Danielle Burton

CMPE 310

3/6/25

Project 1

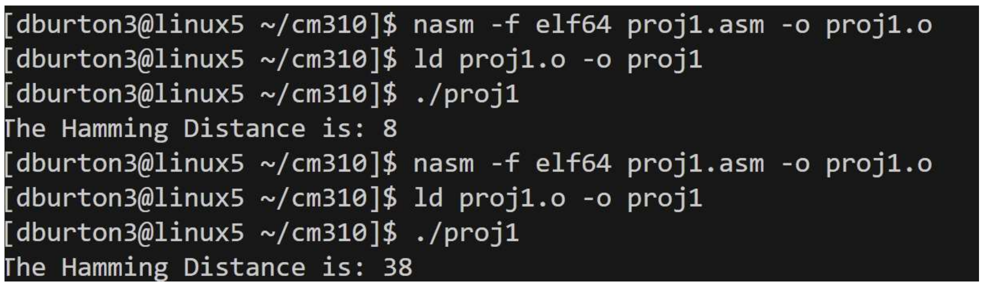
Hamming Distance lab report

In the project, the goal was to calculate the hamming distance of two different strings. The hamming distance is the number of bits that differ between the strings. To do this, we wanted to take in two strings and turn them into their binary form. Then, we could use an XOR comparison to figure out which bits were different. The XOR would yield a 1 when the bits were different. We were them able to count the number of bits that differed and print the number out as the distance.

The code stores the XOR string in one of the registers. I then increment through the indexes of the register and compare it to a value of 1. If the value is one, then the distance is incremented by 1. This approach works, as the xor does the comparison between the two strings, and I am able to easily count the number of bits that differed.

This was only semi successful. I was able to count the number of bits that differed, when the two strings were already in binary. This worked for both the ‘foo’ and ‘bar’ comparison, as well as with the longer comparison. I was unable to figure out how to convert a string into its binary values. In my code, there are already variables for the binary of the four strings, and all that is needed is to switch the variable names of str1 and str2 with str3 and str4, to test the other pair.

Below is the code and the screenshots of two different outputs. The first was comparing ‘foo’ and ‘bar’. The second was comparing ‘this is a test’ and ‘of the emergency broadcast’



```
[dburton3@linux5 ~/cm310]$ nasm -f elf64 proj1.asm -o proj1.o
[dburton3@linux5 ~/cm310]$ ld proj1.o -o proj1
[dburton3@linux5 ~/cm310]$ ./proj1
The Hamming Distance is: 8
[dburton3@linux5 ~/cm310]$ nasm -f elf64 proj1.asm -o proj1.o
[dburton3@linux5 ~/cm310]$ ld proj1.o -o proj1
[dburton3@linux5 ~/cm310]$ ./proj1
The Hamming Distance is: 38
```

```
;CMPE 310 Assembly Project 1
;Danielle Burton
;Calculate the Hamming distance between 2 strings

;Constants are defined
section .data
  str1 db "0110 0110 0110 1111 0110 1111", 0 ;foo
  str2 db "0110 0010 0110 0001 0111 0010" , 0 ;bar

  ;this is a test
  str3 db "01110100 01101000 01101001 01110011 00100000 01101001 01110011
00100000 01100001 00100000 01110100 01100101 01110011 01110100",0 ; this is a test
  str4 db "01101111 01100110 00100000 01110100 01101000 01100101 00100000
01100101 01101101 01100101 01110010 01100111 01100101 01101110 01100011
01111001 00100000 01100010 01110010 01101111 01100001 01100100 01100011
01100001 01110011 01110100",0 ;of the emergency broadcast
  ;of the emergency broadcast

  stat db "The Hamming Distance is: ", 0
  stlen equ $-stat ; statement length
  newline db 0x0A

;Reserving space to use for future data
section .bss
  dist resb 4 ; stores 4 byte for  dist
  dist_len resb 1
section .text
  global _start

_start:

;*******************************************************************
;Initialize registers
  mov ecx, 0 ;i
  mov edx, 0 ;dist

  jmp defCompare ;Not needed but clarity
;*******************************************************************
;Compares strings
defCompare:
  mov al, byte [str1 + ecx] ; Gets a specific bit of str1
  mov bl, byte [str2 + ecx] ; Gets a specific bit of str2

  ; Check for end of strings
```

```
   cmp al, 0
   je defExit
   cmp bl, 0
   je defExit

   ; XOR the bits
   xor al, bl

   ; Check if the XOR result is equal to 1 (bits differ)
   cmp al, 1
   je defCount

   jmp defForLoop ;Back to the loop :)
;*********************************************************************
;Loops through and calls compare.
;checks if we have reached the end of the string
defForLoop:
   inc ecx ; Increment

   ; Check for end of strings
   cmp byte [str1 + ecx], 0
   je defExit
   cmp byte [str2 + ecx], 0
   je defExit

   ; Else
   jmp defCompare
;***************************************************
;Increments distance and goes to for loop
defCount:
   inc edx
   jmp defForLoop
;*********************************************
;prints and finishes the program
defExit:
   ;adds a terminating bit to the end of distance
   mov ecx, dist
   add ecx, 3
   mov byte [ecx], 0

   ; Convert the distance to ASCII (right-to-left in the buffer)
   mov eax, edx        ;eax=dist
   mov ebx, 10         ; Set divisor to 10 (decimal system)
   defConvert:
    mov edx, 0        ;clear edx
    div ebx           ; Divide eax by 10, quotient in eax, remainder in dl
    add dl, '0'       ; Convert remainder to ASCII
    dec ecx
    mov [ecx], dl     ; Store the ASCII character
    test eax, eax     ;if eax != 0
    jnz defConvert    ;keep doing division

  ; Calculate the length of the distance string
  sub ecx, dist         ; Calculate the length of the string
  mov [dist_len], cl    ; Store the length of the distance string

  ; Print "The Hamming Distance is: "
  mov eax, 4
  mov ebx, 1
  mov ecx, stat
  mov edx, stlen; Length of the stat string
  int 0x80

  ; Print statement
  mov eax, 4
  mov ebx, 1
  mov ecx, dist
  mov edx, 4 ; Corrected length to 1 byte
  int 0x80

  ; Prints a new line
  mov eax, 4
  mov ebx, 1
  mov ecx, newline
  mov edx, 1
  int 0x80

  ; Exit program commands
  mov ebx, 0
  mov eax, 1
  int 0x80
;*******************************************************************
```