## Concept

My initial thought on this assignment was to create a while loop to compare which string was smaller, then compare the two and utilize the smaller distance (measured through a counter) to calculate the hamming distance.

My algorithm works by taking the distance of both strings, then calculating the hamming distance of the two strings so long as they are equal in length. If they are not equal in length, it breaks away to calculate the distance of the change in length, though I deemed this feature unnecessary after re-reading the assignment requirements and stopped halfway in its creation. It then takes the calculated hamming distance, adds ‘0’ to it, and prints it as a single character. If the character is between 0-9, it prints it. If the character is greater, it prints another ASCII character equal to 48 + the number’s value.

The parts of my algorithm that actually work to the result are as follows, the calculation of the hamming distance, and the break by calculating when a bit is null through comparing it to byte 0.

Code

```
section .data
	foo db "1111", 0 ;the word
	lenFoo equ $ - foo ;length of the word
	bar db "0000", 0  ;the other word
	lenBar equ $ - bar ;length of the other word

	newline db 0x0A, 0 ; variables i used for testing output statements. no longer necessary
;	testVar db "Test\t "
;	testLen equ $ - testVar

	hamming dw 0, 0
	count dw 0, 0

section .bss
	distance resb 5

section .text
	global _start

_start:
;	mov eax, 4
;	mov ebx, 1
;	mov ecx, testVar
;	mov edx, testLen
;	int 0x80
;
;	mov eax, 4
;	mov ebx, 1
;	mov ecx, newline
;	mov edx, 1
;	int 0x80

	xor esi, esi
	xor ecx, ecx
	xor edx, edx
	xor bl, bl
	jmp largerFoo

largerFoo:
	mov al, [foo + edx]
	cmp al, byte 0
	je largerBar
```

```
        inc edx
        jmp largerFoo


largerBar:
        mov ah, [bar + esi]
        cmp ah, byte 0
        je label

        inc esi
        jmp largerBar


label:
        cmp ecx, 255
        jge out

        mov al, [foo + ecx]
        mov ah, [bar + ecx]

        cmp al, byte 0
        je sumLoop

        cmp ah, byte 0
        je sum2


        cmp al, ah
        je equal

        inc bl


equal:
        add ecx, 1
        cmp al, ah

        jmp label


;bunch of functions because i thought I had to add the longer string's extra length to the
distance. I don't but this works so i'm not changing it!!!
sum2:
        cmp esi, edx
```

```
        je finale2

        sub esi, 1

finale2:
        cmp ecx, edx
        je out

        inc ecx

sumLoop:
        xor ecx, ecx
        cmp esi, edx
        je finale

        sub edx, 1

finale:
        cmp ecx, esi
        je out

        inc ecx

out:
        add bl, '0' ;converts bl into a string
        mov [hamming], bl ;moves value of "bl" into hamming distance

        mov eax, 4
        mov ebx, 1
        mov ecx, hamming
        mov edx, lenFoo
        int 0x80

        mov eax, 4
        mov ebx, 1
        mov ecx, newline
        mov edx, 1
        int 0x80

        mov eax, 1
        xor ebx, eax
        int 0x80
```

Output

Test for “foo bar” - hamming distance of 8

```
[cward6@linux6 lab2]$ nasm -felf64 hamming.asm && ld hamming.o && ./a.out
8
[cward6@linux6 lab2]$
```

Returns 8 as expected

Test for “this is a test” “of the emergency broadcast” - hamming distance of 38

```
[cward6@linux6 lab2]$ nasm -felf64 hamming.asm && ld hamming.o && ./a.out
V
[cward6@linux6 lab2]$
```

While it may appear to have a wrong output, I converted the output for a single character to be readable for outputs 0-9 by adding ‘0’ to the output. “V” is 38 in ASCII, thus it calculated the correct hamming distance.

Test for “0000” and “1111”

```
[cward6@linux6 lab2]$ nasm -felf64 hamming.asm && ld hamming.o && ./a.out
4
[cward6@linux6 lab2]$
```