```
;    nasm -f elf -g -F stabs Vchiper Encode .asm
;    gcc -m32 Vchiper Encode .o -o Vchiper Encode

SECTION .bss
TextLenght EQU 1024          ; Define length of a line of  text data
Text resb TextLenght         ; Define array

SECTION .data
keyword: db "CRYPTOGRAM"
len: equ $ - keyword
CountMsg:   db 10,"%d characters read",10,0
Count:      dd 0                    ; Character count

SECTION .text
global main
extern getchar
extern putchar
extern printf

main:
   nop                  ; This no-op keeps the debugger happy
   push ebp
   mov ebp, esp             ; Setting the base pointer
   push ebx
   xor ebx, ebx             ; Resetting the register to 0

Read:
   call getchar             ; Read a character

   cmp eax, -1              ; End of input file?
   je Done                  ; if return 0 or under, we at EOF

   mov BYTE[Text+ebx], al     ; store the characters to the array
   inc ebx                  ; increasing ebx

   inc dword [Count]          ; Increment count
   cmp dword [Count],199      ; Check for buffer overflow

   jmp Read                 ; read next character

Done:
   mov BYTE[Text+ebx+1], -1    ; setting the end of the file
   xor ebx, ebx             ; resetting the register
   xor eax, eax             ; reset the register

Encrypt:
   mov al, BYTE[Text+ebx]     ; setting al
```

```
    cmp al, -1              ; End of input file?
    je Exit

    cmp al, 'A'
    jb Write                ; print if char is below 'A'
    cmp al, 'z'
    ja Write                ; print if char is above 'z'

    cmp al, 'Z'             ; if char is below or equal to 'Z'
    jbe Uppercase           ; process as uppercase
    cmp al, 'a'             ; if char is above or equal to 'a'
    jae Lowercase           ; process as lowercase

    jmp Write               ; else print special characters

Uppercase:
    push ebx                ; save the counter
    push eax                ; save the current char
    mov eax, ebx            ; setting eax adress to be ebx adress
    mov edx, 0              ; clearing the remainder
    mov ebx, len            ; setting the divisor
    div ebx                 ; division
    xor ebx, ebx            ; resetting the register
    mov bl, BYTE[keyword+edx]   ; setting bl to be char in keyword
    sub ebx, 65             ; subtract with 65, to work within
                        ; the range 0→25, ebx has the shift value
    pop eax                 ; restoring the letter
    add eax, ebx            ; adding to get the value to shift
    mov edx, ebx            ; let edx hold the shifting number
    pop ebx                 ; restoring the counter
    cmp eax, 'Z'            ; if eax is below or equal to 'Z'
    jbe Write               ; print the char
    sub eax, 26             ; else correct the value
    jmp Write               ; and print

Lowercase:
    push ebx                ; save the counter
    push eax                ; save the current char
    mov eax, ebx            ; setting eax adress to be ebx adress
    mov edx, 0              ; clearing the remainder
    mov ebx, len            ; setting the divisor
    div ebx                 ; division
    xor ebx, ebx            ; resetting the register
    mov bl, BYTE[keyword+edx]   ; setting bl to be char in keyword
    sub ebx, 65             ; subtract with 65, to work within
                        ; the range 0→25, ebx has the shift value
    pop eax                 ; restoring the letter
```

```
    add eax, ebx            ; adding to get the value to shift
    mov edx, ebx            ; let edx hold the shifting number
    pop ebx                 ; restoring the counter
    cmp eax, 'z'            ; if eax is below or equal to 'z'
    jbe Write               ; print the char
    sub eax, 26             ; else correct the value

Write:
    push eax                ; push eax to print
    call putchar
    add esp, 4              ; Clean stack, one parm

    push dword [Count]      ; Value of Count

    xor eax, eax            ; reset the register
    inc ebx                 ; increasing ebx
    jmp Encrypt

Exit:
    push CountMsg           ; Format string
    call printf
    add esp,8               ; Clean stack, two parms
    mov esp, ebp            ; restore stack pointer
    pop ebp                 ; same as "leave" operation
    ret                     ; return
```

```
;    nasm -f elf -g -F stabs Vchiper Decode .asm
;    gcc -m32 Vchiper Decode .o -o Vchiper Decode

SECTION .bss
TextLenght EQU 1024          ; Define length of a line of  text data
Text resb TextLenght         ; Define array

SECTION .data
keyword: db "CRYPTOGRAM"
len: equ $ - keyword
CountMsg:   db 10,"%d characters read",10,0
Count:      dd 0             ; Character count

SECTION .text
global main
extern getchar
extern putchar
extern printf

main:
   nop                  ; This no-op keeps the debugger happy
   push ebp
   mov ebp, esp           ; Setting the base pointer
   push ebx
   xor ebx, ebx           ; Resetting the register to 0

Read:
   call getchar           ; Read a character

   cmp eax, -1            ; End of input file?
   je Done                ; if return 0 or under, we at EOF

   mov BYTE[Text+ebx], al     ; store the characters to the array
   inc ebx                ; increasing ebx

   inc dword [Count]         ; Increment count
   cmp dword [Count],199      ; Check for buffer overflow

   jmp Read               ; read next character

Done:
   mov BYTE[Text+ebx+1], -1    ; setting the end of the file
   xor ebx, ebx           ; resetting the register
   xor eax, eax           ; reset the register

Encrypt:
   mov al, BYTE[Text+ebx]     ; setting al
```

```
    cmp al, -1              ; End of input file?
    je Exit

    cmp al, 'A'
    jb Write                ; print if char is below 'A'
    cmp al, 'z'
    ja Write                ; print if char is above 'z'

    cmp al, 'Z'             ; if char is below or equal to 'Z'
    jbe Uppercase           ; process as uppercase
    cmp al, 'a'             ; if char is above or equal to 'a'
    jae Lowercase           ; process as lowercase

    jmp Write               ; else print special characters

Uppercase:
    push ebx                ; save the counter
    push eax                ; save the current char
    mov eax, ebx            ; setting eax adress to be ebx adress
    mov edx, 0              ; clearing the remainder
    mov ebx, len            ; setting the divisor
    div ebx                 ; division
    xor ebx, ebx            ; resetting the register
    mov bl, BYTE[keyword+edx]   ; setting bl to be char in keyword
    sub ebx, 65             ; subtract with 65, to work within
                        ; the range 0→25, ebx has the shift value
    pop eax                 ; restoring the letter
    sub eax, ebx            ; subtracting to get the value to shift
    mov edx, ebx            ; let edx hold the shifting number
    pop ebx                 ; restoring the counter
    cmp eax, 'A'            ; if eax is above or equal to 'A'
    jae Write               ; print the char
    add eax, 26             ; else correct the value
    jmp Write               ; and print

Lowercase:
    push ebx                ; save the counter
    push eax                ; save the current char
    mov eax, ebx            ; setting eax adress to be ebx adress
    mov edx, 0              ; clearing the remainder
    mov ebx, len            ; setting the divisor
    div ebx                 ; division
    xor ebx, ebx            ; resetting the register
    mov bl, BYTE[keyword+edx]   ; setting bl to be char in keyword
    sub ebx, 65             ; subtract with 65, to work within
                        ; the range 0→25, ebx has the shift value
    pop eax                 ; restoring the letter
```

```
    sub eax, ebx            ; subtracting to get the value to shift
    mov edx, ebx            ; let edx hold the shifting number
    pop ebx                 ; restoring the counter
    cmp eax, 'a'            ; if eax is above or equal to 'a'
    jae Write               ; print the char
    add eax, 26             ; else correct the value

Write:
    push eax                ; push eax to print
    call putchar
    add esp, 4              ; Clean stack, one parm

    push dword [Count]      ; Value of Count

    xor eax, eax            ; reset the register
    inc ebx                 ; increasing ebx
    jmp Encrypt

Exit:
    push CountMsg           ; Format string
    call printf
    add esp,8               ; Clean stack, two parms
    mov esp, ebp            ; restore stack pointer
    pop ebp                 ; same as "leave" operation
    ret                     ; return
```

The difference between the two files are small, it is different in the *Uppercase* and *Lowercase* sections. The difference is that I check that eax is below or equal to ‘Z’ and ‘z’ in the **encode** file and if eax is above or equal to ‘A’ and ‘a’ in the **decode** file. The other thing is that I add the shift value in the **encode** and subtract in the **decode**.

I did not figure out how to do both encrypt and decrypt in one file so I had to make two. If there is an easier way to do this solution, is it possible to get that solution?

~ — ssh -YC tanusanr@vor.ifi.uio.no
-bash-4.1$ cat input.txt
Education is the passport to the future, for tomorrow belongs to those who prepare for it today. - Malcom X
-bash-4.1$ cat output.txt
-bash-4.1$ cat output2.txt
-bash-4.1$ nasm -f elf -g -F stabs VchiperDecode.asm
-bash-4.1$ gcc -m32 VchiperDecode.o -o VchiperDecode
-bash-4.1$ nasm -f elf -g -F stabs VchiperEncode.asm
-bash-4.1$ gcc -m32 VchiperEncode.o -o VchiperEncode
-bash-4.1$ ./VchiperEncode < input.txt > output.txt
-bash-4.1$ cat output.txt
Gusrthofn kj ias gaeugmgm zf fjv unhaie, wmg hudodtfu usrfnsu rd hnfsq nfd dxvpmtv uhf zt vfbpr. - Yccadf D
108 characters read
-bash-4.1$ ./VchiperDecode < output.txt > output2.txt
-bash-4.1$ cat output2.txt
Education is the passport to the future, for tomorrow belongs to those who prepare for it today. - Malcom X
108 characters read
-bash-4.1$ cat input.txt
Education is the passport to the future, for tomorrow belongs to those who prepare for it today. - Malcom X
-bash-4.1$