Part 3 - Mysterious Cipher

Python

```
        .section .data
ciphertext:
        .asciz "ddbw"       # Hardcoded ciphertext (encrypted message)
num1:
        .long 2
num2:
        .long -5

        .section .text
        .globl main
main:
        # Load the address of ciphertext into %rdi for processing.
        lea ciphertext(%rip), %rdi
        movl num1(%rip), %eax
        movl num2(%rip), %ecx
        addl %ecx, %eax
        movb %al, %bl

decrypt_loop:
        movzbq (%rdi), %rax
        testb %al, %al
        je decrypt_done
        subb %bl, %al
        movb %al, (%rdi)
        inc %rdi
        jmp decrypt_loop

decrypt_done:
        # Reload the address of ciphertext to print the decrypted string.
        lea ciphertext(%rip), %rdi
        call puts           # Call external function 'puts' to display the
message.

        # Exit the program.
        movl $0, %eax
        ret
```

Breaking down the code label-by-label for analysis, starting with the constants:

```
Python
ciphertext:
       .asciz "ddbw"       # Hardcoded ciphertext (encrypted message)
num1:
       .long 2
num2:
       .long -5

       .section .text
       .globl main
```

ciphertext contains the cipher text that will be decoded later.
num1 and **num2** are long numbers that presumably will become the basis that we use to decipher the ciphertext.

```
Python
main:
       # Load the address of ciphertext into %rdi for processing.
       lea ciphertext(%rip), %rdi
       movl num1(%rip), %eax
       movl num2(%rip), %ecx
       addl %ecx, %eax
       movb %al, %bl
```

After the commented code, which explains the first line, the program moves the long values of both **num1** and **num2** into two registers - **%eax** and **%ecx**.
Then it adds **%ecx** to **%eax**, basically being the operation 2 + (-5) = -3
The final line moves the least significant bit in **%eax**, which in this case will be -3, into **%bl**.

```
Python
decrypt_loop:
       movzbq (%rdi), %rax
       testb %al, %al
       je decrypt_done
       subb %bl, %al
       movb %al, (%rdi)
       inc %rdi
       jmp decrypt_loop
```

Next is the loop.
The first step moves the current byte pointed to by **%rdi** into **%rax**.
By doing **testb %al, %al**, we are testing whether we've reached the end of **ciphertext** (in C, this is denoted by a null terminator, in Assembly, presumably a zero). If so, we jump to **decrypt_done.**
If not, we subtract **%bl** from **%al**. Recall that **%bl** stores -3, so this *essentially means we are actually shifting the ASCII code of the byte stored in **%al** up by 3.*
We move this shifted byte into (**%rdi**), which stores the decrypted byte in its respective position in the string.
After that, we do an increment. This makes **%rdi** point to the next character or byte of the string.
After that, we repeat this process until we reach the end of the string.

```
Python
decrypt_done:
	# Reload the address of ciphertext to print the decrypted string.
	lea ciphertext(%rip), %rdi
	call puts		# Call external function 'puts' to display the
message.

	# Exit the program.
	movl $0, %eax
	ret
```

The final label, **decrypt_done**, basically finishes everything.
First, it reloads the address of the ciphertext that we decrypted in the loop, and then calls an external function **puts** that will print the decrypted message.
Then it exits the program.

Decoded Message

Since we are shifting the ASCII values by three, this is what the decrypted message will be:

- d -> ef **g**
- d -> ef **g**
- b -> cd **e**
- w -> xy **z**

So the final decrypted message that will be printed at the end of the program is **ggez.**