dos maneras distintas. En primer lugar, mediante una división entre polinomios, lo que implica operar con los coeficientes de cada variable. Por otro lado, se puede realizar utilizando el método de reducción por cajas. En ambos casos, se utiliza la compuerta XOR para eliminar elementos y evitar así la generación de acarreos.

Código

La función que se presenta a continuación solicita dos valores, expresados como cadenas de caracteres, y devuelve como resultado un número binario.

1. Primero, creamos una variable llamada product y la inicializamos en 0. Esta variable almacenará el resultado de la multiplicación.
2. Luego, recorremos cada dígito del segundo número (el número b) de derecha a izquierda utilizando un bucle for. Es decir, comenzamos desde el último dígito.
3. Si el dígito actual es un 1, realizamos una operación especial. Tomamos el primer número (a), que también está en formato binario, y lo convertimos a un número entero usando int(a, 2). Luego, lo desplazamos a la izquierda según la posición del dígito actual en el segundo número (i). Esto se hace usando el operador << que es un desplazamiento de bits.
4. El resultado de esta operación lo combinamos con el valor actual de product usando una operación XOR (^=).
5. Finalmente, devolvemos el resultado de la multiplicación, que está en formato binario, eliminando el prefijo 0b usando bin(product)[2:].

```
1. def multiply_binary(a, b):
2.     product = 0
3.     for i, digit in enumerate(reversed(b)):
4.         if digit == '1':
5.             product ^= int(a, 2) << i
6.     return bin(product)[2:]
7.
```

En la función **divide_binary(dividend, divisor)** toma como argumentos que son cadenas de caracteres que representan números binarios

1. Primero inicializamos dos variables: quotient (cociente) la cual se inicializa en 0 y remainder (residuo) se inicializa con el valor entero del dividend.
2. Convertimos el divisor también en un número entero para poder usarlo en cálculos matemáticos.
3. Calculamos la longitud del divisor en formato binario y la almacenamos en divisor_length.
4. Ahora entramos en un bucle while que continuará hasta que el tamaño del residuo (remainder) sea mayor o igual que el tamaño del divisor (divisor). Esto asegura que estemos realizando la división mientras el residuo pueda seguir siendo dividido por el divisor.
5. Dentro del bucle, calculamos el desplazamiento (shift) restando la longitud del divisor del tamaño del remainder. Esto nos da la cantidad de bits que necesitamos desplazar el divisor para que coincida con el residuo.
6. Luego, realizamos una operación XOR entre el remainder y el divisor desplazado a la izquierda. Esto se hace para eliminar una parte del remainder que corresponde al divisor, simulando así una división.
7. Continuamos repitiendo este proceso hasta que el tamaño del remainder ya no sea mayor o igual que el tamaño del divisor.
8. Finalmente, devolvemos el residuo resultante de la división en formato binario, eliminando el prefijo '0b'.

```
8. def divide_binary(dividend, divisor):
9.     quotient = 0
10.     remainder = int(dividend, 2)
11.     divisor = int(divisor, 2)
12.     divisor_length = len(bin(divisor)[2:])
13.     while len(bin(remainder)[2:]) >= len(bin(divisor)[2:]):
14.         shift = len(bin(remainder)[2:]) - divisor_length
15.         remainder ^= divisor << shift
16.
17.     return bin(remainder)[2:]
18.
```

La función **convert_hexa_to_binary** toma un único argumento, que es una cadena de caracteres representando un número en hexadecimal.

1. Primero se convierte el número hexadecimal a un número entero utilizando int(hexa, 16). Esto se hace mediante la función int, donde el segundo argumento 16 especifica que estamos tratando con un número en base 16 (hexadecimal).
2. Luego, el número entero resultante se convierte a binario utilizando la función bin(). Esto nos da una cadena de caracteres que representa el número en binario, pero con un prefijo 0b para indicar que está en binario.

3. Finalmente, para obtener solo la parte binaria sin el prefijo 0b, el código devuelve binary[2:], que es una porción de la cadena resultante, comenzando desde el tercer carácter hasta el final.

```
19.     def convert_hexa_to_binary(hexa):
20.         binary = bin(int(hexa, 16))
21.         return binary[2:]
22.
```

La función **binary_to_polynomial** toma un único argumento, que es una cadena de caracteres representando un número en formato binario.

1. Primero se inicializa una cadena de caracteres vacía llamada polynomial, que almacenará el polinomio resultante.
2. Luego, se itera sobre cada dígito del número binario utilizando un bucle for que recorre los índices de 0 a la longitud de binary menos 1.
3. Por cada dígito en el número binario, si es igual a '1', se determina el grado del término correspondiente en el polinomio. Esto se hace calculando la diferencia entre la longitud de binary y el índice actual menos 1. Esto nos da el grado del término.
4. Dependiendo del grado del término, se construye una representación del término en el polinomio. Si el grado es 0, se agrega simplemente "1" al polinomio. Si es 1, se agrega "x +", y si es mayor que 1, se agrega "x^grado +".
5. Después de iterar sobre todos los dígitos del número binario, puede haber un caso donde el polinomio termine con "+", lo cual es redundante. Por lo tanto, se verifica si el polinomio termina con "+", y si es así, se elimina los últimos 3 caracteres (que serían "+ ").
6. Finalmente, se devuelve el polinomio construido.

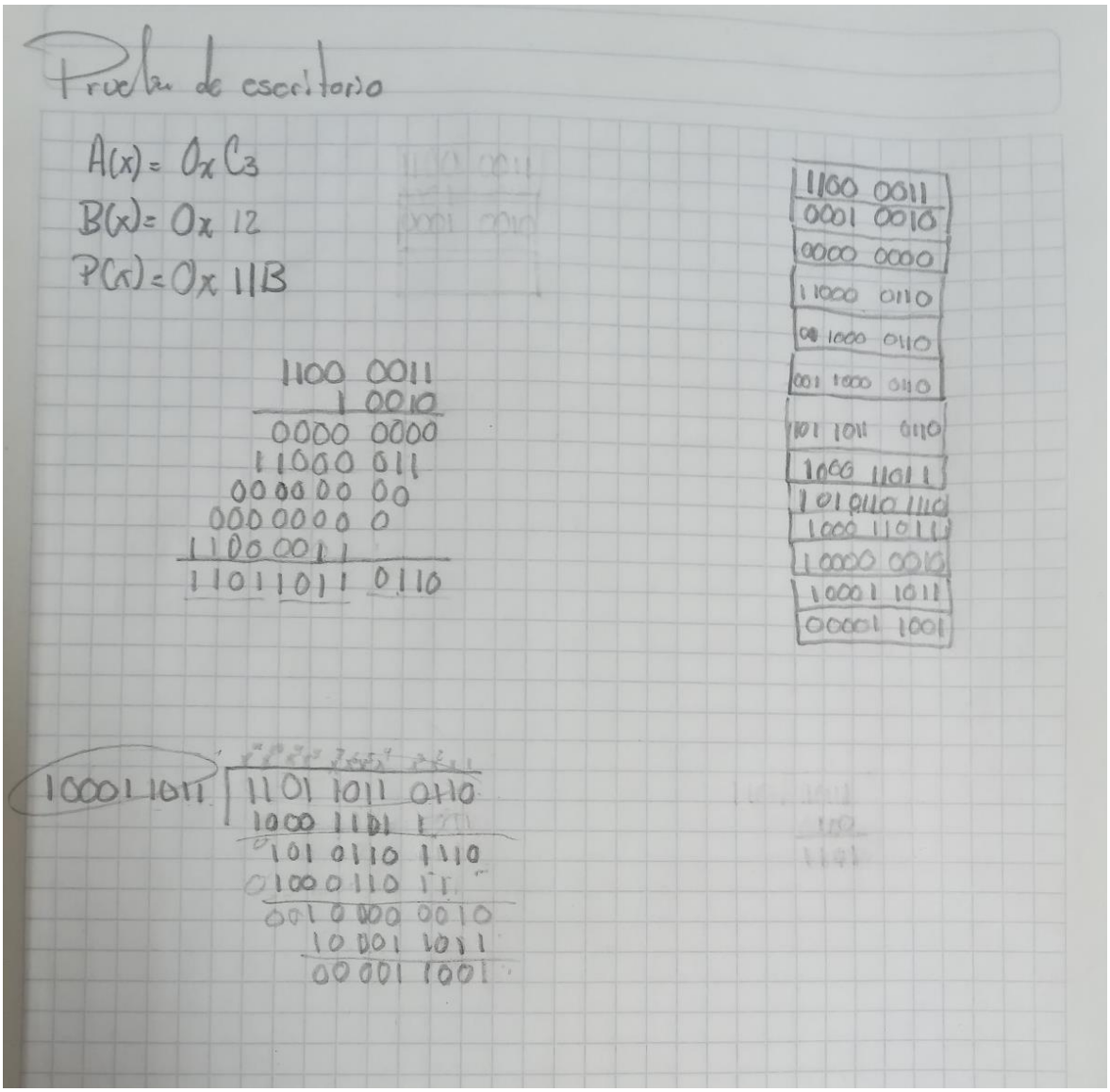
```
23.     def binary_to_polynomial(binary):
24.         polynomial = ""
25.         for i in range(len(binary)):
26.             if binary[i] == '1':
27.                 degree = len(binary) - 1 - i
28.                 if degree == 0:
29.                     polynomial += "1"
30.                 elif degree == 1:
31.                     polynomial += "x + "
32.                 else:
33.                     polynomial += "x^" + str(degree) + " + "
34.         if polynomial.endswith(" + "):
35.             polynomial = polynomial[:-3]
36.         return polynomial
```

La función **binary_to_hexa** toma un único argumento, que es una cadena de caracteres representando un número en formato binario.

1. Primero convierte el número binario a un número entero utilizando int(binario, 2). Esto se hace mediante la función int, donde el segundo argumento 2 especifica que estamos tratando con un número en base 2 (binario).
2. Una vez que tenemos el número entero equivalente, lo convertimos a hexadecimal utilizando format(decimal, 'X'). Esto nos da una cadena de caracteres que representa el número en hexadecimal.
3. Finalmente, se devuelve el número hexadecimal resultante.

```
37.     def binary_to_hexa(binario):
38.         decimal = int(binario, 2)
39.         hexadecimal = format(decimal, 'X')
40.         return hexadecimal
41.
```

Prueba de escritorio



Salidas

	0x	0x	0x	0x
A(x)	**C3**	**E0**	**A2**	**FF**
B(x)	**12**	**07**	**16**	**FF**
C(x)