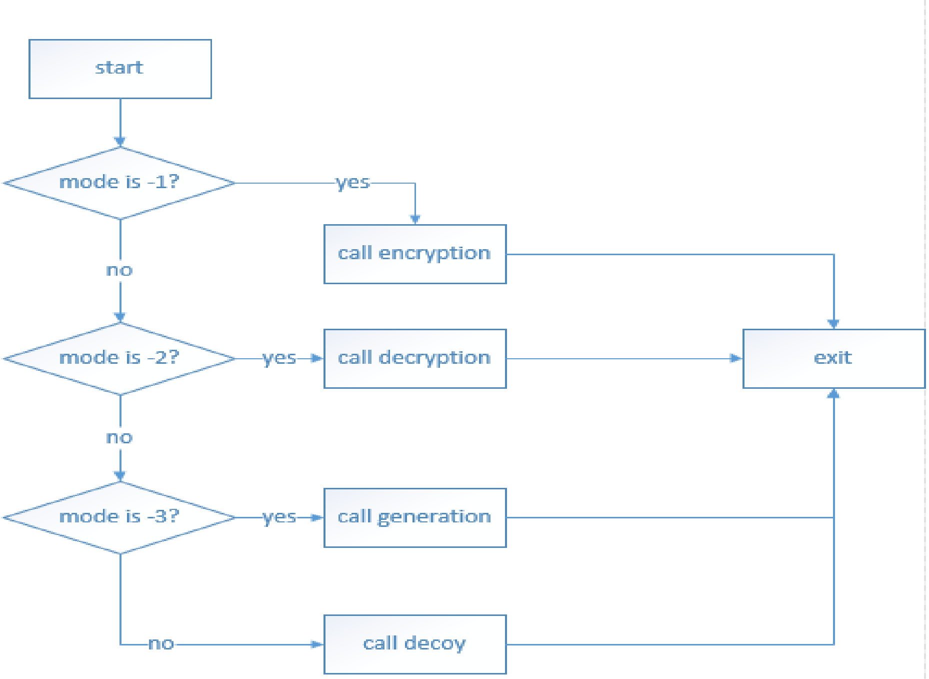
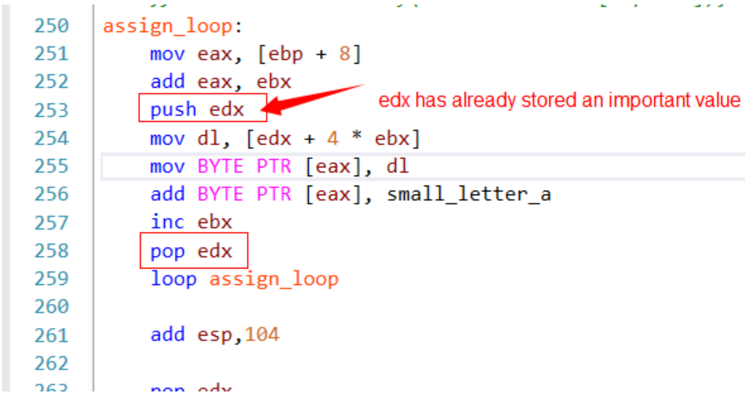
1. The last parameter stored in [ebp+8] is the mode reference.



2. When there is no usable register, I will push a register for temporary usages , then finish all operations and pop the register finally.



```
assign_loop:
    mov eax, [ebp + 8]
    add eax, ebx
    push edx
    mov dl, [edx + 4 * ebx]
    mov BYTE PTR [eax], dl
    add BYTE PTR [eax], small_letter_a
    inc ebx
    pop edx
    loop assign_loop

    add esp,104

    pop edx
```

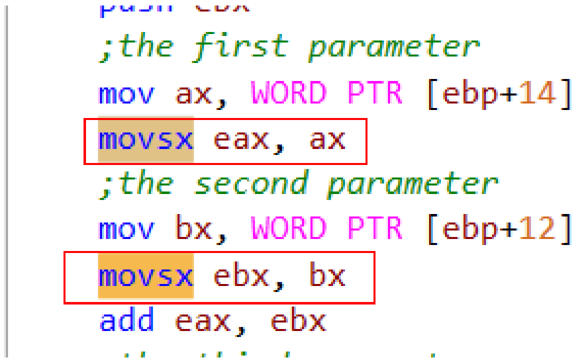
edx has already stored an important value

3. I think my implementation is already very good, so I have nothing to implement differently. Maybe do something for extra credit.. but have no idea about that.
4. And I think that the most challenge is how to make my coder shorter. So, I should understand the entire process completely. For example, how to multiplex the ‘encryption’ code when implementing ‘decryption‘, how to shuffle an array randomly with little effort.
5. https://www.youtube.com/watch?v=75gBFiFtAb8 Youtube channel
   and make friends with other people who master the Assembly Language

Extra Credit:

The main challenge is the boundary calculation, for example -32768 + -32768.

I use the ‘movsx’ directive to tackle the problem.



```
push ebx
;the first parameter
mov ax, WORD PTR [ebp+14]
movsx eax, ax
;the second parameter
mov bx, WORD PTR [ebp+12]
movsx ebx, bx
add eax, ebx
```

Encryption:

```
c = message begin address

    while c’s content is not 0

    c’s content = key[c’s content-’a’]

    c = c’s next byte address
```

Decryption:

```
get reverse key from original key

call encryption
```

Generation:

The key problem is to shuffle a array [0,1,2,....,25] randomly.

```
arr = [0,1,2,....,25]

left_count = 26

while left_count > 1

    i = random range value in [0, left_count)

    swap arr[i], arr[left_count-1]
```

```
left_count = left_count - 1
```