CMPE310 – Spring 2025 – Haming Distance Project 1

Johann Mission / ID QU57259

a) First I considered structuring it like this:
    a. In section data, “enter string 1” and “enter string 2”, and then having prompts for two user string inputs.
    b. In section bss, allocating a variable for the sum of the hamming distance
    c. In start:
        i. Converting the strings into ASCII binary
        ii. Having a for loop iterating through the binary representations at the same time.
        iii. XOR’ing the binary values 2 to see if they match, and adding it to the hamming distance sum.
        iv. Then, print the values.
    d. Special cases, if the string was shorter, it would have the length of the shortest string and only iterate up to that.
b) On Friday 2/28 lab I was having substantial issues logging into GL. These issues have been occurring off and on since Thursday 2/27. I have been contacting DoIT and they seem to be useless, not only mentioning a 3 business day response time. I have notified the TAs during lab but my ability to work on the projects is limited.
c) From what I could do, I attempted to debug as much as I could using GDB, it helped with segmentation faults but not the logic/syntax errors.

d) After numerous failed attempts I got help from ChatGPT, which revised the code that I am submitting below.

```
section .data
    prompt1 db "Enter first string: ", 0
    prompt1_len equ $ - prompt1
    prompt2 db "Enter second string: ", 0
    prompt2_len equ $ - prompt2
    result_msg db "Hamming Distance: ", 0
    result_msg_len equ $ - result_msg
    newline db 10  ; Newline character

section .bss
    str1 resb 256
    str2 resb 256
    hamming_dist resb 4

section .text
    global _start

_start:
    ; Prompt for first string
    mov rax, 1          ; syscall: write
    mov rdi, 1          ; stdout
    mov rsi, prompt1    ; message
    mov rdx, prompt1_len
    syscall

    ; Read first string
    mov rax, 0          ; syscall: read
    mov rdi, 0          ; stdin
    mov rsi, str1       ; buffer
    mov rdx, 256        ; max length
    syscall

    ; Remove newline from first string
    call strip_newline

    ; Prompt for second string
    mov rax, 1
    mov rdi, 1
    mov rsi, prompt2
    mov rdx, prompt2_len
    syscall

    ; Read second string
    mov rax, 0
    mov rdi, 0
    mov rsi, str2
    mov rdx, 256
    syscall

    ; Remove newline from second string
    call strip_newline
```

```
    ; Compute Hamming distance
    call compute_hamming_distance

    ; Print result message
    mov rax, 1
    mov rdi, 1
    mov rsi, result_msg
    mov rdx, result_msg_len
    syscall

    ; Print the computed Hamming distance
    mov rdi, [hamming_dist]
    call print_number

    ; Print newline
    mov rax, 1
    mov rdi, 1
    mov rsi, newline
    mov rdx, 1
    syscall

    ; Exit program
    mov rax, 60
    xor rdi, rdi
    syscall

; Function to compute Hamming distance
compute_hamming_distance:
    xor rcx, rcx        ; Reset counter (Hamming distance)
    mov rsi, str1
    mov rdi, str2

loop_compare:
    mov al, [rsi]       ; Load character from str1
    mov bl, [rdi]       ; Load character from str2
    test al, al         ; Check for null terminator
    jz end_compare
    test bl, bl
    jz end_compare

    xor al, bl          ; Bitwise XOR (find differing bits)
    test al, al         ; If different, count
    jz skip_increment
    inc rcx             ; Increment Hamming distance

skip_increment:
    inc rsi             ; Move to next character
    inc rdi
    jmp loop_compare
```

```
skip_increment:

end_compare:
    mov [hamming_dist], rcx
    ret

; Function to remove newline character from string
strip_newline:
    mov rsi, str1       ; Start from first string
strip_loop:
    cmp byte [rsi], 10  ; Check for newline
    jz remove_newline
    test byte [rsi], 0  ; Check for null terminator
    jz end_strip
    inc rsi
    jmp strip_loop

remove_newline:
    mov byte [rsi], 0   ; Replace newline with null terminator
end_strip:
    ret

; Function to print a number
print_number:
    mov rax, 0          ; Clear RAX
    mov rsi, 10         ; Divisor for decimal conversion
    mov rbx, 0          ; Output buffer

convert_loop:
    mov rdx, 0          ; Clear RDX
    div rsi             ; Divide RAX by 10
    add dl, '0'         ; Convert remainder to ASCII
    push rdx            ; Store character
    inc rbx             ; Increase length
    test rax, rax       ; Check if quotient is 0
    jnz convert_loop

print_loop:
    pop rax             ; Get character
    mov rdi, 1          ; stdout
    mov rsi, rsp        ; Buffer
    mov rdx, 1          ; Length
    syscall
    dec rbx
    jnz print_loop
    ret
```

Foo bar output

```
[m376@linux6 proj1]$ nasm -f elf64 hamming4.asm -o hamming4.o
[m376@linux6 proj1]$ ld hamming4.o -o hamming4
[m376@linux6 proj1]$ hamming4
Enter first string: foo
Enter second string: bar
Hamming Distance: 8
[m376@linux6 proj1]$
```

This is a test of the emergency broadcast:

```
[m376@linux6 proj1]$ nasm -f elf64 hamming4.asm -o hamming4.o
[m376@linux6 proj1]$ ld hamming4.o -o hamming4
[m376@linux6 proj1]$ hamming4
Enter first string: this is a test
Enter second string: of the emergency broadcast
Hamming Distance: 38[m376@linux6 proj1]$
```

Unfortunately I am still running into errors with some custom inputs, either it’s printing the completely wrong number, garbage, or nothing at all.